Automata, Logic and Games: Theory and Application

Lecture 4. Higher-Order Model Checking 2 / 2

Luke Ong

University of Oxford
http://www.cs.ox.ac.uk/people/luke.ong/personal/

TACL Summer School
University of Salerno, 14-19 June 2015

Overview of Part 2

1. Relating Families of Generators of Infinite Trees / Graphs.
2. Algorithmics and Expressivity
3. Reducing Model Checking to Type Inference
4. Compositional Model Checking of Higher-Type Böhm Trees
5. Practical Algorithms for Higher-Order Model Checking
6. Conclusions and Further Directions

Slides are viewable at
`http://www.cs.ox.ac.uk/people/luke.ong/personal/TACL15`

Infinite structures generated by recursion schemes: key questions

1. MSO decidability: Is safety a genuine constraint for decidability? I.e. do trees generated by (arbitrary) recursion schemes have decidable MSO theories?
2. Machine characterisation: Find a hierarchy of automata that characterise the expressive power of recursion schemes.
3. Expressivity: Is safety a genuine constraint for expressivity? I.e. are there inherently unsafe word languages / trees / graphs?
4. Graph families:
    1. Definition: What is a good definition of "graphs generated by recursion schemes"?
    2. Model-checking properties: What are the decidable theories of the graph families?

A Tale of Two Higher-Order Systems

Safe Types (Damm TCS 82) $D_{i+1} := \bigcup_{k\geq 0}[\underbrace{D_i \times \cdots \times D_i}_{k} \to D_i]$ Safey: awkward constraint but has clear algorithmic value	**Simple Types** (Church JSL 40) $\kappa := o \mid \kappa \to \kappa'$ Natural and standard, semantically and in programming
MSO model checking of safe recursion scheme is decidable (KNU 02)	**Q1**: MSO model checking of recursion scheme is decidable (O. 06)
Order-n safe RS $\equiv$ order-n PDA (Damm 82, KNU 02)	**Q2**: Order-n RS $\equiv$?
	Q3: Are there inherently unsafe word languages / trees / graphs?
Hierarchy is strict (Damm 82)	?
Word languages are context-sensitive (Inaba & Maneth 08)	?

Q1. (cont'd) Four different proofs of the MSO decidability result

1. Game semantics and traversals (O. LICS06)
   - variable profiles
2. Collapsible pushdown automata (Hague, Murawski, O. & Serre LICS08)
   - equi-expressivity theorem + rank aware automata
3. Type-theoretic characterisation of APT (Kobayashi & O. LICS09)
   - intersection refinement types
4. Krivine machines (Salvati & Walukiewicz ICALP11)
   - residuals

A common pattern

1. Decision problem equivalent to solving an infinite parity game.
2. Simulate the infinite parity game by a finite parity game.
3. Key ingredient of the game: variable profiles / automaton control-states / intersection types / residuals.

Order-2 collapsible pushdown automata [HOMS, LiCS 08a] are essentially the same as 2PDA with links [AdMO 05] and panic automata [KNUW 05].

Idea: Each stack symbol in 2-stack "remembers" the stack content at the point it was first created (i.e. $push_1$ed onto the stack), by way of a pointer to some 1-stack underneath it (if there is one such).

Two new stack operations: $a \in \Gamma$ (stack alphabet)

- $push_1\, a$: pushes a onto the top of the top 1-stack, together with a pointer to the 1-stack immediately below the top 1-stack.
- $collapse$ (= panic) collapses the 2-stack down to the prefix pointed to by the top_1-element of the 2-stack.

Note that the pointer-relation is preserved by $push_2$.

Example: Urzyczyn's Language U over alphabet $\{\,(,\,),*\,\}$

Definition (Aehlig, de Miranda + O. FoSSaCS 05) A **U-word** has 3 segments:

$$\underbrace{(\cdots(\cdots(}_{A}\;\underbrace{(\cdots)\cdots(\cdots)}_{B}\;\underbrace{*\cdots*}_{C}$$

- Segment A is a prefix of a well-bracketed word that ends in **(**, and the opening **(** is not matched in the entire word.
- Segment B is a well-bracketed word.
- Segment C has length equal to the number of **(** in segment A.

Examples

1. (( ) (( ) (( )) $*\ *\ *$ is a U-word
2. For each $n \geq 0$, we have $(\,(^n\,)^n\,(\,*^{n+2}$ is a U-word.
   Hence by "$uvwxy$ Lemma", U is not context-free.

