Higher-Order Model Checking

A Tutorial

Naoki Kobayashi   Luke Ong

18 January 2016, POPL 2016 Tutorial

Model Checking

The development of techniques (notably model checking) for the computer-aided verification of computing systems has been a truly successful application of logic to computer science.

2007 ACM Turing Award (Clarke, Emerson and Sifakis) "for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

Model Checking

The development of techniques (notably model checking) for the computer-aided verification of computing systems has been a truly successful application of logic to computer science.

2007 ACM Turing Award (Clarke, Emerson and Sifakis) "for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

What is Model Checking?

Problem: Given a system Sys (e.g. an OS) and a correctness property $Spec$ (e.g. deadlock freedom), does Sys satisfy $Spec$?

The model checking approach:

1. Find an abstract model M of the system Sys.
2. Describe the property $Spec$ as a formula φ of a (decidable) logic.
3. Exhaustively check if φ is violated by M.

Verification of Higher-Order Programs

In the past two decades, there have been significant advances in the theory and engineering of scalable software model checkers (especially for first-order imperative programs such as C). E.g. SLAM, BLAST, CMBC.

- These techniques are much less useful for higher-order programs.
- Yet higher-order features (e.g. lambdas, streams) are already standard in today's leading languages: Java8, C++11, C#5.0, Python, Scala, etc.

Verification of Higher-Order Programs

In the past two decades, there have been significant advances in the theory and engineering of scalable software model checkers (especially for first-order imperative programs such as C). E.g. SLAM, BLAST, CMBC.

- These techniques are much less useful for higher-order programs.
- Yet higher-order features (e.g. lambdas, streams) are already standard in today's leading languages: Java8, C++11, C#5.0, Python, Scala, etc.

Verifying higher-order functional programs: 2 standard approaches

1. Type-based program analysis. E.g. type-and-effect, qualifier, linear
   - sound, scalable but often imprecise
2. Theorem proving and dependent types. E.g. Coq, Agda
   - accurate, typically requires human intervention; does not scale well

We present an approach to verifying higher-order programs via higher-order model checking.

Tutorial Outline

Higher-Order Model Checking is the model checking of infinite structures, such as trees, that are defined by recursion schemes (equivalently $\lambda\mathbf{Y}$-calculus) and related families of higher-order generators.

This tutorial has four parts:

1. Introduction (Ong)
2. Applications to Program Verification (Kobayashi)
3. Type Systems and Algorithms for Higher-Order Model Checking (Kobayashi)
4. Advanced Topics (Ong)

Simple Types (Church JSL 1940)

Types $\quad A \quad ::= \quad o \quad | \quad (A \to B)$

o is the type of trees.

Order of a type: measures "nestedness" on LHS of $\to$.

$$\begin{array}{rcl} \mathsf{order}(o) & := & 0 \\ \mathsf{order}(A \to B) & := & \max(\mathsf{order}(A) + 1, \mathsf{order}(B)) \end{array}$$

Simple Types (Church JSL 1940)

Types $\quad A \quad ::= \quad o \quad | \quad (A \to B)$

o is the type of trees.

Order of a type: measures “nestedness” on LHS of $\to$.

$$\begin{array}{rcl} \mathsf{order}(o) & := & 0 \\ \mathsf{order}(A \to B) & := & \max(\mathsf{order}(A) + 1, \mathsf{order}(B)) \end{array}$$

Examples

1. $\mathbb{N} \to \mathbb{N}$ and $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ both have order 1;
2. $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ has order 2.

Notation $\quad e : A \quad$ means “expression e has type A”.

(Park 68, de Roever 72, Nivat 72, Nivat-Courcelle 78, Damm 82, . . .)

HORS are grammars for trees (and tree languages).

Order-n recursion schemes over Σ = programs of the order-n fragment of $\lambda^{\rightarrow}$**Y**-calculus (i.e. simply-typed λ-calculus + **Y** + order-1 Σ-symbols).

Higher-Order Recursion Schemes (HORS)

(Park 68, de Roever 72, Nivat 72, Nivat-Courcelle 78, Damm 82, . . .)

HORS are grammars for trees (and tree languages).

