

LINEARITY IN HIGHER-ORDER RECURSION SCHEMES

Pierre Clairambault

CNRS & ENS Lyon

Charles Grellois

Aix Marseille Université

Andrzej Murawski

University of Oxford

IFIP WG 2.2

Unsafe Grammars and Panic Automata

Tyodor Knapik¹, Damian Niewiński^{2,*},
Paweł Urzyczyn^{2,3,†}, and Igor Walukiewicz^{3,...}

¹ Université de la Nouvelle Calédonie
knapi@univ-nc.nc
² Institute of Informatics, Warsaw University
{niewinski, urzyczyn}@mimuw.edu.pl
³ CNRS LARIH, Université Bordeaux
ig@labri.fr

Abstract. We show that the problem of checking if an infinite tree generated by a higher-order grammar of level 2 (hyperalgebraic) satisfies a given μ -calculus formula (or, equivalently, if it is accepted by an alternating parity automaton) is decidable, actually 2-EXPTIME-complete. Consequently, the monadic second-order theory of any hyperalgebraic tree is decidable, so that the safety restriction can be removed from our previous decidability result. The last result has been independently obtained by Ashby, de Miranda and Ong. Our proof goes via a characterization of possibly unsafe second-order grammars by a new variant of higher-order pushdown automata, which we call *panic automata*. In addition to the standard pop, and pop₂ operations, these automata have an option of a destructive move called *panic*. The model-checking problem is then reduced to the problem of deciding the winner in a parity game over a suitable 2nd order pushdown system.

1 Introduction

Context-free tree grammars constitute the basic level in an infinite hierarchy of higher-order grammars introduced by W. Danan [8] (built on the earlier ideas of [11]). Courcelle [6] proved decidability of the monadic second-order (MSO) theory of any tree generated by an *algebraic* (context-free) tree grammar. Later Knapik *et al.* [13, 14] attempted to extend this decidability result to all levels of the Danan hierarchy. This has been achieved partially, namely with an additional syntactic restriction imposed on the grammars, called *safety*: the MSO theory of any tree generated by a safe grammar of level n is decidable.

Higher-order grammars can be seen as program schemes, where functions can take higher-order arguments. The tree generated by such a grammar describes completely the semantics of the program scheme. Thus decidability of

* Partly supported by KBN Grant 4 T11C 042 25.

** Partly supported by KBN Grant 3 T11C 002 27.

† The 2nd and the 4th author were also supported by the EC Research Training Network GINex.

L. Ciarelli *et al.* (Eds.): ICALP 2005, LNCS 3586, pp. 1449–1461, 2005.
© Springer-Verlag Berlin Heidelberg 2005

On model-checking trees generated by higher-order recursion schemes

C.-H. L. Ong^{*}
Oxford University Computing Laboratory

Abstract

We prove that the modal μ -calculus model-checking problem for (ranked and ordered) node-labeled trees that are generated by order- n recursion schemes (whether safe or not, and whether homogeneously typed or not) is Π -EXPTIME complete, for every $n \geq 0$. It follows that the monadic second-order theories of these trees are decidable. There are three major ingredients. The first is a certain transference principle from the tree generated by the scheme – the value tree – to an auxiliary computation tree, which is itself a tree generated by a related order-0 recursion scheme (equivalently, a regular tree). Using innocent game semantics in the sense of Hyland and Ong, we establish a strong correspondence between paths in the value tree and traversals in the computation tree. This allows us to prove that a given alternating parity tree automaton (APT) has an (accepting) run-tree over the value tree iff it has an (accepting) traversal-tree over the computation tree. The second ingredient is the simulation of an (accepting) traversal-tree by a certain set of annotated paths over the computation tree; we introduce traversal-simulating APT as a recognising device for the latter. Finally, for the complexity result, we prove that traversal-simulating APT enjoy a succinctness property: for deciding acceptance, it is enough to consider run-trees that have a reduced branching factor. The desired bound is then obtained by analysing the complexity of solving an associated (finite) acceptance parity game.

1. Introduction

What classes of finitely-presentable infinite-state systems have decidable monadic second-order (MSO) theories? This is a basic problem in Computer-Aided Verification that is important to practice because standard temporal logics such as LTL, CTL and CTL* are embeddable in MSO logic. One of the best known examples of such a class are the regular trees as studied by Rabin in 1969. A notable advance occurred some fifteen years later, when Muller and

* www.comlab.ox.ac.uk/luke.ong/index.html

Shupp [13] proved that the configuration graphs of pushdown systems have decidable MSO theories. In the 90's, as finite-state technologies matured, researchers embraced the challenges of software verification. A highlight from this period was Caucal's result [5] that prefix-recognisable graphs have decidable MSO theories. In 2002 a flurry of discoveries significantly extended and unified earlier developments. In a FOSSACS'02 paper [11], Knapik, Niewiński 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. They showed that for every $n \geq 0$, trees generated by order- n safe schemes are exactly those that are accepted by order- n pushdown automata; further they have decidable MSO theories. Later in the year at MFCS'02 [6], 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 pushdown automata.

Knapik *et al.* [11] have asked if the safety assumption is really necessary for their MSO decidability result. A partial answer has recently been obtained by Ashby, de Miranda and Ong; they showed at TLCA'05 [2] that all trees up to order 2, whether safe or not, have decidable MSO theories. Independently, Knapik, Niewiński, Urzyczyn and Walukiewicz obtained a sharper result: they proved at ICALP'05 [12] that the modal μ -calculus model-checking problem for trees generated by order-2 recursion schemes (whether safe or not) is 2-EXPTIME complete. In this paper we give a complete answer to the question:

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

Our approach is to transfer the algorithmic analysis from the tree generated by a recursion scheme, which we call *value tree*, to an auxiliary 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

Types and Higher-Order Recursion Schemes for Verification of Higher-Order Programs

Naoki Kobayashi
Tohoku University
koba@ent.tohoku.ac.jp

Abstract

We propose a new verification method for temporal properties of higher-order functional programs, which takes advantage of Ong's recent result on the decidability of the model-checking problem for higher-order recursion schemes (HORS's). A program is transformed to an HORS that generates a tree representing all the possible event sequences of the program, and then the HORS is model-checked. Unlike most of the previous methods for verification of higher-order programs, our verification method is sound and complete. Moreover, this new verification framework allows a smooth integration of abstract model-checking techniques into verification of higher-order programs. We also present a type-based verification algorithm for HORS's. The algorithm can deal with only a fragment of the properties expressed by modal μ -calculus, but the algorithm and its correctness proof are (roughly) much simpler than those of Ong's game-semantics-based algorithm. Moreover, while the HORS model checking problem is Π -EXPTIME in general, our algorithm is linear in the size of HORS, under the assumption that the sizes of types and specifications are bounded by a constant.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software-Program Verification, F.3.1 [Logic and Meaning of Programs]: Specifying and Verifying and Reasoning about Programs

General Terms: Languages, Verification

1. Introduction

With the increasing importance of software reliability, program verification techniques have been studied extensively. There are still limitations in the current verification technology, however. Software model checking [3–5] has been mainly applied to imperative programming languages, and applications to programming languages with more dynamic control, such as higher order languages with dynamic allocation of resources (such as heap memory), have been limited. For higher order programs, type systems have been recognized as effective techniques for program verification. However, they either require explicit type annotations (as in dependent type systems), or suffer from many false alarms.