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* * * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

[[ ]]

([[ ] [a]]

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* \; * * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

	`[[ ]]`
(	`[[ ][` a `]]`
(	`[[ ][` a `][` a a `]]`

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* * * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

	[[ ]]
(	[[ ] [a]]
(	[[ ] [a] [a a]]
)	[[ ] [a] [a]]

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* * * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

	[[ ]]
(	[[ ] [a]]
(	[[ ] [a] [a a]]
)	[[ ] [a] [a]]
(	[[ ] [a] [a] [a a]]

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* * * \in U$

(Ignoring control states for simplicity)

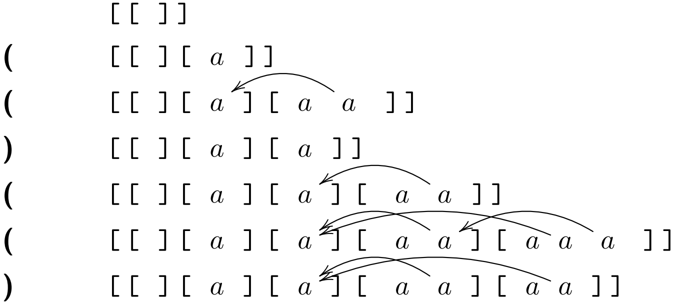
Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

	[[ ]]
(	[[ ] [a]]
(	[[ ] [a] [a a]]
)	[[ ] [a] [a]]
(	[[ ] [a] [a] [a a]]
(	[[ ] [a] [a] [a a] [a a a]]

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* * * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2



Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* * * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

	[[ ]]
(	[[ ] [a]]
(	[[ ] [a] [a a]]
)	[[ ] [a] [a]]
(	[[ ] [a] [a] [a a]]
(	[[ ] [a] [a] [a a] [a a a]]
)	[[ ] [a] [a] [a a] [a a]]
$*$	[[ ] [a] [a]] Collapse!

Recognising U by a (det.) 2CPDA. E.g. $(\, (\,) \, (\, (\,) \, * \, * \, * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

	`[[ ]]`
(	`[[ ][ a ]]`
(	`[[ ][ a ][ a a ]]`
)	`[[ ][ a ][ a ]]`
(	`[[ ][ a ][ a ][ a a ]]`
(	`[[ ][ a ][ a ][ a a ][ a a a ]]`
)	`[[ ][ a ][ a ][ a a ][ a a ]]`
$*$	`[[ ][ a ][ a ]]` Collapse!
$*$	`[[ ][ a ]]`

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $* * * \in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	$collapse$
subsequent $*$	pop_2

	[[ ]]	
(	[[ ] [a]]	
(	[[ ] [a] [a a]]	
)	[[ ] [a] [a]]	
(	[[ ] [a] [a] [a a]]	
(	[[ ] [a] [a] [a a] [a a a]]	
)	[[ ] [a] [a] [a a] [a a]]	
$*$	[[ ] [a] [a]]	Collapse!
$*$	[[ ] [a]]	
$*$	[[ ]]	

What does the height of the top 1-stack measure?

Is order-n CPDA strictly more expressive than order-n PDA?

Equivalently, does the *collapse* operation add any expressive power?

Lemma (AdMO FoSSaCS05): Urzyczyn's language U is quite telling!

1. U is *not* recognised by any 1PDA.
2. U is recognised by a non-deterministic 2PDA.
3. U is recognised by a deterministic 2CPDA.

Question

1. *Is U recognisable by a deterministic 2PDA?*
2. *More generally, is U recognisable by a deterministic nPDA for any n?*

If answer (to 1) is no, then there is an associated tree that is generated by an order-2 recursion scheme, but not by any order-2 safe recursion scheme.

Q2: Machine characterization: order-n RS = order-n CPDA

Theorem (Equi-expressivity [Hague, Murawski, O. & Serre LICS08])

For each $n \geq 0$, order-n collapsible PDA and order-n recursion schemes are equi-expressive for Σ-labelled trees.

Proof idea

- From recursion scheme to CPDA: Use game semantics [Hyland & O. 00] Idea: code traversals as n-stacks.
  Invariant: The top 1-stack is the P-view of the encoded traversal.
  For a direct proof (no game semantics), see [Carayol & Serre LICS12].
- From CPDA to recursion scheme:
  Code CPDA configuration c as a term M_c, so that c transitions to c' in CPDA implies M_c rewrites to $M_{c'}$.