Order-n recursion schemes over Σ = programs of the order-n fragment of $\lambda^{\rightarrow}\mathbf{Y}$-calculus (i.e. simply-typed λ-calculus + $\mathbf{Y}$ + order-1 Σ-symbols).

Concretely, a HORS is a finite set of simply-typed (higher-order) functions, defined by mutual recursion over Σ, with a distinguished start function S of ground type.

Example (order 1). $\Sigma = \{\, f : o \to (o \to o),\ g : o \to o,\ a : o \,\}$.

$$\mathcal{G} \ : \left\{ \begin{array}{rcl} S & \to & F\,a \\ F\,x & \to & (f\,x)\,(F\,(g\,x)) \end{array} \right.$$

Example (order 1)

$\Sigma = \{ f : o \to (o \to o),\ g : o \to o, a : o \}.$

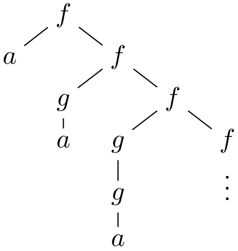
$$\mathcal{G} \ : \ \begin{cases} \quad S & \to \quad F\, a \\ F\, x & \to \quad (f\, x)\, (F\, (g\, x)) \end{cases}$$

Example (order 1)

$$\Sigma = \{\, f : o \to (o \to o),\ g : o \to o, a : o \,\}.$$

$$\mathcal{G} \ : \ \left\{ \begin{array}{rcl} S & \to & F\,a \\ F\,x & \to & (f\,x)\,(F\,(g\,x)) \end{array} \right.$$

$$\begin{array}{rcl} S & \to & F\,a \\ & \to & (f\,a)\,(F\,(g\,a)) \\ & \to & (f\,a)\,(f\,(g\,a)\,(F\,(g\,(g\,a)))) \\ & \to & \cdots \end{array}$$



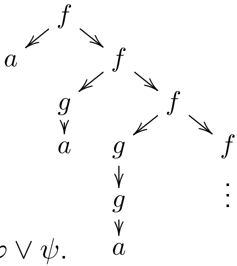
The tree generated, $[\![\, \mathcal{G} \,]\!]$, is the abstract syntax tree underlying $(f\,a)\,(f\,(g\,a)\,(f\,(g\,(g\,a))(\cdots)))$.

Many equivalent ways of defining $[\![\, \mathcal{G} \,]\!]$ (as least fixpoint, least solution, initial algebra, etc.).

A Basic Verification Problem in Higher-Order Computation

E.g. Consider properties of nodes of $[\![G]\!]$:

- $\varphi =$ "Infinitely many f-nodes are reachable".
- $\psi =$ "Only finitely many g-nodes are reachable".



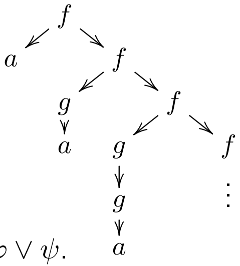
Every node of the tree satisfies $\varphi \vee \psi$.

Monadic second-order logic (MSO) is an expressive logic that can describe correctness properties such as $\varphi \vee \psi$.

A Basic Verification Problem in Higher-Order Computation

E.g. Consider properties of nodes of $[\![G]\!]$:

- φ = "Infinitely many f-nodes are reachable".
- ψ = "Only finitely many g-nodes are reachable".



Every node of the tree satisfies $\varphi \vee \psi$.

Monadic second-order logic (MSO) is an expressive logic that can describe correctness properties such as $\varphi \vee \psi$.

MSO Model-Checking Problem for Trees generated by HORS

- INSTANCE: An order-n recursion scheme $\mathcal{G}$, and an MSO formula φ
- QUESTION: Does the Σ-labelled tree $[\![\mathcal{G}]\!]$ satisfy φ?

QUESTION: Is the above problem decidable?

Theorem (O. LICS06)

For $n \geq 0$, the alternating parity tree automaton (APT) model-checking problem for order-n recursion schemes is n-EXPTIME complete. Hence the MSO model checking problem is decidable.

