Overview of Part 2

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.org/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

<table>
<thead>
<tr>
<th>Safe Types (Damm TCS 82)</th>
<th>Simple Types (Church JSL 40)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{i+1} := \bigcup_{k \geq 0}[D_i \times \cdots \times D_i \to D_i]$</td>
<td>$\kappa := o \mid \kappa \to \kappa'$</td>
</tr>
<tr>
<td>Safey: awkward constraint but has clear algorithmic value</td>
<td>Natural and standard, semantically and in programming</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MSO model checking of safe recursion scheme is decidable (KNU 02)</th>
<th>Q1: MSO model checking of recursion scheme is decidable (O. 06)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order-$n$ safe RS $\equiv$ order-$n$ PDA (Damm 82, KNU 02)</td>
<td>Q2: Order-$n$ RS $\equiv$ ?</td>
</tr>
<tr>
<td>Q3: Are there inherently unsafe word languages / trees / graphs?</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hierarchy is strict (Damm 82)</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word languages are context-sensitive (Inaba &amp; Maneth 08)</td>
<td>?</td>
</tr>
</tbody>
</table>
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.
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 $\{(,),\ast\}$

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

$$
(\cdots(\cdots)\cdots(\cdots)\cdots)(\cdots)\ast\cdots\ast
$$

- 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. $((()((()())()())\ast\ast\ast$ is a $U$-word
2. For each $n \geq 0$, we have $((n)^n(\ast)^{n+2}$ 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)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>$\text{push}_2 ~; ~ \text{push}_1 a$</td>
</tr>
<tr>
<td>first *</td>
<td>$\text{pop}_1$</td>
</tr>
<tr>
<td>subsequent *</td>
<td>$\text{collapse}$</td>
</tr>
<tr>
<td></td>
<td>$\text{pop}_2$</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ll}
    [ [  ] ] \\
    ( [ [ ] [ a ] ] )
\end{array}
\]
Recognising $U$ by a (det.) 2CPDA. E.g. $(())(()) * * * \in U$
(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>$()$</td>
<td>$\text{push}_2; \text{push}_1 a$</td>
</tr>
<tr>
<td>first *</td>
<td>$\text{pop}_1$</td>
</tr>
<tr>
<td>subsequent *</td>
<td>$\text{collapse}$</td>
</tr>
<tr>
<td></td>
<td>$\text{pop}_2$</td>
</tr>
</tbody>
</table>