Order-n CPDA are a machine characterization of order-n simply-typed lambda calculus with recursion.

Q3: Is safety a genuine constraint on expressivity? (1)

Question (Safety, KNW FoSSaCS02)

Are there inherently unsafe word languages / trees / graphs?

Word languages? Yes

Theorem (Parys STACS11, LICS12)

There is a language (essentially U) recognised by a deterministic 2CPDA but not by any deterministic nPDA for all $n \geq 0$.

Proof uses a powerful pumping lemma for HOPDA.

Another pumping lemma for nCPDA is used to prove a hierarchy theorem for collapsible graphs and unsafe trees [Kartzow & Parys, MFCS12].

Kobayashi (LICS13) gives a simpler proof of a pumping lemma (hence hierarchy theorem) using intersection types.

Q3: Is safety a genuine constraint on expressivity? (2)

Are there inherently unsafe trees?
Yes

Theorem (Parys STACS11, LICS12)

There is a tree generated by an order-2 recursion scheme, but not by a safe order-n RS, for any n.

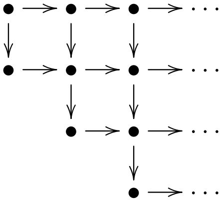
The tree is constructed from language U.

Are there inherently unsafe graphs? Yes.

Theorem (Hague, Murawski, O and Serre LICS08)

Solvability of parity games over nCPDA graphs is n-EXPTIME complete.

There is an MSO interpretation of the configuration graph of a 2CPDA configuration graph that give the "infinite half grid" which has an undecidable MSO theory.



Corollary There is a 2CPDA whose configuration graph (semi-infinite grid) is not that of any nPDA, for any n.

A safety question for non-determinacy

Question (Safety for Non-determinacy)

Is there a word language recognised by a order-n CPDA which is not recognisable by any non-deterministic higher-order PDA?

For order 2, the answer is no.

Theorem (Aehlig, de Miranda and O. FoSSaCS 2005)

For every order-2 recursion scheme, there is a safe non-deterministic order-2 recursion scheme that generates the same word language.

Summary: A Tale of Two Higher-Order Systems

Safe Types (Damm 82) $D_{i+1} := \bigcup_{k\geq 1}[\underbrace{D_i \times \cdots \times D_i}_{k} \to D_i]$	**Types** (Church JSL 40) $\kappa := o \mid \kappa \to \kappa'$
MSO model checking of safe recursion schemes is decidable [KNU 02]	**Q1**: MSO model checking of recursion schemes is decidable [O. 06]
Order-n safe RS $\equiv$ order-n PDA [Damm 82, KNU 01]	**Q2**: Order-n RS $\equiv$ order-n CPDA [Hague, Murawski, O. & Serre 08]
	Q3a: Inherently unsafe trees exist. [Parys 12]
	Q3b: Inherently unsafe graphs exist. [Hague, Murawski, O. & Serre 08]
Hierarchy is strict [Damm 82]	Hierarchy is strict [Kartzow & Parys 12]
Word languages are context-sensitive [Inaba & Maneth 08]	Order-3 unsafe word languages are context-sensitive (Kobayashi et al. 14)

Parity Games, Mu-Calculus and APT are Equivalent

Mu-Calculus: modal logic extended with least and greatest fixpoint operators. (Scott, de Bakker; Kozen 82)
Mu-calculus and MSOL are equi-expressive for tree languages (Niwinski ?).
Mu-calculus is the bisimulation-invariant fragment of MSOL (JW 96).

Mu-calculus Model Checking Problem and PARITY are inter-reducible

- $\Rightarrow$: Essentially: Fundamental Semantic Theorem [Streett and Emerson Info & Comp 1989]
- $\Leftarrow$: E.g. [Walukiewicz Info & Comp 2001]

Mu-calculus and APT are effectively equi-expressive for tree languages [Emerson & Jutla FoCS 91]

- $\Rightarrow$: E.g. [Kupferman & Vardi JACM 2000]
- $\Leftarrow$: E.g. [Walukiewicz Info & Comp 2001]

A Type System Characterising MSO Decidability

Theorem (Kobayashi + O. LiCS 2009)

Given an APT $\mathcal{A}$ (equivalently MSO or mu-calculus formula) there is a type system $\mathcal{K}_{\mathcal{A}}$ such that for every HORS G, $\mathsf{tree}(G)$ *is accepted by $\mathcal{A}$ iff G is $\mathcal{K}_{\mathcal{A}}$-typable.*

Refinement intersection types embedded with states and priorities

Fix an APT $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$.
Construct refinement types from states $q \in Q$ and priorities m_i.