Decidability and Complexity of MSO Model Checking of HORS

Theorem (O. LICS06)

For $n \geq 0$, the alternating parity tree automaton (APT) model-checking problem for order-n recursion schemes is n-EXPTIME complete. Hence the MSO model checking problem is decidable.

Proofs of Decidability of HOMC / Models of Higher-Order Computation

1. Game semantics (O. LICS06)
2. Collapsible pushdown automata (Hague, Murawski, O. & Serre LICS08)
3. Intersection types (Kobayashi & O. LICS09)
4. Krivine machine (Salvati & Walukiewicz ICALP11)

Outline of Part 4

1. Higher-Order Pushdown Automata: A Model of Higher-order Computation
   - Properties of the Maslov(= Higher-order Pushdown) Hierarchy of Word Languages
   - Computing Downwards Closure of Higher-order Pushdown Languages
2. Model Checking Higher-type Böhm Trees
   - Challenge of Compositional Higher-order Model Checking
   - Automata-Logic-Games Correspondence for Higher-type Computation
3. Some Open Problems

Outline

1. Higher-Order Pushdown Automata: A Model of Higher-order Computation
   - Properties of the Maslov(= Higher-order Pushdown) Hierarchy of Word Languages
   - Computing Downwards Closure of Higher-order Pushdown Languages
2. Model Checking Higher-type Böhm Trees
   - Challenge of Compositional Higher-order Model Checking
   - Automata-Logic-Games Correspondence for Higher-type Computation
3. Some Open Problems

Order-2 pushdown automata

A 1-stack is an ordinary stack. A 2-stack (resp. $(n+1)$-stack) is a stack of 1-stacks (resp. n-stack).

Order-2 pushdown automata

A 1-stack is an ordinary stack. A 2-stack (resp. $(n+1)$-stack) is a stack of 1-stacks (resp. n-stack).

Operations on 2-stacks: s_i ranges over 1-stacks.

$$
\begin{array}{rcll}
push_2 & : & [s_1 \cdots s_{i-1}\, \underbrace{[\gamma_1 \cdots \gamma_n]}_{s_i}] & \mapsto \quad [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1}\, [\gamma_1 \cdots \gamma_n]] & \mapsto \quad [s_1 \cdots s_{i-1}] \\
push_1\, \gamma & : & [s_1 \cdots s_{i-1}\, [\gamma_1 \cdots \gamma_n]] & \mapsto \quad [s_1 \cdots s_{i-1}\, [\gamma_1 \cdots \gamma_n\, \gamma]] \\
pop_1 & : & [s_1 \cdots s_{i-1}\, [\gamma_1 \cdots \gamma_n\, \gamma_{n+1}]] & \mapsto \quad [s_1 \cdots s_{i-1}\, [\gamma_1 \cdots \gamma_n]]
\end{array}
$$

Higher-order pushdown automata (HOPDA) [Maslov 74]

Order-2 pushdown automata

A 1-stack is an ordinary stack. A 2-stack (resp. $(n+1)$-stack) is a stack of 1-stacks (resp. n-stack).

Operations on 2-stacks: s_i ranges over 1-stacks.

$$
\begin{array}{rcll}
push_2 & : & [s_1 \cdots s_{i-1} \underbrace{[\gamma_1 \cdots \gamma_n]}_{s_i}] & \mapsto \quad [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]] & \mapsto \quad [s_1 \cdots s_{i-1}] \\
push_1\,\gamma & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]] & \mapsto \quad [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n\, \gamma]] \\
pop_1 & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n\, \gamma_{n+1}]] & \mapsto \quad [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]]
\end{array}
$$

Idea extends to all finite orders: an order-n PDA has an order-n stack, and has $push_i$ and pop_i for each $1 \leq i \leq n$.

Example: $L := \{ a^n \, b^n \, c^n : n \geq 0 \}$ is recognisable by an order-2 PDA

L is not context free—thanks to the "$uvwxy$ Lemma".

Example: $L := \{ a^n \, b^n \, c^n : n \geq 0 \}$ is recognisable by an order-2 PDA

L is not context free—thanks to the "$uvwxy$ Lemma".

Idea: Use top 1-stack to process $a^n \, b^n$, and height of 2-stack to "remember" n.