| [ [ ] ] |
| ( [ [ ] [ a ] ] |
| ( [ [ ] [ a ] [ a a ] ] |

$\text{Luke Ong (University of Oxford)}$

Higher-Order Model Checking

14-19 June 2015 9 / 46
Recognising $U$ by a (det.) 2CPDA. E.g. $((())(())***) \in U$
(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>$()$</td>
<td>$\text{push}_2$ ; $\text{push}_1 a$</td>
</tr>
<tr>
<td>first $*$</td>
<td>$\text{pop}_1$</td>
</tr>
<tr>
<td>subsequent $*$</td>
<td>$\text{collapse}$</td>
</tr>
<tr>
<td></td>
<td>$\text{pop}_2$</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ll}
[[ ]] \\
( [[ ] [ a ]] ) \\
( [[ ] [ a ] [ a a ] ] ) \\
) [[ ] [ a ] [ a ] ]
\end{array}
\]
Recognising $U$ by a (det.) 2CPDA. E.g. $(())(())* * * \in U$
(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>$()$</td>
<td>$\text{push}_2$ ; $\text{push}_1a$</td>
</tr>
<tr>
<td>first $*$</td>
<td>$\text{pop}_1$</td>
</tr>
<tr>
<td>subsequent $*$</td>
<td>$\text{collapse}$</td>
</tr>
<tr>
<td></td>
<td>$\text{pop}_2$</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
[[[ ]]] \\
( & [[ [ a ]]] \\
( & [[ [ a ] [ a ]]] \\
) & [[ [ a ] [ a ]]] \\
( & [[ [ a ] [ a ] [ a ]]] \\
( & [[ [ a ] [ a ] [ a a ]]] \\
( & [[ [ a ] [ a ] [ a a a ]]]
\end{align*}
\]
Recognising $U$ by a (det.) 2CPDA. E.g. $(()) (()) * * * \in U$
(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>${}$</td>
<td>$\text{push}_2 \ ; \ \text{push}_1 a$</td>
</tr>
<tr>
<td>first *</td>
<td>$\text{pop}_1$</td>
</tr>
<tr>
<td>subsequent *</td>
<td>$\text{collapse}$</td>
</tr>
<tr>
<td></td>
<td>$\text{pop}_2$</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\text{[( ][ ])} \\
( \begin{array}{c}
[ ][ ] [ a ] \\
[ ][ ] [ a ] [ a a ] \\
( \begin{array}{c}
[ ][ ] [ a ] [ a ] \\
[ ][ ] [ a ] [ a a ] \\
( \begin{array}{c}
[ ][ ] [ a ] [ a ] [ a a ] \\
[ ][ ] [ a ] [ a a ] [ a a a ] \\
( \begin{array}{c}
[ ][ ] [ a ] [ a a ] [ a a a ] \\
\end{array}\end{array}\end{array}\end{array}\end{array}\end{array}\end{array}
\]
Recognising $U$ by a (det.) 2CPDA. E.g. $( ( ) ( ( ) * * * ) \in U$
(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>${}$</td>
<td>$\text{push}_2$ ; $\text{push}_1a$</td>
</tr>
<tr>
<td>first $*$</td>
<td>$\text{pop}_1$</td>
</tr>
<tr>
<td>subsequent $*$</td>
<td>$\text{collapse}$</td>
</tr>
<tr>
<td></td>
<td>$\text{pop}_2$</td>
</tr>
</tbody>
</table>

Luke Ong (University of Oxford)
Higher-Order Model Checking
14-19 June 2015 13 / 46
Recognising $U$ by a (det.) 2CPDA. E.g. $( ( ) ( ( ) * * * ) ) \in U$

(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
</table>
| $)$ first $*$ | $\text{push}_2$ ; $\text{push}_1 a$
| subsequent $*$ | $\text{pop}_1$
| | $\text{collapse}$
| | $\text{pop}_2$

\[
\begin{array}{c}
[ [ ] ] \\
( [ [ ] [ a ] ] ) \\
( [ [ ] [ a ] [ a ] ] ) \\
) [ [ ] [ a ] [ a ] ] \\
( [ [ ] [ a ] [ a ] [ a ] ] ) \\
( [ [ ] [ a ] [ a ] [ a ] [ a ] ] ) \\
) [ [ ] [ a ] [ a ] [ a ] [ a ] ] \\
* [ [ ] [ a ] [ a ] ]
\end{array}
\]

Collapse!
Recognising $U$ by a (det.) 2CPDA. E.g. $(()) (()) * * * \in U$
(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>${}$</td>
<td>$push_2; push_1 a$</td>
</tr>
<tr>
<td>first $*$</td>
<td>$pop_1$</td>
</tr>
<tr>
<td>subsequent $*$</td>
<td>$collapse$</td>
</tr>
<tr>
<td></td>
<td>$pop_2$</td>
</tr>
</tbody>
</table>

$[ [ ] ]$

$( [ [ ] [ a ] ] )$

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

$) [ [ ] [ a ] [ a ] ]$

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

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

$) [ [ ] [ a ] [ a ] [ a a ] [ a a ] ]$

$* [ [ ] [ a ] [ a a ] ]$

$* [ [ ] [ a ] ]$

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

(Ignoring control states for simplicity)

<table>
<thead>
<tr>
<th>Upon reading</th>
<th>Do</th>
</tr>
</thead>
</table>
| \{ \} first * subsequent * | $\text{push}_2$ ; $\text{push}_1a$
| | $\text{pop}_1$
| | $\text{collapse}$
| | $\text{pop}_2$

What does the height of the top 1-stack measure?
Is order-\(n\) CPDA strictly more expressive than order-\(n\) PDA?