In this paper, we propose a novel verification technique for temporal properties of higher-order programs. We consider the prob-

lem of resource usage verification [19] for higher-order functional languages with dynamic resource creation and access primitives. The goal of the verification is to check that each dynamically created resource is accessed in a proper manner (like “an opened file is eventually closed, and it is not read or written after being closed”). Assertion-based model-checking problems (like “ $X > 0$ holds at program point p ”) can also be recasted as the resource verification problem, by regarding an assertion failure as an access to a global resource. (For example, “assert(b)” can be transformed into “if b , then skip, else fail,” where fail is an action to the global resource. Then the problem of checking lack of assertion failures is reduced to the resource usage verification problem of checking whether the fail action occurs.)

Our verification technique is built on the recent result on model checking of higher-order recursion schemes (HORS's, for short) [29]. A higher order recursion scheme is a grammar for describing an infinite tree. HORS is a generalization of regular tree grammars; they are described by HORS's of order 0. Ong [29] has recently shown the decidability of the problem of checking whether the infinite tree generated by G satisfies φ , given a modal μ -calculus formula φ and an HORS G .

The first main idea of this paper is to transform a higher-order functional program into an HORS that produces an infinite tree, each of whose path (from the root) corresponds to a possible access sequence to each resource. By the transformation, the problem of checking a regular property of resource-wise access behaviors of a functional program is reduced to that of checking the corresponding regular property of the infinite tree generated by the HORS. The latter can be solved by Ong's model checking algorithm for HORS [29]. For programs having only resources and functions as values, the transformation into HORS is achieved by CPS conversion and λ -lifting, along with an additional trick to extract “resource wise” access behavior. For programs with ordinary values (such as integers), we can apply the technique of predicate abstractions and counter-example-guided abstraction refinement. The resulting verification framework is sketched in Figure 1. Given a source program, we first apply CPS conversion and λ -lifting to get a system of top-level function definitions (Step 1). We then apply predicate abstractions to get a higher order boolean program (Step 2). It is then converted to HORS (Steps 3 and 4), and the HORS is model-checked (Step 5). If the model checking fails, a counterexample is investigated, and the abstraction is refined (Steps 6 and 7).

The second main idea of this paper is to use types for model-checking HORS, instead of Ong's algorithm based on game semantics. For a fragment of modal μ -calculus (for describing safety properties, which are sufficient for the purpose of resource usage verification), we develop an intersection type system that is sound and complete, in the sense that an HORS is well-typed if and only if the HORS satisfies the given property. Thus, a type inference algo-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made for distribution for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to list, requires prior specific permission and/or a fee.
ICP'05, January 18–23, 2005, Alexandria, Georgia, USA.
Copyright © 2005 ACM 978-1-4658-379-2/05/01...\$5.00

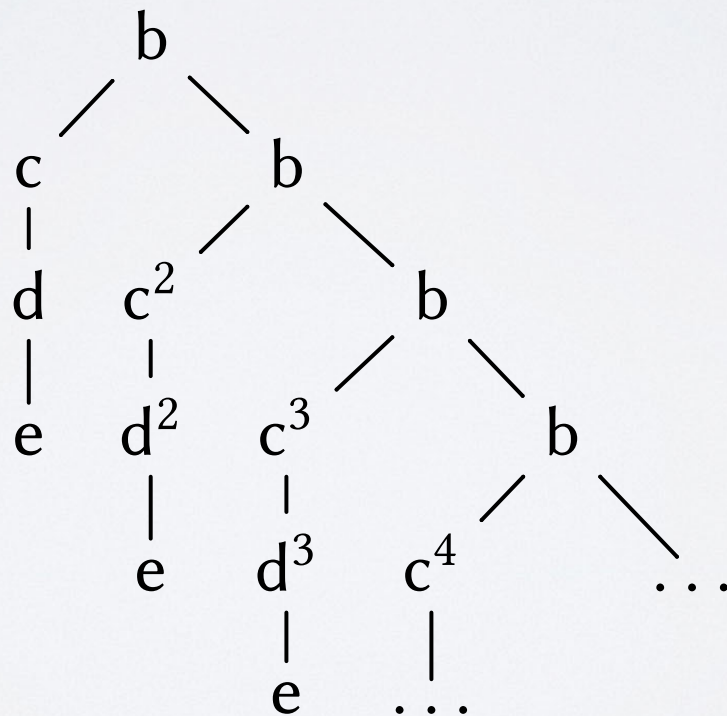
Walukiewicz et al
(ICALP 2005)

Ong
(LICS 2006)

Kobayashi
(FOPPL 2009)

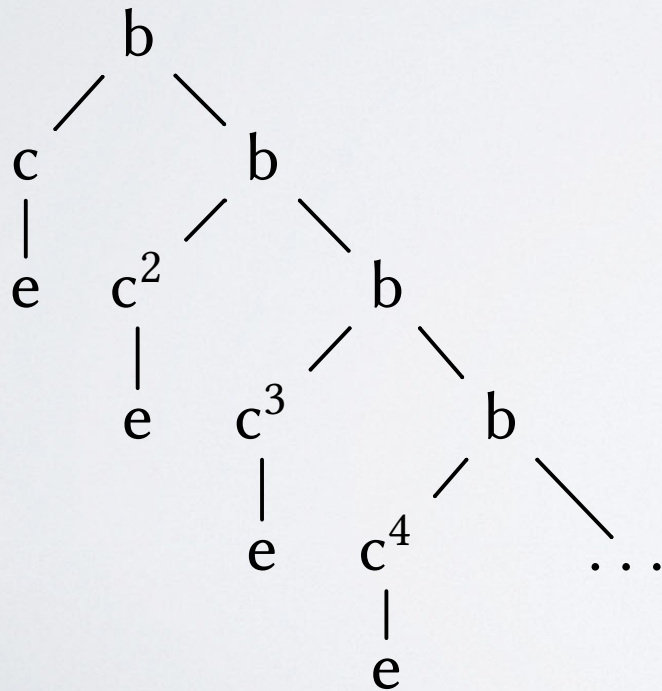
TREES OVER FIRST-ORDER ALPHABETS

$$\Sigma = \{b :: o \rightarrow o \rightarrow o, c :: o \rightarrow o, d :: o \rightarrow o, e :: o\}$$