Refinement type (simply type)	θ	$::=$	$q \mid \tau \to \theta$
Intersection	τ	$::=$	$\bigwedge_{i=1}^{l} (\theta_i, m_i)$

Thus a refinement type has the form

$$\tau_1 \to \cdots \to \tau_n \to q$$

where each τ_i is an intersection. Write $\top = \bigwedge \varnothing$.
Extend Ω to a priority map on types:

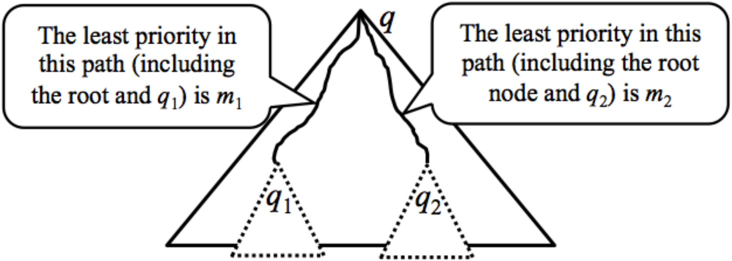
$$\Omega(\tau_1 \to \cdots \to \tau_n \to q) := \Omega(q).$$

Intuition

Regard automaton states as the base types i.e. types of trees

- q is the type of trees accepted by the automaton from state q
- $q_1 \wedge q_2$ is the type of trees accepted from both q_1 and q_2
- $\tau \rightarrow q$ is the type of functions that take a tree of type τ and return a tree of type q

A tree function described by $(q_1, m_1) \wedge (q_2, m_2) \rightarrow q$.



(The above is a tree context of a run-tree, not the generated tree.)

Typing judgments $\Gamma \vdash t : \theta$

where environment Γ is a finite set of bindings of the form $F : (\theta, m)$ or $x : (\theta, m)$, and $F \in \mathcal{N}$ ranges over function symbols of the HORS.

Typing System $\mathcal{K}_{\mathcal{A}}$: Validity is defined by induction over four rules.

$$\frac{}{x : (\theta, \Omega(\theta)) \vdash x : \theta} \quad \text{(T-VAR)}$$

$$\frac{\Gamma, x : \bigwedge_{i \in I}(\theta_i, m_i) \vdash t : \theta \qquad I \subseteq J}{\Gamma \vdash \lambda x.t : \bigwedge_{i \in J}(\theta_i, m_i) \to \theta} \quad \text{(T-ABS)}$$

$$\frac{\Gamma_0 \vdash s : \bigwedge_{i=1}^{k}(\theta_i, m_i) \to \theta \qquad \Gamma_i \vdash t : \theta_i \ (\forall i \in \{1, \ldots, k\})}{\Gamma_0 \cup (\Gamma_1 \uparrow m_1) \cup \cdots \cup (\Gamma_k \uparrow m_i) \vdash s\ t : \theta} \quad \text{(T-APP)}$$

where $\Gamma \uparrow m := \{ F : (\theta, \max(m, m')) \mid F : (\theta, m') \in \Gamma \}$.
Note: no weakening.)

A Typing Parity Game: Assume HORS G & APT $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$

Verifier's vertices are bindings $F : (\theta, m)$, with priority $\Omega(\theta)$.
Refuter's vertices are environments, ranged over by Γ, with priority 0.
Edge set of the (bipartite) digraph is defined as:

$$\{ (F : (\theta, m),\ \Gamma) \mid \Gamma \vdash rhs(F) : \theta \} \cup \{ (\Gamma,\ F : (\theta, m)) \mid F : (\theta, m) \in \Gamma \}$$

This defines a finite parity game.

Idea: Verifier makes typing assertions; Refuter challenges the assumptions (bindings in environment).