Equivalently, does the \textit{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 \(n\)PDA 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 \textit{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 \(\Sigma\)-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 $n$PDA for all $n \geq 0$.*

Proof uses a powerful pumping lemma for HOPDA.

Another pumping lemma for $n$CPDA 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.
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 $n$CPDA graphs is $n$-EXPTIME complete.

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

![Diagram of infinite half grid configuration graph]

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

<table>
<thead>
<tr>
<th>Safe Types (Damm 82)</th>
<th>Types (Church JSL 40)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{i+1} := \bigcup_{k \geq 1} [D_i \times \cdots \times D_i \rightarrow D_i]$</td>
<td>$\kappa := o \mid \kappa \rightarrow \kappa'$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MSO model checking of safe recursion schemes is decidable [KNU 02]</th>
<th>Q1: MSO model checking of recursion schemes is decidable [O. 06]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order-$n$ safe RS $\equiv$ order-$n$ PDA [Damm 82, KNU 01]</td>
<td>Q2: Order-$n$ RS $\equiv$ order-$n$ CPDA [Hague, Murawski, O. &amp; Serre 08]</td>
</tr>
<tr>
<td>Q3a: Inherently unsafe trees exist. [Parys 12]</td>
<td>Q3b: Inherently unsafe graphs exist. [Hague, Murawski, O. &amp; Serre 08]</td>
</tr>
<tr>
<td>Hierarchy is strict [Damm 82]</td>
<td>Hierarchy is strict [Kartzow &amp; Parys 12]</td>
</tr>
<tr>
<td>Word languages are context-sensitive [Inaba &amp; Maneth 08]</td>
<td>Order-3 unsafe word languages are context-sensitive (Kobayashi et al. 14)</td>
</tr>
</tbody>
</table>
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**

⇒: Essentially: Fundamental Semantic Theorem [Streett and Emerson Info & Comp 1989]

⇐: E.g. [Walukiewicz Info & Comp 2001]

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

⇒: E.g. [Kupferman & Vardi JACM 2000]

⇐: E.g. [Walukiewicz Info & Comp 2001]
Theorem (Kobayashi + O. LiCS 2009)

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

$$\theta ::= q \mid \tau \rightarrow \theta$$

### Intersection

$$\tau ::= \bigwedge_{i=1}^{l}(\theta_i, m_i)$$

Thus a refinement type has the form

$$\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow q$$

where each $\tau_i$ is an intersection. Write $\top = \bigwedge \emptyset$.

Extend $\Omega$ to a priority map on types:

$$\Omega(\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow q) := \Omega(q).$$
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 \( q \)
- \( q_1 \land q_2 \) 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 \( \tau \) and return a tree of type \( q \)

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

(The above is a tree context of a run-tree, not the generated tree.)
where environment $\Gamma$ 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 $\mathcal{K}_A$: Validity is defined by induction over four rules.

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

\[
\frac{
    \Gamma, x : \bigwedge_{i \in I} (\theta_i, m_i) \vdash t : \theta \quad I \subseteq J
}{
    \Gamma \vdash \lambda x.t : \bigwedge_{i \in J} (\theta_i, m_i) \rightarrow \theta
}
\]

\[
\frac{
    \Gamma_0 \vdash s : \bigwedge_{i=1}^k (\theta_i, m_i) \rightarrow \theta \quad \Gamma_i \vdash t : \theta_i \ (\forall i \in \{1, \ldots, k\})
}{
    \Gamma_0 \cup (\Gamma_1 \uparrow m_1) \cup \cdots \cup (\Gamma_k \uparrow m_i) \vdash s \ t : \theta
}
\]

(T-APP)

where $\Gamma \uparrow m := \{ F : (\theta, \max(m, m')) \mid F : (\theta, m') \in \Gamma \}$. 