RECURSIVE APPLICATIVE PROGRAM SCHEMES (NIVAT, 1970S)

$$F_i(x_1, \dots, x_{n_i}) = e_i$$



$$\begin{aligned} S &= F(ce) \\ F(x) &= b(x, F(cx)) \end{aligned}$$

HIGHER-ORDER TYPES

$$A ::= o \mid A \rightarrow A$$

$$\textit{ord}(o) = 0$$

$$\textit{ord}(A_1 \rightarrow A_2) = \max(\textit{ord}(A_1) + 1, \textit{ord}(A_2))$$

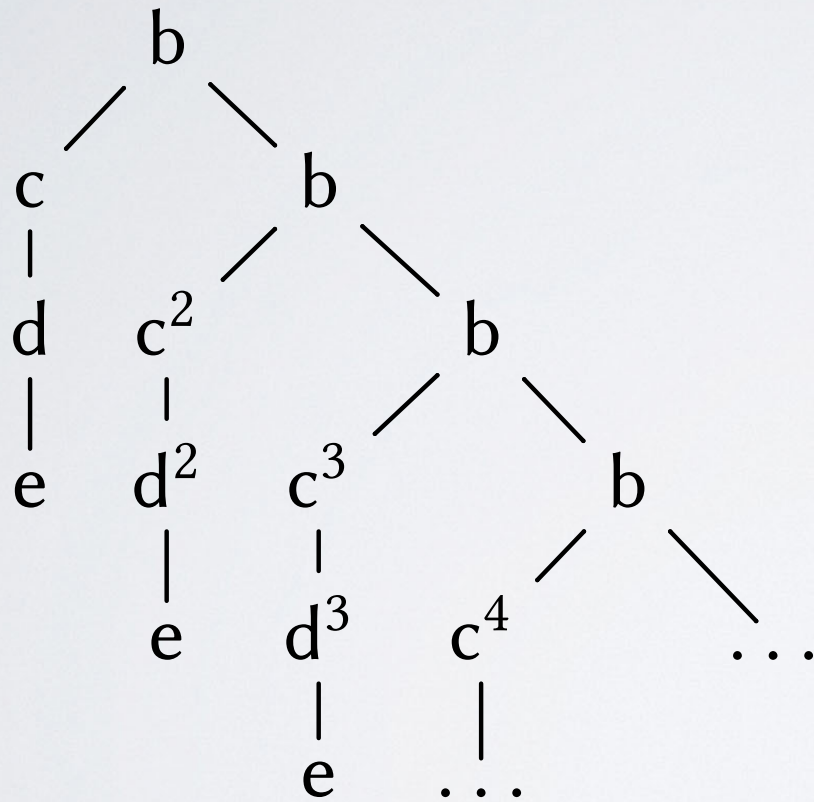
HIGHER-ORDER RECURSION SCHEMES

order-2 HORS $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$

$$\begin{aligned}\Sigma &= \{b :: o \rightarrow o \rightarrow o, c :: o \rightarrow o, d :: o \rightarrow o, e :: o\} \\ \mathcal{N} &= \{S :: o, F :: (o \rightarrow o) \rightarrow o, G :: o \rightarrow o, H :: (o \rightarrow o) \rightarrow o \rightarrow o\}\end{aligned}$$

$$\begin{aligned}S &= FG \\ Ff &= b(f\ e)(F(H\ f)) \\ Gx &= c(dx) \\ Hfx &= c(f(dx))\end{aligned}$$

EXAMPLE



$$\begin{aligned} S &= F G \\ F f &= b (f e) (F (H f)) \\ G x &= c (d x) \\ H f x &= c (f (d x)) \end{aligned}$$

MSO (LICS 2006)

On model-checking trees generated by higher-order recursion schemes

C.-H. L. Ong*
Oxford University Computing Laboratory

Theorem 1. *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.*

MSO (FOSSSACS 2002)

Higher-Order Pushdown Trees Are Easy

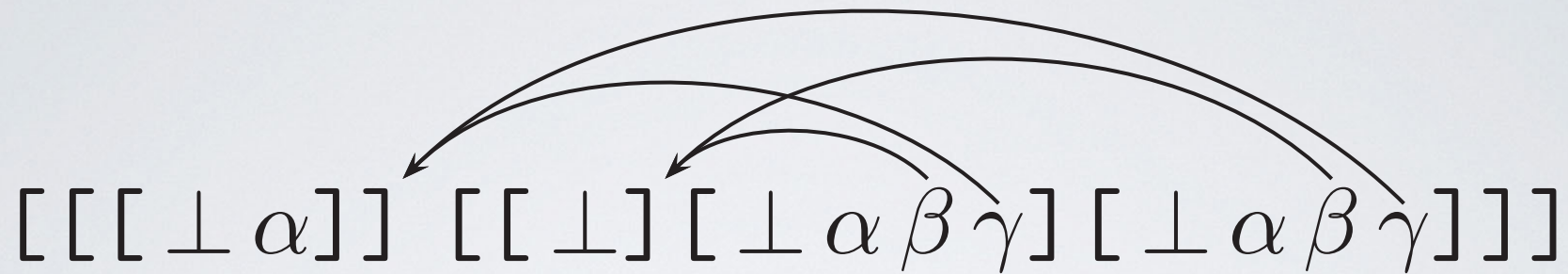
Teodor Knapik¹, Damian Niwiński^{2*}, and Paweł Urzyczyn^{2**}

¹ Université de la Réunion, BP 7151,
97715 Saint Denis Messageries Cedex 9, Réunion
`knapik@univ--reunion.fr`

² Institute of Informatics, Warsaw University
ul. Banacha 2, 02-097 Warszawa, Poland
`{niwinski,urzy}@mimuw.edu.pl`

Abstract. We show that the monadic second-order theory of an infinite tree recognized by a higher-order pushdown automaton of any level is decidable. We also show that trees recognized by pushdown automata of level n coincide with trees generated by safe higher-order grammars of level n . Our decidability result extends the result of Courcelle on algebraic (pushdown of level 1) trees and our own result on trees of level 2.