- Start vertex: $S : (q_I, \Omega(q_I))$.
- Given a binding $F : (\theta, m)$, Verifier chooses an environment Γ such that $\Gamma \vdash rhs(F) : \theta$ is valid.
- Given Γ, Refuter chooses a binding $F : (\theta, m)$ in Γ, and challenges Verifier to prove that F has type θ.

We say that G is typable just if Verifier has a winning strategy.

Reducing APT Model Checking to a Typing Parity Game

Theorem (Correctness)

Given an APT $\mathcal{A}$ there is a type system $\mathcal{K}_{\mathcal{A}}$ such that for every HORS G, tree(G) is accepted by $\mathcal{A}$ iff G is $\mathcal{K}_{\mathcal{A}}$-typable.

Parameterised complexity: There is a fixed-parameter polytime (in the size of HORS) type inference algorithm for $\mathcal{K}_A$.

Using an upper bound for PARITY, the runtime[1] is

$$O(r^{1+\lfloor p/2 \rfloor}\,\mathbf{exp}_n((a \cdot |Q| \cdot p)^{1+\epsilon}))$$

where

- n and r are respectively the order and number of rules of the HORS
- a is largest arity of the types
- p and $|Q|$ are respectively the number of priority and states of the APT.

[1] $\mathbf{exp}_0(x) = x$; $\mathbf{exp}_{k+1}(x) := 2^{\mathbf{exp}_k(x)}$.

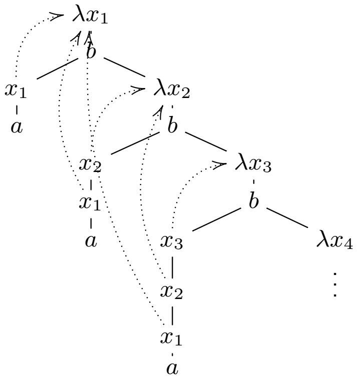
Compositional Higher-Order Model Checking? ... Several Obstacles

1. Like (most of) standard model checking, higher-order model checking is a whole program analysis. This can seem surprising: higher order is *supposed* to aid modular structuring of programs!
2. Parity games are central to model checking (of reactive systems). We don't know how to compose parity game.
   A foundational problem for higher-order model checking: we lack a cartersian closed category of parity games!
3. There are several algorithms for model checking ground-type trees (= trees without binders). But we do not know how to model check higher-order trees i.e. Böhm trees.
4. The elegant theorems of "Rabin's Heaven" fail in the world of Böhm trees.
   What is the appropriate (decidable) logical theory for Böhm trees?

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

1. A $\lambda\mathbf{Y}$-definable Böhm tree with undecidable MSO theory
   (Salvati; Clairambault & Murawski FSTTCS 2013)

$$\mathrm{BT}(\mathbf{Y}\ (\lambda f.\lambda y^{o}.\lambda x^{o\to o}.b\,(x\,y)\,(f\,(x\,y)))\,a)$$



However the question whether a given $\lambda\mathbf{Y}$-definable Böhm tree has a given intersection type is decidable.

E.g. the property "*there are only finitely many occurrences of bound variables in each branch*" is describable as an intersection type.

2. Emptiness of Stirling's alternating dependency tree automata—a compelling device for analysing Böhm trees—is undecidable.
   (O. & Tzevelekos LICS 2009)

Type-Checking Game

$$U \models \tau$$

"Verifier has a winning strategy in the game that checks Böhm tree U has type τ"

Formulas τ are a slight variant of the types in (Kobayashi & O. LICS 2009), parameterised by base types Q, and a winning condition $(\mathbb{E}, \mathbb{F}, \Omega)$, which is an algebraic abstraction of the ω-regular winning conditions:

prime types	τ, σ	$::=$	$q \mid \alpha \to \tau$
intersection types	α, β	$::=$	$\bigwedge_{i \in I} \langle\, \tau_i ; e_i \,\rangle$

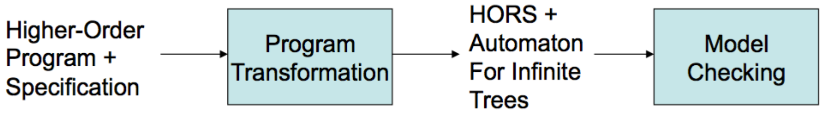
where $q \in Q$, $e_i \in \mathbb{E}$ (effect set), and I is a finite indexing set.