Note: no weakening.)
Verifier’s vertices are bindings \( F : (\theta, m) \), with priority \( \Omega(\theta) \). Refuter’s vertices are environments, ranged over by \( \Gamma \), with priority 0. Edge set of the (bipartite) digraph is defined as:

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

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 \( \Gamma \) such that \( \Gamma \vdash \text{rhs}(F) : \theta \) is valid.
- Given \( \Gamma \), Refuter chooses a binding \( F : (\theta, m) \) in \( \Gamma \), and challenges Verifier to prove that \( F \) has type \( \theta \).

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 $A$ there is a type system $K_A$ such that for every HORS $G$, $\text{tree}(G')$ is accepted by $A$ iff $G$ is $K_A$-typable.

**Parameterised complexity**: There is a fixed-parameter polytime (in the size of HORS) type inference algorithm for $K_A$.

Using an upper bound for PARITY, the runtime\(^1\) is

$$O(r^{1+[p/2]} \exp_n((a \cdot |Q| \cdot p)^{1+\epsilon}))$$

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.

\[1\exp_0(x) = x; \exp_{k+1}(x) := 2^{\exp_k(x)}.\]
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!

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 cartesian closed category of parity games!

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.

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 \(\lambda Y\)-definable Böhm tree with undecidable MSO theory
   (Salvati; Clairambault & Murawski FSTTCS 2013)

\[
\text{BT}(Y (\lambda f. \lambda y^o. \lambda x^{o\rightarrow o}. b (x y) (f (x y)))) a)
\]

However the question whether a given \(\lambda Y\)-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 \( \tau \)”

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

- **Prime types**: \( \tau, \sigma ::= q \mid \alpha \rightarrow \tau \)
- **Intersection types**: \( \alpha, \beta ::= \bigwedge_{i \in I} \langle \tau_i; e_i \rangle \)

where \( q \in Q \), \( e_i \in \mathcal{E} \) (effect set), and \( I \) is a finite indexing set.
1. $|=\text{conservatively extends}$ the MSO properties of trees (without binders).

2. **Decidability of $\lambda Y$-definable Böhm trees:** It is decidable, given a $\lambda Y$-term $M$ and $\tau$, whether $|=\text{BT}(M) : \tau$.

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 $|= U : \tau$ and $|= V : \sigma$ imply $|= U \circ V : \delta$, then the winning strategies $s^\tau_U$ of $|= U : \tau$ and $s^\sigma_V$ of $|= V : \sigma$ are composable, and yield a winning strategy $s^\tau_U \circ s^\sigma_V$ of $|= U \circ V : \delta$.

4. **Effective Selection:** If $|=\text{BT}(M) : \tau$ then there exists, constructively, a $\lambda Y$-definable winning strategy of $|=\text{BT}(M) : \tau$.

5. **Transfer Theorem:** $\Gamma \vdash M : \tau$ iff $\Gamma \models\text{BT}(M) : \tau$.

Underpining the above is a cartesian closed category of $\omega$-regular games. They give a “runnable” or strategy-aware model, which can be used to model check higher-type Böhm trees.
Verification Problem: “Does $P$ satisfy specification $\varphi$?”

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 $\lambda$-calculus + recursion + finite base types)
- many verification problems; e.g. resource usage, reachability and flow analysis.
Brute-force search of the state space will not work!

Assume a 2-state automaton.

<table>
<thead>
<tr>
<th>Order</th>
<th>Types</th>
<th># Refinement Types of $\kappa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$o$</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>$o \to o$</td>
<td>$2^2 \times 2 = 8$</td>
</tr>
<tr>
<td>2</td>
<td>$(o \to o) \to o$</td>
<td>$2^8 \times 2 = 512$</td>
</tr>
<tr>
<td>3</td>
<td>$((o \to o) \to o) \to o$</td>
<td>$2^{513} \approx 10^{154}$ $\gg$ # atoms in universe!</td>
</tr>
</tbody>
</table>

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.