Example: $L := \{ a^n\, b^n\, c^n : n \geq 0 \}$ is recognisable by an order-2 PDA

L is not context free—thanks to the "$uvwxy$ Lemma".

Idea: Use top 1-stack to process $a^n\, b^n$, and height of 2-stack to "remember" n.

$$
\begin{tikzcd}
q_1\,[[\,]] \arrow[r, "a"] & q_1\,[[\,]\,[z]] \arrow[r, "a"] & q_1\,[[\,]\,[z]\,[z\ z]] \arrow[d, "b"] \\
 & & q_2\,[[\,]\,[z]\,[z]] \arrow[d, "b"] \\
q_3\,[[\,]] & q_3\,[[\,]\,[z]] \arrow[l, "c"'] & q_2\,[[\,]\,[z]\,[\,]] \arrow[l, "c"']
\end{tikzcd}
$$

Example: $L := \{ a^n \, b^n \, c^n : n \geq 0 \}$ is recognisable by an order-2 PDA

L is not context free—thanks to the "$uvwxy$ Lemma".

Idea: Use top 1-stack to process $a^n \, b^n$, and height of 2-stack to "remember" n.

$$
\begin{tikzcd}
q_1 \; [[\,]] \arrow[r, "a"] & q_1 \; [[\,][z]] \arrow[r, "a"] & q_1 \; [[\,][z][z\; z]] \arrow[d, "b"] \\
 & & q_2 \; [[\,][z][z]] \arrow[d, "b"] \\
q_3 \; [[\,]] & q_3 \; [[\,][z]] \arrow[l, "c"'] & q_2 \; [[\,][z][\,]] \arrow[l, "c"']
\end{tikzcd}
$$

$$
\begin{tikzcd}[column sep=huge]
q_1 \arrow[loop above, "- \xrightarrow{a} push_2 \,;\, push_1 z"] \arrow[r, "z \xrightarrow{b} pop_1 z"'] & q_2 \arrow[loop above, "z \xrightarrow{b} pop_1"] \arrow[r, "\perp \xrightarrow{c} pop_2"'] & q_3 \arrow[loop above, "z \xrightarrow{c} pop_2"]
\end{tikzcd}
$$

'read a' 'read b' 'read c'

Some properties of the Maslov Hierarchy (Maslov 74, 76)

1. HOPDA define an infinite hierarchy of word languages.

Some properties of the Maslov Hierarchy (Maslov 74, 76)

1. HOPDA define an infinite hierarchy of word languages.
2. Orders 0, 1 and 2 languages are regular, context free, and indexed (Aho 68); higher-order languages are not well understood.

Some properties of the Maslov Hierarchy (Maslov 74, 76)

1. HOPDA define an infinite hierarchy of word languages.
2. Orders 0, 1 and 2 languages are regular, context free, and indexed (Aho 68); higher-order languages are not well understood.
3. For each $n \geq 0$, the order-n languages form an abstract family of languages (closed under $+, \cdot, (-)^*$, intersection with regular languages, homomorphism and inverse homo.)

Some properties of the Maslov Hierarchy (Maslov 74, 76)

1. HOPDA define an infinite hierarchy of word languages.
2. Orders 0, 1 and 2 languages are regular, context free, and indexed (Aho 68); higher-order languages are not well understood.
3. For each $n \geq 0$, the order-n languages form an abstract family of languages (closed under $+, \cdot, (-)^*$, intersection with regular languages, homomorphism and inverse homo.)
4. The acceptance problem of alternating order-k PDA is k-EXPTIME complete. (Engelfriet '81)

Some properties of the Maslov Hierarchy (Maslov 74, 76)

1. HOPDA define an infinite hierarchy of word languages.
2. Orders 0, 1 and 2 languages are regular, context free, and indexed (Aho 68); higher-order languages are not well understood.
3. For each $n \geq 0$, the order-n languages form an abstract family of languages (closed under $+, \cdot, (-)^*$, intersection with regular languages, homomorphism and inverse homo.)
4. The acceptance problem of alternating order-k PDA is k-EXPTIME complete. (Engelfriet '81)
5. The emptiness problem of nondeterministic order-k PDA is $(k-1)$-EXPTIME complete. (Engelfriet '81)