Some Results (Tsukada & O. LICS 2014)

1. $\models$ conservatively extends the MSO properties of trees (without binders).
2. Decidability of $\lambda\mathbf{Y}$-definable Böhm trees: It is decidable, given a $\lambda\mathbf{Y}$-term M and τ, whether $\models \mathrm{BT}(M) : \tau$.
3. Two-Level Compositionality:
   - If Böhm trees U and V are composable, then the set of properties (i.e. types) of $U \circ V$ is completely determined by those of U and of V.
   - Further if $\models U : \tau$ and $\models V : \sigma$ imply $\models U \circ V : \delta$, then the winning strategies $\mathfrak{s}_U^\tau$ of $\models U : \tau$ and $\mathfrak{s}_V^\sigma$ of $\models V : \sigma$ are composable, and yield a winning strategy $\mathfrak{s}_U^\tau \circ \mathfrak{s}_V^\sigma$ of $\models U \circ V : \delta$.
4. Effective Selection: If $\models \mathrm{BT}(M) : \tau$ then there exists, constructively, a $\lambda\mathbf{Y}$-definable winning strategy of $\models \mathrm{BT}(M) : \tau$.
5. Transfer Theorem: $\Gamma \vdash M : \tau$ iff $\Gamma \models \mathrm{BT}(M) : \tau$.

Underpinning the above is a cartesian closed category of ω-regular games. They give a "runnable" or strategy-aware model, which can be used to model check higher-type Böhm trees.

Verification Problem: "Does $\mathcal{P}$ satisfy specification φ?"



Safety Verification by Reduction to Higher-Order Model Checking [Kobayashi POPL09]

This method is fully automatic, sound and complete for

- functional boolean programs (simply-typed λ-calculus + recursion + finite base types)
- many verification problems; e.g. resource usage, reachability and flow analysis.

Practical[2] Algorithms for Model Checking HORS

Brute-force search of the state space will not work!

Assume a 2-state automaton.

Order	Types κ	# Refinement Types of κ $\rho(\kappa)$
0	o	2
1	$o \to o$	$2^2 \times 2 = 8$
2	$(o \to o) \to o$	$2^8 \times 2 = 512$
3	$((o \to o) \to o) \to o$	$2^{513} \approx 10^{154}$ $\gg$ # atoms in universe!

Note: $\rho(\kappa_1 \to \kappa_2) = 2^{\rho(\kappa_1)} \times \rho(\kappa_2)$

An intensively active and competitive research topic: are there practical algorithms for model checking HORS?

Working Hypothesis: The worst-case complexity is realised only by pathological / contrived examples, not by programs that humans write.

[2]On realistic examples, terminate in minutes rather than months or years.

HOMC Algorithms

Recall different proofs of the MSO decidability of HORS:

(G) Game semantics [O. LICS06]

(C) Collapsible PDA [Hague, Murawski, O. & Serre LICS08]

(T) Intersection refinement types [Kobayashi POPL09; K. & O. LICS09]

Each has been the basis of attempts to construct practical algorithms.

Algorithm	Basis	Properties	Propagation	
TRecS	T	trivial	forward	Tohoku, 2009
GTRecS1 & 2	G	trivial	forward	Tohoku, 2011
TravMC	G	trivial	forward	Oxford, 2012
C-SHORe	C	co-trivial	backward	RHL, TUM, LIAFIA, '13
HorSat	C	co-trivial	backward	Tokyo, 2013
HorSatT	C/T	trivial	mixed	Tokyo, 2013

None of the above can scale robustly beyond HORS of a few hundred rules!

Preface[3] [Ramsay, Neatherway & Ong 2013]

Based on refinement types, but uses abstraction refinement.

Input: HORS G, alternating trivial automaton $\mathcal{A} = \langle\, \Sigma, Q, \delta, q_I \,\rangle$
Output: YES if $\mathcal{A}$ accepts tree(G); NO otherwise.

Preface constructs an eventually stable sequence of type contexts $(C_i)_{i \in \omega}$:

$$\begin{aligned} C_0 &= \langle\, \Gamma^0_\exists, \Gamma^0_\forall \,\rangle &&= \langle\, \varnothing, \varnothing \,\rangle \\ C_{k+1} &= \langle\, \Gamma^{k+1}_\exists, \Gamma^{k+1}_\forall \,\rangle &&= \langle\, \Gamma^k_\exists \uplus \mathsf{env}_A(C_k),\ \Gamma^k_\forall \uplus \mathsf{env}_R(C_k) \,\rangle \end{aligned}$$

with limit $C = \langle\, \Gamma_\exists, \Gamma_\forall \,\rangle$. If $S : q_I \in \Gamma_\exists$ return YES: return NO otherwise.