\[^2\text{On realistic examples, terminate in minutes rather than months or years.}\]
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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Basis</th>
<th>Properties</th>
<th>Propagation</th>
<th>Venue</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRecS</td>
<td>T</td>
<td>trivial</td>
<td>forward</td>
<td>Tohoku, 2009</td>
</tr>
<tr>
<td>GTRecS1 &amp; 2</td>
<td>G</td>
<td>trivial</td>
<td>forward</td>
<td>Tohoku, 2011</td>
</tr>
<tr>
<td>TravMC</td>
<td>G</td>
<td>trivial</td>
<td>forward</td>
<td>Oxford, 2012</td>
</tr>
<tr>
<td>C-SHORE</td>
<td>C</td>
<td>co-trivial</td>
<td>backward</td>
<td>RHL, TUM, LIAFIA, ’13</td>
</tr>
<tr>
<td>HorSat</td>
<td>C</td>
<td>co-trivial</td>
<td>backward</td>
<td>Tokyo, 2013</td>
</tr>
<tr>
<td>HorSatT</td>
<td>C/T</td>
<td>trivial</td>
<td>mixed</td>
<td>Tokyo, 2013</td>
</tr>
</tbody>
</table>

None of the above can scale robustly beyond HORS of a few hundred rules!
Based on refinement types, but uses abstraction refinement.

**Input:** HORS $G$, alternating trivial automaton $A = \langle \Sigma, Q, \delta, q_I \rangle$

**Output:** YES if $A$ accepts $\text{tree}(G)$; NO otherwise.

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

$$
C_0 = \langle \Gamma_0^\exists, \Gamma_0^\forall \rangle = \langle \emptyset, \emptyset \rangle \\
C_{k+1} = \langle \Gamma_{k+1}^\exists, \Gamma_{k+1}^\forall \rangle = \langle \Gamma_k^\exists \cup \text{env}_A(C_k), \Gamma_k^\forall \cup \text{env}_R(C_k) \rangle
$$

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 $(\Gamma_k^\exists, A)$.
- Verifier has a winning strategy in typing parity game induced by $(\Gamma_k^\forall, \neg A)$.

**Variant (Termination).** $(S: q_I \in \Gamma_\exists) \lor (\text{env}_R(C_k) \setminus \Gamma_\forall \neq \emptyset)$. 

[^3]: [http://mjolnir.cs.ox.ac.uk/web/preface](http://mjolnir.cs.ox.ac.uk/web/preface)
### Category 1 Benchmarks (Times in seconds.)

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

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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Rules</th>
<th>Order</th>
<th>Preface</th>
<th>HorSat</th>
<th>HorSatT</th>
<th>C-SHORE</th>
<th>GTRecS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfa-psdes</td>
<td>237</td>
<td>7</td>
<td>0.51</td>
<td>0.28</td>
<td>1.81</td>
<td>3.44</td>
<td>–</td>
</tr>
<tr>
<td>cfa-matrix-1</td>
<td>383</td>
<td>8</td>
<td>0.61</td>
<td>0.73</td>
<td>6.30</td>
<td>18.58</td>
<td>–</td>
</tr>
<tr>
<td>cfa-life2</td>
<td>898</td>
<td>14</td>
<td>1.46</td>
<td>5.94</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

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

### Category 3 Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Rules</th>
<th>Order</th>
<th>Preface</th>
<th>HorSat</th>
<th>HorSatT</th>
<th>C-SHORE</th>
<th>GTRecS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp2-1600</td>
<td>1606</td>
<td>2</td>
<td>8.39</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>10.47</td>
</tr>
<tr>
<td>exp2-3200</td>
<td>3206</td>
<td>2</td>
<td>17.51</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>59.13</td>
</tr>
<tr>
<td>exp2-6400</td>
<td>6406</td>
<td>2</td>
<td>39.58</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>exp2-12800</td>
<td>12806</td>
<td>2</td>
<td>92.19</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>exp4-400</td>
<td>408</td>
<td>4</td>
<td>14.12</td>
<td>–</td>
<td>106.53</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>exp4-800</td>
<td>808</td>
<td>4</td>
<td>30.55</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>exp4-1600</td>
<td>1608</td>
<td>4</td>
<td>71.06</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>exp4-3200</td>
<td>3208</td>
<td>4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

These order-$n$ RS generate $\exp_n$-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 $F_{\omega}$-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?
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 semantics 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.