Some properties of the Maslov Hierarchy (Maslov 74, 76)

1. HOPDA define an infinite hierarchy of word languages.
2. Orders 0, 1 and 2 languages are regular, context free, and indexed (Aho 68); higher-order languages are not well understood.
3. For each $n \geq 0$, the order-n languages form an abstract family of languages (closed under $+, \cdot, (-)^*$, intersection with regular languages, homomorphism and inverse homo.)
4. The acceptance problem of alternating order-k PDA is k-EXPTIME complete. (Engelfriet '81)
5. The emptiness problem of nondeterministic order-k PDA is $(k-1)$-EXPTIME complete. (Engelfriet '81)

A recent breakthrough

Theorem (Inaba + Maneth FSTTCS08)

All languages of the Maslov Hierarchy are context-sensitive. So Maslov Hierarchy refines Chomsky Hierarchy.

Maslov – a robust hierarchy of word languages

Theorem (Equi-expressivity)

For each $n \geq 0$, the three formalisms

1. *order-n pushdown automata (Maslov 76)*
2. *order-n safe recursion schemes (Damm 82, Damm + Goerdt 86)*
3. *order-n indexed grammars (Maslov 76)*

generate the same class of word languages.

Computing downwards closure of higher-order pushdown languages

Downward closure of $\mathcal{L}$, $\downarrow(\mathcal{L})$, is the set of all subwords of words in $\mathcal{L}$.
E.g. $SubWords(a\,b\,c) = \{\,a\,b\,c,\ b\,c,\ a\,c,\ a\,b,\ a,\ b,\ c,\ \epsilon\,\}$

Theorem (Haines 1969) For all $\mathcal{L} \subseteq \Sigma^*$, $\downarrow(\mathcal{L})$ is regular.

Computing downwards closure of higher-order pushdown languages

Downward closure of $\mathcal{L}$, $\downarrow(\mathcal{L})$, is the set of all subwords of words in $\mathcal{L}$.
E.g. $SubWords(a\,b\,c) = \{\,a\,b\,c,\ b\,c,\ a\,c,\ a\,b,\ a,\ b,\ c,\ \epsilon\,\}$

Theorem (Haines 1969) For all $\mathcal{L} \subseteq \Sigma^*$, $\downarrow(\mathcal{L})$ is regular.

Unfortunately downward closures are not computable, in general.

Algorithms only exist for a few language classes; e.g. context-free, Petri net, indexed (Zetzsche ICALP15).

Computing downwards closure of higher-order pushdown languages

Downward closure of $\mathcal{L}$, $\downarrow(\mathcal{L})$, is the set of all subwords of words in $\mathcal{L}$.
E.g. $SubWords(a\,b\,c) = \{\, a\,b\,c,\ b\,c,\ a\,c,\ a\,b,\ a,\ b,\ c,\ \epsilon \,\}$

Theorem (Haines 1969) For all $\mathcal{L} \subseteq \Sigma^*$, $\downarrow(\mathcal{L})$ is regular.

Unfortunately downward closures are not computable, in general.

Algorithms only exist for a few language classes; e.g. context-free, Petri net, indexed (Zetzsche ICALP15).

Regular representations of downward closures are very useful:
- Regular languages are well behaved under many transformations.
- Many systems permit synchronisation with a regular language.

Computing downwards closure of higher-order pushdown languages

Downward closure of $\mathcal{L}$, $\downarrow(\mathcal{L})$, is the set of all subwords of words in $\mathcal{L}$.
E.g. $SubWords(a\,b\,c) = \{\, a\,b\,c,\ b\,c,\ a\,c,\ a\,b,\ a,\ b,\ c,\ \epsilon \,\}$

Theorem (Haines 1969) For all $\mathcal{L} \subseteq \Sigma^*$, $\downarrow(\mathcal{L})$ is regular.

Unfortunately downward closures are not computable, in general.

Algorithms only exist for a few language classes; e.g. context-free, Petri net, indexed (Zetzsche ICALP15).