AUTOMATA



Unsafe Grammars and Panic Automata

Teodor Knapik¹, Damian Niwiński^{2,*},
Paweł Urzyczyn^{2,**}, and Igor Walukiewicz^{3,***}

ICALP'05

Collapsible Pushdown Automata and Recursion Schemes

MATTHEW HAGUE, Royal Holloway, University of London
ANDRZEJ S. MURAWSKI, DIMAP and Department of Computer Science, University of Warwick
C.-H. LUKE ONG, Department of Computer Science, University of Oxford
OLIVIER SERRE, IRIF, CNRS & Université Paris Diderot – Paris 7

LICS'08

HORS (POPL 2009)

Types and Higher-Order Recursion Schemes for Verification of Higher-Order Programs

Naoki Kobayashi

Tohoku University
koba@ecei.tohoku.ac.jp

Abstract

We propose a new verification method for temporal properties of higher-order functional programs, which takes advantage of Ong's recent result on the decidability of the model-checking problem for higher-order recursion schemes (HORS's). A program is transformed to an HORS that generates a tree representing all the possible event sequences of the program, and then the HORS is model-checked. Unlike most of the previous methods for verification of higher-order programs, our verification method is sound and complete. Moreover, this new verification framework allows a smooth integration of abstract model checking techniques into verification of higher-order programs. We also present a type-based verification

lem of resource usage verification [19] for higher-order functional languages with dynamic resource creation and access primitives. The goal of the verification is to check that each dynamically created resource is accessed in a proper manner (like “an opened file is eventually closed, and it is not read or written after being closed”). Assertion-based model-checking problems (like “ $X > 0$ holds at program point p ”) can also be recasted as the resource verification problem, by regarding an assertion failure as an access to a global resource. (For example, “`assert(b)`” can be transformed into “`if b then skip else fail`,” where `fail` is an action to the global resource. Then the problem of checking lack of assertion failures is reduced to the resource usage verification problem of checking whether the `fail` action occurs.)

LINEARITY (POPL'18)

Linearity in Higher-Order Recursion Schemes

PIERRE CLAIRAMBAULT, Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France

CHARLES GRELOIS, INRIA, Sophia Antipolis, France and Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, Marseille, France

ANDRZEJ S. MURAWSKI, Department of Computer Science, University of Oxford, United Kingdom

Higher-order recursion schemes (HORS) have recently emerged as a promising foundation for higher-order program verification. We examine the impact of enriching HORS with linear types. To that end, we introduce two frameworks that blend non-linear and linear types: a variant of the λY -calculus and an extension of HORS, called linear HORS (LHORS).

First we prove that the two formalisms are equivalent and there exist polynomial-time translations between them. Then, in order to support model-checking of (trees generated by) LHORS, we propose a refined version of alternating parity tree automata, called LNAPTA, whose behaviour depends on information about linearity. We show that the complexity of LNAPTA model-checking for LHORS depends on two type-theoretic parameters: linear order and linear depth. The former is in general smaller than the standard notion of order and ignores linear function spaces. In contrast, the latter measures the depth of linear clusters inside a type. Our main result states that LNAPTA model-checking of LHORS of linear order n is n -EXPTIME-complete, when linear depth is fixed. This generalizes and improves upon the classic result of Ong, which relies on the standard

- **Data (booleans)**

$$0 \rightarrow \cdots \rightarrow 0 \rightarrow 0$$

- **CBV**

$$(A_1 \rightarrow_v A_2)^* = A_1^* \rightarrow (A_2^* \rightarrow o) \rightarrow o$$

THREE EXAMPLES

Higher-Order Multi-Parameter Tree Transducers and Recursion Schemes for Program Verification

Naoki Kobayashi
Tokoh University
kobae@cc.tohoku.ac.jp

Naoshi Tabuchi
Tokoh University
tabee@b.ecel.tohoku.ac.jp

Hiroshi Unno
Tokoh University
unno@b.ecel.tohoku.ac.jp

Abstract

We introduce higher-order, multi-parameter, tree transducers (HMTTs, for short), which are kinds of higher-order tree transducers that take input trees and output a (possibly infinite) tree. We study the problem of checking whether the tree generated by a given HMTT conforms to a given output specification, provided that the input trees conform to input specifications (where both input and output specifications are regular tree languages). HMTTs subsume higher-order recursion schemes and ordinary tree transducers, so that their verification has a number of potential applications to verification of functional programs using recursive data structures, including resource usage verification, string analysis, and exact type-checking of XML-processing programs.

We propose a sound but incomplete verification algorithm for the HMTT verification problem: the algorithm reduces the verification problem to a model-checking problem for higher-order recursion schemes extended with finite data domains, and then uses (an extension of) Kobayashi's algorithm for model-checking recursion schemes. While the algorithm is incomplete (indeed, as we show in the paper, the verification problem is undecidable in general), it is sound and complete for a subclass of HMTTs called *linear HMTTs*. We have applied our HMTT verification algorithm to various program verification problems and obtained promising results.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; F.3.1 [Logics and Meaning of Programs]: Specifying and Verifying and Reasoning about Programs

General Terms: Languages, Verification

1. Introduction

Kobayashi [20] has recently proposed a verification method for higher-order functional programs based on Ong's decidability result on model-checking recursion schemes [32]. A higher-order recursion scheme (recursion scheme, for short) is a grammar for generating a (possibly infinite) tree. It is an extension of regular tree grammars, where non-terminal symbols can take trees and higher-order functions on trees as parameters. For example, the following grammar G_0 is an order-1 recursion scheme, where the non-

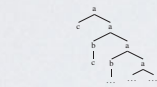


Figure 1. The tree generated by G_0 .

terminal F takes a tree x as an argument.

$S \rightarrow Fc \quad Fx \rightarrow a x (F(b x))$

Here, each non-terminal has exactly one rewrite rule. By infinitary rewriting of the start symbol S :

$S \rightarrow Fc \rightarrow a c (F(b c)) \rightarrow \dots$

we obtain the infinite tree shown in Figure 1. Ong [32] has shown that model mu-calculus model-checking of recursion schemes (Given a recursion scheme G and a model mu-calculus formula φ , does the tree generated by G satisfy φ ?) is π -EXPTIME-complete (where π is the order of the recursion scheme G). The idea of Kobayashi's verification method [20] is to translate a functional program into a recursion scheme that generates a tree whose paths represent all the possible event sequences of the program, so that temporal properties of the functional program can be verified by model-checking the recursion scheme. For example, consider the following program that accesses a file (where x denotes a random boolean value):

```
let x = open_in "foo" in
let rec f() = if x then close(x) else read(x); f()
in f()
```

It can be translated into the following recursion scheme G_0 :

$S \rightarrow F a \quad F k \rightarrow b r(c k) (r(F k))$

Here, r , c , $b r$ and a denote a read operation, a close operation, a non-deterministic branch, and program termination respectively. The recursion scheme generates the tree shown in Figure 2, which represents all the possible event (i.e., read, write, branch, and termination) sequences of the program. Kobayashi [20] applied the verification method to resource usage verification [14] (the problem of checking whether a program accesses resources such as files in a valid manner), and showed that it is sound and complete for the simply-typed λ -calculus extended with recursion, resource creation/access primitives, and booleans. The completeness follows intuitively because recursion schemes are essentially terms of the simply-typed λ -calculus with recursion and