Invariant: For each $k \geq 0$

- Verifier has a winning strategy in typing parity game induced by $(\Gamma^k_\exists, \mathcal{A})$.
- Verifier has a winning strategy in typing parity game induced by $(\Gamma^k_\forall, \neg\mathcal{A})$.

Variant (Termination). $(S : q_I \in \Gamma_\exists) \ \vee \ (\mathsf{env}_R(C_k) \setminus \Gamma_\forall \neq \varnothing)$.

[3] http://mjolnir.cs.ox.ac.uk/web/preface

Category 1 Benchmarks (Times in seconds.)

Benchmark	Rules	Order	Decision	Preface	TRecS
map_filter-e	64	5	R	0.53	0.01
fold_left	65	4	A	0.39	0.03
fold_right	65	4	A	0.39	0.03
forall_eq_pair	66	4	A	0.39	0.03
forall_leq	66	4	A	0.39	0.03
a-cppr	74	3	R	0.38	0.01
search-e	96	5	R	0.90	0.01
search	119	4	A	0.46	1.04
map_filter	143	5	A	0.51	0.13
risers	148	5	A	0.44	0.33
r-file	156	2	A	0.82	1.50
fold_fun_list	197	6	A	0.44	0.89
zip	210	3	A	0.58	15.10

JIT compilation on Mono incurs a performance overhead.

When compiled ahead-of-time on Windows, Preface solves all the above in < 0.05 sec, though still slightly slower than TRecS.

General Trend: Preface overtakes TRecS for larger HORS (> 200 rules).

Category 2 Benchmarks

Benchmark	Rules	Order	Preface	HorSat	HorSatT	C-SHORe	GTRecS2
cfa-psdes	237	7	0.51	0.28	1.81	3.44	–
cfa-matrix-1	383	8	0.61	0.73	6.30	18.58	–
cfa-life2	898	14	1.46	5.94	–	–	–

Instances arising from a control flow analysis tool. cfs-life2 has arity 29!

Category 3 Benchmarks

Benchmark	Rules	Order	Preface	HorSat	HorSatT	C-SHORe	GTRecS2
exp2-1600	1606	2	8.39	–	–	–	10.47
exp2-3200	3206	2	17.51	–	–	–	59.13
exp2-6400	6406	2	39.58	–	–	–	–
exp2-12800	12806	2	92.19	–	–	–	–
exp4-400	408	4	14.12	–	106.53	–	–
exp4-800	808	4	30.55	–	–	–	–
exp4-1600	1608	4	71.06	–	–	–	–
exp4-3200	3208	4	–	–	–	–	–

These order-n RS generate $\mathbf{exp}_n$-sized trees (hence exercising their full power); their certificates are proportional to the number of rules.

"–" means TIMEOUT; set to 2 mins.

Conclusion: Preface scales readily to thousands of rules, well-beyond the capabilities of state-of-the-art HOMC tools.

Reality check: How far are we from verifying all of (say) Haskell?

HORS do not model:

1. algebraic data types and infinite data structures (e.g. integers)
2. function definition by pattern matching.

An approach based on pattern-matching recursion schemes (PMRS) [O. & Ramsay POPL11, ICFP12]

- PMRS is a good model of functional programs: PMRS is essentially the IR of Glasgow Haskell Compiler less the F_ω-types
- Verification problem is undecidable: use static (flow) analysis + higher-order model checking + CEGAR loop.

Realistic Goal: Verify thousands of SLOC in seconds; or verify Haskell libraries in tens of seconds.

Questions: How does the model checking compare with (i) other approaches to verify functional programs? (ii) model checking of C programs?

Conclusions

- Higher-order model checking is challenging and worthwhile.
- HORS are a robust and highly expressive grammar for infinite trees. They have rich algorithmic properties.
- Recent progress in the theory have benefitted from semantic methods (game samantics and types), in conjunction with more standard techniques from algorithmic verification.
- Despite prohibitive (hyper-exponential) complexity, there is growing evidence that practical HOMC algorithms are possible.