Regular representations of downward closures are very useful:
- Regular languages are well behaved under many transformations.
- Many systems permit synchronisation with a regular language.

Example: In message-passing concurrency, complex environments can be abstracted by the downward closure of the messages it sends (or processes it spawns).

Theorem (Hague, Kochems & O. POPL16)

The downward closure of every order-n pushdown language is computable.

(Zetzsche 2015) If $\mathcal{C}$ is a full trio and has decidable $\textsc{Diagonal}(\mathcal{C})$, then it has computable downward closures.

Fix $\Sigma = \{ a_1, \cdots, a_n \}$. $\textsc{Diagonal}(\mathcal{C})$: Given $L \in \mathcal{C}$, does it hold that $\forall k \geq 0 \, . \, \exists w_k \in L \, . \, \big(\#_{a_1}(w_k) \geq k \wedge \cdots \wedge \#_{a_n}(w_k) \geq k\big)$?

Theorem (Hague, Kochems & O. POPL16)

The downward closure of every order-n pushdown language is computable.

(Zetzsche 2015) If $\mathcal{C}$ is a full trio and has decidable $\textsc{Diagonal}(\mathcal{C})$, then it has computable downward closures.

Fix $\Sigma = \{ a_1, \cdots, a_n \}$. $\textsc{Diagonal}(\mathcal{C})$: Given $L \in \mathcal{C}$, does it hold that $\forall k \geq 0 \,.\, \exists w_k \in L \,.\, \big(\#_{a_1}(w_k) \geq k \wedge \cdots \wedge \#_{a_n}(w_k) \geq k\big)$?

Several Consequences

1. Reachability for parameterised concurrent systems of HOPDA communicating asynchronously via a shared global register (La Torre et al. 2015)
2. Finiteness of a language defined by a HOPDA, and
3. Downward closure of the Parikh image of a HOPDA.

Hierarchy of trees generated by HORS

For $n \geq 0$, let $\textbf{RecSchTree}_n$ be the class of Σ-labelled trees generated by order-n recursion schemes.

Some Properties

1. Hierarchy Theorem (Damm 1982) for $\langle\, \textbf{RecSchTree}_n \mid n \in \omega \,\rangle$
2. The hierarchy is highly expressive: order-0 are the regular trees, order-1 are the algebraic trees (Courcelle 1995); order-2 are the hyperalgebraic trees (Knapik et al. 2001).
3. Machine characterization: order-n trees are exactly those generated by order-n collapsible pushdown automata (HMOS LiCS 2008)
4. MSO theories are decidable (to date, the "largest" known such hierarchy of trees).

Outline

1. Higher-Order Pushdown Automata: A Model of Higher-order Computation
   - Properties of the Maslov($=$ Higher-order Pushdown) Hierarchy of Word Languages
   - Computing Downwards Closure of Higher-order Pushdown Languages

2. Model Checking Higher-type Böhm Trees
   - Challenge of Compositional Higher-order Model Checking
   - Automata-Logic-Games Correspondence for Higher-type Computation

3. Some Open Problems

Compositional Higher-Order Model Checking? ... Several Obstacles

1. Like standard model checking, higher-order model checking is mostly a whole program analysis. This can seem surprising: higher order is supposed to aid modular structuring of programs!

Compositional Higher-Order Model Checking? ... Several Obstacles

1. Like standard model checking, higher-order model checking is mostly a whole program analysis. This can seem surprising: higher order is supposed to aid modular structuring of programs!
2. Hitherto HOMC is about computation trees of ground-type functional programs.
   Aim: model check the computation trees of higher-type functional programs (= Böhm trees i.e. trees with binders).

Compositional Higher-Order Model Checking? ... Several Obstacles

1. Like standard model checking, higher-order model checking is mostly a whole program analysis. This can seem surprising: higher order is supposed to aid modular structuring of programs!
2. Hitherto HOMC is about computation trees of ground-type functional programs.
   Aim: model check the computation trees of higher-type functional programs (= Böhm trees i.e. trees with binders).
3. Need a denotational model to support compositional model checking, which should be strategy aware (i.e. modelling Böhm trees, and witnesses of correctness properties of Böhm trees), and organisable into a cartesian closed category of parity games.