A Traversal-based Algorithm for Higher-Order Model Checking

Robin P. Neatherway
University of Oxford
robin.neatherway@cs.ox.ac.uk

C.-H. Luke Ong
University of Oxford
luke.ong@cs.ox.ac.uk

Steven J. Ramsay
University of Oxford
steven.ramsay@cs.ox.ac.uk

Abstract

Higher-order model checking—the model checking of trees generated by higher-order recursion schemes (HORS)—is a natural generalisation of finite-state and pushdown model checking. Recent work has shown that it can serve as a basis for hardware model checking for functional languages such as ML and Haskell. In this paper, we introduce higher-order recursion schemes with cases (HORSC), which extend HORS with a definition-by-cases construct (to express program branching based on data) and non-determinism (to express abstractions of behaviours). This paper is a study of the *universal HORSC model checking problem for deterministic trivial automata*: does the automaton accept every tree in the tree language generated by the given HORSC? We first characterise the model checking problem by an intersection type system extended with a carefully restricted form of union types. We then present an algorithm for deciding the model checking problem, which is based on the notion of *traverse* induced by the fully abstract game semantics of these schemes, but presented as a goal-directed construction of derivations in the intersection and union type system. We view HORSC model checking as a suitable backend engine for an approach to verifying functional programs. We have implemented the algorithm in a tool called *Tx* to MC, and demonstrated its effectiveness on a test suite of programs, including abstract models of functional programs obtained via an abstraction-refinement procedure from pattern-matching recursion schemes.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; F.3.1 [Logics and Meaning of Programs]: Specifying and Verifying and Reasoning about Programs

General Terms: Algorithms, Verification

Keywords: Model-checking, Higher-order Programs

1. Introduction

Over the past decade, model checking and its allied methods have been applied to program verifications with great effect. For

first-order, imperative programs, highly optimised finite-state and pushdown model checkers (such as SLAM [2] and BLAST [13]) have been successfully applied to bug-finding, property checking and test case generation. Building on theoretical results on the model checking of higher-order recursion schemes (HORS) [6, 16], Kobayashi [8] has sparked a growing interest in the development of an analogous model checking framework for higher-order, functional programs.

A HORS is a kind of higher-order grammar, which can be viewed as a mechanism for generating a possibly-infinite, ranked tree. HORS model checking is concerned with the problem of deciding whether the tree generated by a given HORS satisfies a given property and, when the property is expressed by a formula of the modal mu-calculus (equivalently, an alternating parity tree automaton), then the problem is known to be decidable [16]. Since they can equally well be viewed as a closed, ground-type term of the simply-typed lambda calculus with recursion and misinterpreted first-order constants, HORS are a natural home for models of higher-order computation. Indeed, HORS model checking is a smooth generalisation of finite-state and pushdown model checking (finite-state programs and pushdown systems/boolean programs are captured by order-0 and order-1 HORS respectively).

HORS model checking is, inherently, an extremely complex problem. Ong [16] has shown that the model mu-calculus model checking problem for order- n recursion schemes is π -EXPTIME (i.e., tower of exponentials of height n) complete. Even for the purposes of safety verification (model checking against properties expressible as *deterministic trivial tree automata* (DTT)), the problem is $(n-1)$ -EXPTIME complete [11], which is still formidably complex. Hence, the feasibility of HORS model checking as a verification technology is predicated upon the ability to design decision procedures that lift the worst-case complexity only in pathological cases.

That such algorithms are possible was demonstrated by Kobayashi's *hybrid algorithm*, presented in [7], which solves the safety verification problem. In an attempt to avoid the hyper-exponential bottleneck, the algorithm closely analyses the actual behaviour of the HORS as it is evaluated, generating the ranked tree. The hybrid algorithm builds a graph to record the trace of this computational behaviour and from the graph derives guesses at proofs which witness the satisfaction of the property. The algorithm is implemented in the *Tx* to CS tool [9], which has been shown to perform remarkably well in a variety of applications.

However, whilst HORS allow for the expression of higher-order behaviour very naturally, they lack two important features which, we believe, are highly desirable in a convenient abstract model of functional programs. The first is a case analysis construct, with which one can express program branching based on data; the second

Complexity of Model-Checking Call-by-Value Programs

Takeshi Tsukada^{1,2} and Naoki Kobayashi³

¹ University of Oxford
² JSPS Postdoctoral Fellow for Research Abroad
³ The University of Tokyo

Abstract. This paper studies the complexity of the reachability problem (a typical and practically important instance of the model-checking problem) for simply-typed call-by-value programs with recursion, Boolean values, and non-deterministic branch, and proves the following results. (1) The reachability problem for order-3 programs is nondeterministic. Thus, unlike in the call-by-name case, the order of the input program does not serve as a good measure of the complexity. (2) Instead, the depth of types is an appropriate measure: the reachability problem for depth- n programs is π -EXPTIME complete. In particular, the previous upper bound given by the CPS translation is not tight. The algorithm used to prove the upper bound result is based on a novel intersection type system, which we believe is of independent interest.

1 Introduction

A promising approach to verifying higher-order functional programs is to use higher-order model checking [7, 8, 15], which is a decision problem about the trees generated by higher-order recursion schemes. Various verification problems such as the reachability problem and the resource usage verification [5] are reducible to the higher-order model checking [8].

This paper addresses a variant of the higher-order model checking, namely, the reachability problem for simply-typed *call-by-value* Boolean programs. It is the problem to decide, given a program with Boolean primitives and a special constant meaning the failure, whether the evaluation of the program fails. This is a practically important problem that can be a basis for verification of programs written in call-by-value languages such as ML and OCaml. In fact, MoChi [11], a software model-checker for a subset of OCaml, reduces a verification problem to a reachability problem for a call-by-value Boolean program.

In the previous approach [11], the reachability problem for call-by-value programs was reduced to that for call-by-name programs via the CPS transformation. From a complexity-theoretic point of view, however, this reduction via the CPS transformation has a bad effect: the order of a function is raised by 2 for each increase of the arity of the function. Since the reachability of order- n call-by-name programs is $(n-1)$ -EXPTIME complete in general, the approach may suffer from double exponential blow-up of the time complexity for each increase

A. Munchall (Ed.): FOSSACS 2014, LNCS 8412, pp. 180–194, 2014.
© Springer-Verlag Berlin Heidelberg 2014

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DOI: 10.1007/978-3-642-54058-8_10
Copyright © 2014 ACM 978-3-642-54058-8/14/0001...\$10.00

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DOI: 10.1007/978-3-642-54058-8_10
Copyright © 2014 ACM 978-3-642-54058-8/14/0001...\$10.00

RSFL

(POPL 2010)

IORSC

(POPL 2012)

CBV

(FOSSACS 2014)

LINEAR HORS

$$\varphi, \psi, \dots ::= o \mid \bar{\omega} \multimap \psi \mid \varphi \rightarrow \psi$$

$$\bar{\omega}, \kappa, l, \dots ::= \varphi \mid \&_{i \in I} \varphi_i$$

TYPING RULES

$$\frac{\Gamma \mid \Delta_1 \vdash_{\text{ap}} t :: \varpi \multimap \varphi \quad \Gamma \mid \Delta_2 \vdash_{\text{ap}} u :: \varpi}{\Gamma \mid \Delta_1, \Delta_2 \vdash_{\text{ap}} t u :: \varphi}$$

$$\frac{\Gamma \mid \Delta \vdash_{\text{ap}} t :: \varphi_1 \rightarrow \varphi_2 \quad \Gamma \mid _ \vdash_{\text{ap}} u :: \varphi_1}{\Gamma \mid \Delta \vdash_{\text{ap}} t u :: \varphi_2}$$

$$\frac{\Gamma \mid \Delta \vdash_{\text{ap}} t_i :: \varphi_i \quad (i \in I)}{\Gamma \mid \Delta \vdash_{\text{ap}} \langle t_i \mid i \in I \rangle :: \&_{i \in I} \varphi_i}$$

$\lambda\mathcal{E}Y$

$$\overline{\Gamma, x :: \kappa \mid \Delta \vdash x :: \kappa}$$

$$\overline{j \in I \quad \Gamma \mid \Delta, x :: \&_{i \in I} \varphi_i \vdash \pi_j x :: \varphi_j}$$

$$\frac{\Gamma \mid \Delta_1 \vdash t :: \kappa \multimap \varphi \quad \Gamma \mid \Delta_2 \vdash u :: \kappa}{\Gamma \mid \Delta_1, \Delta_2 \vdash t u :: \varphi}$$

$$\frac{\Gamma \mid \Delta \vdash t :: \varphi \rightarrow \psi \quad \Gamma \mid _ \vdash u :: \varphi}{\Gamma \mid \Delta \vdash t u :: \psi}$$

$$\frac{\Gamma, x :: \varphi \mid \Delta \vdash t :: \psi}{\Gamma \mid \Delta \vdash \lambda x^\varphi. t :: \varphi \rightarrow \psi}$$

$$\frac{\Gamma \mid \Delta, x :: \kappa \vdash t :: \varphi}{\Gamma \mid \Delta \vdash \ell x^\kappa. t :: \kappa \multimap \varphi}$$

$$\frac{\Gamma \mid \Delta \vdash t_i :: \varphi_i \quad (1 \leq i \leq n)}{\Gamma \mid \Delta \vdash \langle t_i \mid 1 \leq i \leq n \rangle :: \&_{1 \leq i \leq n} \varphi_i}$$

$$\frac{\Gamma \mid \Delta \vdash t :: \&_{1 \leq i \leq n} \varphi_i}{\Gamma \mid \Delta \vdash \pi_i t :: \varphi_i}$$

$$\frac{\Gamma, x : \kappa \mid _ \vdash t :: \kappa}{\Gamma \mid _ \vdash \forall x^\kappa. t :: \kappa}$$

LINEAR ORDER

$$\varphi, \psi, \dots ::= o \mid \varpi \multimap \psi \mid \varphi \rightarrow \psi$$

$$\varpi, \kappa, l, \dots ::= \varphi \mid \&_{i \in I} \varphi_i$$

$$\text{lo}(o) = 0$$

$$\text{lo}(\varpi \multimap \varphi) = \max(\text{lo}(\varpi), \text{lo}(\varphi))$$

$$\text{lo}(\varphi \rightarrow \psi) = \max(\text{lo}(\varphi) + 1, \text{lo}(\psi))$$

$$\text{lo}(\&_{i \in I} \varphi_i) = \max_{i \in I} \text{lo}(\varphi_i)$$

LINEAR ORDER-0 LHORS

$$\mathcal{N} = \{ \begin{array}{l} S :: o, \\ F :: (o \multimap o) \multimap o, \\ G :: o \multimap o, \\ H :: (o \multimap o) \multimap o \multimap o \end{array} \}$$

$$\begin{array}{lcl} S & = & F G \\ Ff & = & b \langle f \text{ e}, F (H f) \rangle \\ Gx & = & c (d x) \\ Hfx & = & c (f (d x)) \end{array}$$

MAIN RESULT (POPL'18)

THEOREM 2. *Assume $n \geq 1$. The time complexity of checking whether a LNAPTA $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$ accepts the value tree of a LHORS \mathcal{G} of linear order n (and bounded linear depth) is $\exp_n(O(\text{poly}(|Q||\mathcal{G}|)))$.*

APPLICATIONS

- RSFD

$$o \multimap \dots \multimap o \multimap o$$

- HORSC

$$o \& \dots \& o \multimap o$$

- CBV

$$(A_1 \rightarrow_v A_2)^* = A_1^* \rightarrow (A_2^* \rightarrow o) \multimap o$$

THREE PAPERS

Higher-Order Multi-Parameter Tree Transducers and Recursion Schemes for Program Verification

Naoki Kobayashi
Tokohu University
kobae@ecei.tohoku.ac.jp

Naoshi Tabuchi
Tokohu University
tabee@b.ecei.tohoku.ac.jp

Hiroshi Unno
Tokohu University
unno@b.ecei.tohoku.ac.jp

Abstract

We introduce higher-order, multi-parameter, tree transducers (HMTTs, for short), which are kinds of higher-order tree transducers that take input trees and output a (possibly infinite) tree. We study the problem of checking whether the tree generated by a given HMTT conforms to a given output specification, provided that the input trees conform to input specifications (where both input specifications are regular tree languages). HMTTs subsume higher-order recursion schemes and ordinary tree transducers, so that their verification has a number of potential applications to verification of functional programs using recursive data structures, including resource usage verification, string analysis, and exact type-checking of XML-processing programs.

We propose a sound but incomplete verification algorithm for the HMTT verification problem: the algorithm reduces the verification problem to a model-checking problem for higher-order recursion schemes extended with finite data domains, and then uses (an extension of) Kobayashi's algorithm for model-checking recursion schemes. While the algorithm is incomplete (indeed, as we show in the paper, the verification problem is undecidable in general), it is sound and complete for a subclass of HMTTs called *linear HMTTs*. We have applied our HMTT verification algorithm to various program verification problems and obtained promising results.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; F.3.1 [Logics and Meaning of Programs]: Specifying and Verifying and Reasoning about Programs

General Terms: Languages, Verification

1. Introduction

Kobayashi [20] has recently proposed a verification method for higher-order functional programs based on Ong's decidability result on model-checking recursion schemes [32]. A higher-order recursion scheme (recursion scheme, for short) is a grammar for generating a (possibly infinite) tree. It is an extension of regular tree grammars, where non-terminal symbols can take trees and higher-order functions on trees as parameters. For example, the following grammar G_0 is an order-1 recursion scheme, where the non-

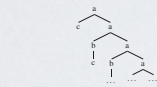


Figure 1. The tree generated by G_0 .

terminal F takes a tree x as an argument.