Compositional Higher-Order Model Checking? ... Several Obstacles

1. Like standard model checking, higher-order model checking is mostly a whole program analysis. This can seem surprising: higher order is supposed to aid modular structuring of programs!
2. Hitherto HOMC is about computation trees of ground-type functional programs.
   Aim: model check the computation trees of higher-type functional programs (= Böhm trees i.e. trees with binders).
3. Need a denotational model to support compositional model checking, which should be strategy aware (i.e. modelling Böhm trees, and witnesses of correctness properties of Böhm trees), and organisable into a cartesian closed category of parity games.
4. Unfortunately the elegant theorems of "Rabin's Heaven" fail in the world of Böhm trees.

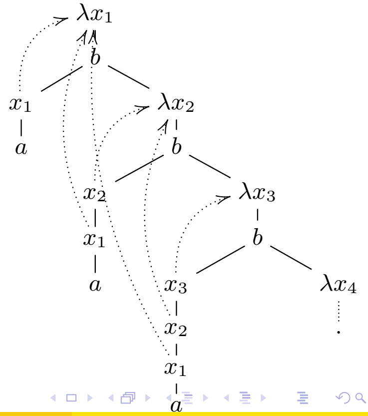
Theorems of "Rabin's Heaven" do not hold for Böhm trees

Let $\Gamma = a : o,\ b : o \to ((o \to o) \to o) \to o$ and

$$\Gamma \ \vdash\ \underbrace{\mathbf{Y}\,(\lambda f.\lambda y^{o}.\lambda x^{o \to o}.b\,(x\,y)\,(f\,(x\,y)))\,a}_{M} \ :\ (o \to o) \to o.$$

$\mathrm{BT}(M)$

- uses infinitely many variable names,
and each variable occurs infinitely often.



Theorems of “Rabin’s Heaven” do not hold for Böhm trees

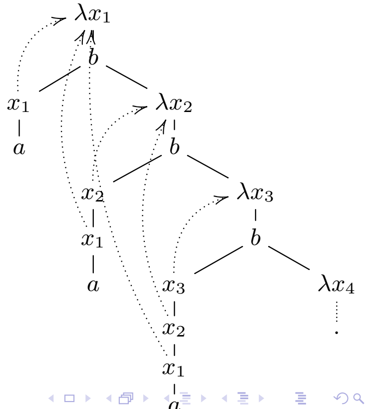
Let $\Gamma = a : o,\ b : o \to ((o \to o) \to o) \to o$ and

$$\Gamma \vdash \underbrace{\mathbf{Y}\,(\lambda f.\lambda y^{o}.\lambda x^{o \to o}.b\,(x\,y)\,(f\,(x\,y)))\,a}_{M} : (o \to o) \to o.$$

$\mathrm{BT}(M)$

- uses infinitely many variable names, and each variable occurs infinitely often.

- has an undecidable MSO theory! (Clairambault & Murawski FSTTCS’13)



An expressive yet decidable logic for higher-type Böhm trees?

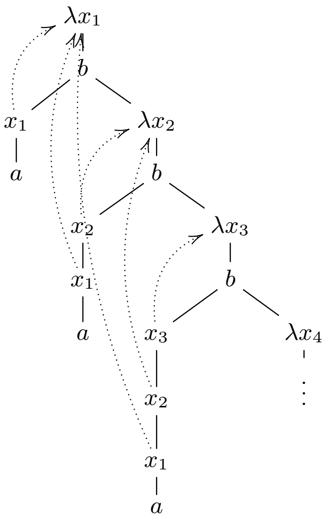
Böhm trees: concrete repr. of higher-order functions on infinite trees.

Take tree property $\mathfrak{P} :=$ "*There are only finitely many occurrences of bound variables in each branch*."

An expressive yet decidable logic for higher-type Böhm trees?

Böhm trees: concrete repr. of higher-order functions on infinite trees.

Take tree property $\mathfrak{P}$:= "*There are only finitely many occurrences of bound variables in each branch*."