$S \rightarrow Fc \quad Fx \rightarrow a x (F(bx))$

Here, each non-terminal has exactly one rewrite rule. By infinitary rewriting of the start symbol S :

$S \rightarrow Fc \rightarrow a c (F(b c)) \rightarrow \dots$

we obtain the infinite tree shown in Figure 1. Ong [32] has shown that model mu-calculus model-checking of recursion schemes (Given a recursion scheme G and a model mu-calculus formula φ , does the tree generated by G satisfy φ ?) is π -EXPTIME-complete (where π is the order of the recursion scheme G). The idea of Kobayashi's verification method [20] is to translate a functional program into a recursion scheme that generates a tree whose paths represent all the possible event sequences of the program, so that temporal properties of the functional program can be verified by model-checking the recursion scheme. For example, consider the following program that accesses a file (where x denotes a random boolean value):

```
let x = open_in "foo" in
let rec f() = if x then close(x) else read(x); f()
in f()
```

It can be translated into the following recursion scheme G_0 :

$S \rightarrow F a \quad F k \rightarrow b r(c k) (r(F k))$

Here, r , c , b , and a denote a read operation, a close operation, a non-deterministic branch, and program termination respectively. The recursion scheme generates the tree shown in Figure 2, which represents all the possible event (i.e., read, write, branch, and termination) sequences of the program. Kobayashi [20] applied the verification method to resource usage verification [14] (the problem of checking whether a program accesses resources such as files in a valid manner), and showed that it is sound and complete for the simply-typed λ -calculus extended with recursion, resource creation/access primitives, and booleans. The completeness follows intuitively because recursion schemes are essentially terms of the simply-typed λ -calculus with recursion and

A Traversal-based Algorithm for Higher-Order Model Checking

Robin P. Naeherway
University of Oxford
robin.naeherway@cs.ox.ac.uk

C.-H. Luke Ong
University of Oxford
luke.ong@cs.ox.ac.uk

Steven J. Ramsay
University of Oxford
steven.ramsay@cs.ox.ac.uk

Abstract

Higher-order model checking—the model checking of trees generated by higher-order recursion schemes (HORS)—is a natural generalisation of finite-state and pushdown model checking. Recent work has shown that it can serve as a basis for Haskell model checking for functional languages such as ML and Haskell. In this paper, we introduce *higher-order recursion schemes with cuts* (HORSC), which extend HORS with a definition-by-cases construct (to express program branching based on data) and non-determinism (to express abstractions of behaviours). This paper is a study of the *universal HORSC model checking problem for deterministic trivial automata*: does the automaton accept every tree in the tree language generated by the given HORSC? We first characterise the model checking problem by an intersection type system extended with a carefully restricted form of union types. We then present an algorithm for deciding the model checking problem, which is based on the notion of *traverse* induced by the fully abstract game semantics of these schemes, but presented as a goal-directed construction of derivations in the intersection and union type system. We view HORSC model checking as a suitable backend engine for an approach to verifying functional programs. We have implemented the algorithm in a tool called *TruMOC*, and demonstrated its effectiveness on a test suite of programs, including abstract models of functional programs obtained via an abstraction-refinement procedure from pattern-matching recursion schemes.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; F.3.1 [Logics and Meaning of Programs]: Specifying and Verifying and Reasoning about Programs

General Terms: Algorithms, Verification

Keywords: Model-checking, Higher-order Programs

1. Introduction

Over the past decade, model checking and its allied methods have been applied to program verification with great effect. For

first-order, imperative programs, highly optimised finite-state and pushdown model checkers (such as SLAM [2] and BLAST [1]) have been successfully applied to bug-finding, property checking and test case generation. Building on theoretical results on the model checking of *higher-order recursion schemes* (HORS) [6, 16], Kobayashi [8] has sparked a growing interest in the development of an analogous model checking framework for higher-order, functional programs.

A HORS is a kind of higher-order grammar, which can be viewed as a mechanism for generating a possibly-infinite, ranked tree. HORS model checking is concerned with the problem of deciding whether the tree generated by a given HORS satisfies a given property and, when the property is expressed by a formula of the modal mu-calculus (equivalently, an alternating parity tree automaton), then the problem is known to be decidable [16]. Since they can equally well be viewed as a closed, ground-type term of the simply-typed lambda calculus with recursion and misinterpreted first-order constants, HORS are a natural home for models of higher-order computation. Indeed, HORS model checking is a smooth generalisation of finite-state and pushdown model checking (finite-state programs and pushdown systems/boolean programs are captured by order-0 and order-1 HORS respectively).

HORS model checking is, inherently, an extremely complex problem. Ong [16] has shown that the model mu-calculus model checking problem for order- n recursion schemes is π -EXPTIME (i.e. tower of exponentials of height n) complete. Even for the purposes of safety verification (model checking against properties expressible as *deterministic trivial tree automata* (DTT)), the problem is $(n-1)$ -EXPTIME complete [11], which is still formidably complex. Hence, the feasibility of HORS model checking as a verification technology is predicated upon the ability to design decision procedures that lift the worst-case complexity only in pathological cases.

That such algorithms are possible was demonstrated by Kobayashi's *hybrid algorithm*, presented in [7], which solves the safety verification problem. In an attempt to avoid the hyper-exponential bottleneck, the algorithm closely analyses the actual behaviour of the HORS as it is evaluated, generating the ranked tree. The hybrid algorithm builds a graph to record the trace of this computational behaviour and from the graph derives guesses at proofs which witness the satisfaction of the property. The algorithm is implemented in the *TruMOC* tool [9], which has been shown to perform remarkably well in a variety of applications.

However, whilst HORS allow for the expression of higher-order behaviour very naturally, they lack two important features which, we believe, are highly desirable in a convenient abstract model of functional programs. The first is a case analysis construct, with which one can express program branching based on data; the sec-

Complexity of Model-Checking Call-by-Value Programs

Takeshi Tsukada^{1,2} and Naoki Kobayashi³

¹ University of Oxford

² JSPS Postdoctoral Fellow for Research Abroad

³ The University of Tokyo

Abstract. This paper studies the complexity of the reachability problem (a typical and practically important instance of the model-checking problem) for simply-typed call-by-value programs with recursion, Boolean values, and non-deterministic branch, and proves the following results. (1) The reachability problem for order-3 programs is nondeterministic. Thus, unlike in the call-by-name case, the order of the input program does not serve as a good measure of the complexity. (2) Instead, the depth of types is an appropriate measure: the reachability problem for depth- n programs is π -EXPTIME complete. In particular, the previous upper bound given by the CPS translation is not tight. The algorithm used to prove the upper bound result is based on a novel intersection type system, which we believe is of independent interest.

1 Introduction

A promising approach to verifying higher-order functional programs is to use higher-order model checking [7, 8, 15], which is a decision problem about the trees generated by higher-order recursion schemes. Various verification problems such as the reachability problem and the resource usage verification [5] are reducible to the higher-order model checking [8].

This paper addresses a variant of the higher-order model checking, namely, the reachability problem for simply-typed *call-by-value* Boolean programs. It is the problem to decide, given a program with Boolean primitives and a special constant meaning the failure, whether the evaluation of the program fails. This is a practically important problem that can be a basis for verification of programs written in call-by-value languages such as ML and OCaml. In fact, MoChI [11], a software model-checker for a subset of OCaml, reduces a verification problem to a reachability problem for a call-by-value Boolean program.

In the previous approach [11], the reachability problem for call-by-value programs was reduced to that for call-by-name programs via the CPS transformation. From a complexity-theoretic point of view, however, this reduction via the CPS transformation has a bad effect: the order of a function is raised by 2 for each increase of the arity of the function. Since the reachability of order- n call-by-name programs is $(n-1)$ -EXPTIME complete in general, the approach may suffer from double exponential blow-up of the time complexity for each increase

A. Munchall (Ed.): FORSACS 2014, LNCS 8412, pp. 180–194, 2014.
© Springer-Verlag Berlin Heidelberg 2014

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DOI: 10.1007/978-3-662-43010-1_10
Copyright © 2014 ACM 978-3-662-43010-1...\$10.00

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DOI: 10.1007/978-3-662-43010-1_10
Copyright © 2014 ACM 978-3-662-43010-1...\$10.00

RSFL

(FOPL 2010)

IORSC

(FOPL 2012)

CBV

(FOSSACS 2014)

LINEAR DEPTH

Definition 2. The **local linear depth** $\text{lld}(\kappa)$ of a kind κ is defined inductively as follows

$$\begin{aligned}\text{lld}(o) &= 0 & \text{lld}(\varpi \multimap \varphi) &= \max(\text{lld}(\varpi), \text{lld}(\varphi)) + 1 \\ \text{lld}(\&_{i \in I} \varphi_i) &= \max_{i \in I} \text{lld}(\varphi_i) + 1 & \text{lld}(\varphi \rightarrow \psi) &= \text{lld}(\psi)\end{aligned}$$

The **linear depth** of κ , written $\ell d(\kappa)$, is the maximum of $\text{lld}(\iota)$, taken over all subkinds ι of κ .

PROPOSITION 33. *Given a LHORS \mathcal{G} with N non-terminals, kinds of maximal size S , linear depth D and linear order n ; and a linear-nonlinear APT \mathcal{A} with colours bounded by p and states bounded by $Q \geq p$. For $n \geq 1$, the time complexity for solving $\text{Typ}(\mathcal{G}, \mathcal{A})$ is $O(N^{\lceil p/2 \rceil + 2} \exp_n(O(2^D)(QS)^{O(2^D)}))$.*

LICS'09

A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes

N. Kobayashi
Tokohu University

C.-H. L. Ong
University of Oxford

Abstract

The model checking of higher-order recursion schemes has important applications in the verification of higher-order programs. Ong has previously shown that the modal mu-calculus model checking of trees generated by order- n recursion scheme is n -EXPTIME complete, but his algorithm and its correctness proof were rather complex. We give an alternative, type-based verification method: Given a modal mu-calculus formula, we can construct a type system in which a recursion scheme is typable if, and only if, the (possibly infinite, ranked) tree generated by the scheme satisfies the formula. The model checking problem is thus reduced to a type checking problem. Our type-based approach yields a simple verification algorithm, and its correctness proof (constructed without recourse to game semantics) is comparatively easy to understand. Furthermore, the algorithm is polynomial-time in the size of the recursion scheme, assuming that the sizes of types and the formula are bounded above by a constant.

1 Introduction

The model checking of infinite structures generated by higher-order recursion schemes has drawn growing attention from both theoretical and practical communities. From

checking problem for trees generated by arbitrary order- n recursion schemes is n -EXPTIME complete (and hence these trees have decidable MSO theories); further [5] these schemes are equi-expressive with a new class of automata, called *collapsible* pushdown automata. On the practical side, Kobayashi [11] has recently shown that the verification of higher-order programs can be reduced to that of higher-order recursion schemes. He constructed a transformation of a higher-order program into a recursion scheme that generates a (possibly infinite) tree representing all the possible event sequences of the program; thus, temporal properties of the program can be verified by model-checking the recursion scheme.

Ong's algorithm for verifying higher-order recursion schemes is rather complex and probably hard to understand: The algorithm reduces the model-checking problem to a parity game over *variable profiles*, and its correctness proof relies on game semantics [7]. Hague et al. [5] gave an alternative proof via a reduction of the model checking of recursion schemes to that of collapsible pushdown automata; their reduction is also based on game semantics. Kobayashi [11] showed that given a Büchi tree automaton with a trivial acceptance condition (a class which Aehlig [1] has called *trivial automata*), one can construct an intersection type system in which a recursion scheme is typable if, and only if, the tree generated by the scheme is accepted by the automaton. (Prior to Kobayashi's work [11], Aehlig [1] had shown a classification of the decidability of the model

REFINEMENT

- Intersection types

$$\frac{q \in Q}{q :: o}$$

$$\frac{\sigma :: \varpi \quad \tau :: \varphi}{\sigma \multimap \tau :: \varpi \multimap \varphi}$$

$$\frac{(\sigma_i :: \varphi)_{i \in I} \quad \sigma :: \psi}{(\bigwedge_{i \in I} \Box_{c_i} \sigma_i) \rightarrow \sigma :: \varphi \rightarrow \psi}$$

- Parity game

$$Typ(\mathcal{G}, \mathcal{A})$$

$$\exp_n(O(poly(|Q||\mathcal{G}|)))$$

- Number of refinements

$$\#(\varphi \rightarrow \psi) \leq 2^{C\#\varphi}\#\psi$$

$$\#(\varpi \multimap \varphi) = (\#\varpi + 1)(\#\varphi)$$

LNAPTA

Definition 4. **Tree kinds** are the kinds θ generated by

$$\theta ::= o \mid v \multimap \theta \mid o \rightarrow \theta \quad \text{and} \quad v ::= \&_{1 \leq i \leq n} o$$

A **tree signature** is a finite list $\Sigma = b_1 :: \theta_1, \dots, b_n :: \theta_n$ where θ_i is a tree kind for all $1 \leq i \leq n$.

APTA = **LNAPTA** for standard tree kinds

CONCLUSIONS

- LHORS: more expressivity for the same asymptotic model-checking complexity
- Unification and extension of existing results, where reduction to HORS would give inaccurate bounds
- A tool to understand and tame complexity of higher-order model checking