Questions: Automata-Logic-Games Correspondence for Higher-Type Trees

1. Is there an expressive logic $\mathcal{L}$ that can describe properties such as $\mathfrak{P}$?
2. Is there a class of automata equi-expressive with $\mathcal{L}$?
3. What kind of games characterise the acceptance problem?
4. Is $\mathcal{L}$ decidable for definable Böhm trees?

Σ**-labelled trees**	**Higher-type Böhm trees**
Alternating Parity Tree Automata	Alternating Dependency Tree Automata - has rules that read λ-binders - generalise Stirling's ADTA to ω-regular winning condition
Mu-calculus $\varphi ::= P \mid \neg\varphi \mid \varphi \vee \psi$ $\mid \nu\alpha.\varphi \mid [i]\varphi$	Higher-type Mu-Calculus - $\varphi ::= \mathbb{v}_x \mid \mathbb{\lambda} x.\psi \mid \cdots$ - $\mathbb{v}_x$ detects variables; $\mathbb{\lambda} x.$- detects λ-binding
Parity Game	Type-Checking Game $\models u : \alpha$

Further, checking these properties against $\lambda^{\rightarrow}\mathbf{Y}$-definable Böhm trees is decidable. (Tsukada & O. LICS14)

Outline

Some Open Problems in Theory of HOMC

1. Equivalence of Recursion Schemes asks whether two given recursion schemes generate the same tree. (Recursively equivalent to Böhm Tree Equivalence of $\lambda\mathbf{Y}$-terms.)
   Is the problem decidable?

Some Open Problems in Theory of HOMC

1. Equivalence of Recursion Schemes asks whether two given recursion schemes generate the same tree. (Recursively equivalent to Böhm Tree Equivalence of $\lambda\mathbf{Y}$-terms.)
   Is the problem decidable?
2. The *Nondeterministic* Safety Conjecture: there is a word language recognisable by a nondeterministic n-CPDA, but not by any nondeterministic HOPDA.
   False for $n = 2$; open for $n \geq 3$.

Some Open Problems in Theory of HOMC

1. Equivalence of Recursion Schemes asks whether two given recursion schemes generate the same tree. (Recursively equivalent to Böhm Tree Equivalence of $\lambda\mathbf{Y}$-terms.)
   Is the problem decidable?
2. The *Nondeterministic* Safety Conjecture: there is a word language recognisable by a nondeterministic n-CPDA, but not by any nondeterministic HOPDA.
   False for $n = 2$; open for $n \geq 3$.
3. Are Unsafe Word Languages Context Sensitive?.
   Answer is Yes for order up to 3 (Kobayashi et al. FoSSaCS14).

Some Open Problems in Theory of HOMC

1. Equivalence of Recursion Schemes asks whether two given recursion schemes generate the same tree. (Recursively equivalent to Böhm Tree Equivalence of $\lambda\mathbf{Y}$-terms.)
   Is the problem decidable?
2. The *Nondeterministic* Safety Conjecture: there is a word language recognisable by a nondeterministic n-CPDA, but not by any nondeterministic HOPDA.
   False for $n = 2$; open for $n \geq 3$.
3. Are Unsafe Word Languages Context Sensitive?.
   Answer is Yes for order up to 3 (Kobayashi et al. FoSSaCS14).
4. Computing Downward Closures of Word Languages of the Higher-Order Collapsible Pushdown Hierarchy.

Some Open Problems in Theory of HOMC

1. Equivalence of Recursion Schemes asks whether two given recursion schemes generate the same tree. (Recursively equivalent to Böhm Tree Equivalence of $\lambda\mathbf{Y}$-terms.)
   Is the problem decidable?
2. The *Nondeterministic* Safety Conjecture: there is a word language recognisable by a nondeterministic n-CPDA, but not by any nondeterministic HOPDA.
   False for $n = 2$; open for $n \geq 3$.
3. Are Unsafe Word Languages Context Sensitive?.
   Answer is Yes for order up to 3 (Kobayashi et al. FoSSaCS14).
4. Computing Downward Closures of Word Languages of the Higher-Order Collapsible Pushdown Hierarchy.
5. Extensions of Higher-Order Model Checking