Higher-Order Model Checking:
from Semantics to Algorithmics

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

CurienFest, 9-11 September 2013, Venice
Higher-Order Model Checking is the model checking of infinite (term-)trees generated by recursion schemes for the formal analysis of higher-order computation.

Recursion schemes = programs of simply-typed $\lambda$-calculus + recursion, generated from uninterpreted alphabet $\Sigma$ of 1st-order symbols.

**MSO Model-Checking Problem for Recursion Schemes [KNU01/02]**

- **INSTANCE:** An order-$n$ recursion scheme $G$, and an MSO formula $\varphi$
- **QUESTION:** Does the generated tree, $\text{tree}(G)$, satisfy $\varphi$?

This problem is decidable. Various proofs:

1. Game semantics [O. LICS06]
2. Collapsible PDA [Hague, Murawski, O. & Serre LICS08]
3. Intersection refinement types [Kobayashi POPL09; K. & O. LICS09]
4. Krivine machine [Salvati & Walukiewicz ICALP11]

The problem is hyper-exponential ($n$-EXPTIME complete). How practical are these model-checking algorithms?
A Type System Characterising MSO Definability

Theorem (Kobayashi + O. LiCS 2009)

Given an alternating parity tree automaton (APT) \( A \) (or, equivalently, an MSO formula), there is a type system \( \mathcal{K}_A \) such that for every recursion scheme \( G \), \( \text{tree}(G) \) is accepted by \( A \) iff \( G \) is \( \mathcal{K}_A \)-typable.

- Correctness properties given by APT or MSO formula or Mu-Calculus:

\[
\begin{align*}
\text{MSOL} & \xrightarrow{(1)} \text{Mu-Calculus} \\
\text{Parity Games} & \xleftarrow{(2)} \Rightarrow \xrightarrow{(3)} \text{APT}
\end{align*}
\]

- (1) + (3) MSOL, mu-calculus and APT are effectively equi-expressive for tree languages. [Niwinski / Emerson & Jutla FoCS 91]
- (2) Mu-calculus Model Checking Problem and PARITY are inter-reducible [Streett & Emerson 1989]
- MSO model checking reduces to type inference.
A parity tree automaton is a tuple $\mathcal{A} = \langle \Sigma, Q, \Delta, q_I, \Omega \rangle$ where
1. $Q$ is a finite set of states, and $q_I \in Q$ is the initial state
2. $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is a transition relation
3. $\Omega : Q \rightarrow \{0, 1, \cdots, p\}$ is a priority map.

A tree $t : \{1, 2\}^* \rightarrow \Sigma$ is accepted by $\mathcal{A}$ just if there is a run of the automaton (i.e. a state-annotation of the tree $t$ that respects the transition relation $\Delta$) that satisfies

**Parity:** for every infinite path $\pi$ in the run tree, the least priority that occurs infinitely often in $\pi$ is even.

**Example.** $\Sigma = \{a, b\}$. A parity tree automaton that recognises

$$\{ t \mid \text{every path through } t \text{ has only finitely many } a \text{'s} \}$$

uses $q_a, q_b$ to signal reading of $a$ and $b$, with $\Omega(a) = 1$ and $\Omega(b) = 2$. [The least infinitely occurring priority in a path is odd iff $a$ occurs infinitely often.]
Refinement types embedded with states and priorities

Fix an (alternating) parity tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$. Construct refinement types from states $q \in Q$ and priorities $m_i \in \{0, \cdots, p\}$.

Refinement type
\[ \theta ::= q \mid \tau \rightarrow \theta \]

Intersection
\[ \tau ::= \bigwedge_{i=1}^{l}(\theta_i, m_i) \quad l \geq 0 \]

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 refinement 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 [Kobayashi POPL09]

- $q$ 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 $q$

E.g. a tree function described by $(q_1, m_1) \land (q_2, m_2) \rightarrow q$. 
Typing judgments \( \Gamma \vdash t : \theta \)

where environment \( \Gamma \) is a finite set of bindings of the form \( F : (\theta, m) \) or \( x : (\theta, m) \), where \( F \) ranges over function symbols of the scheme \( G \).

Typing System \( \mathcal{K}_A \): Validity is defined by induction over four rules.

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

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

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

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

Note: multiplicative flavour; no weakening.
A Typing Parity Game: Assume HORS $G \& APT \mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$

**DEF.** We say that $G$ is typable just if Verifier has a winning strategy in (finite) parity game $G(\mathcal{K}_A)$.

**Idea:** Verifier asserts (valid) typing judgements $\Gamma \vdash s : \theta$; Refuter challenges the assumptions (i.e. type bindings) in $\Gamma$.

- **Start** position: $S : (q_I, \Omega(q_I))$.
- 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 refinement type $\theta$.

Verifier **wins** just if every infinite play satisfies parity condition.

**Typability is decidable** because finite parity games are solvable (given $G$ and $\mathcal{A}$, there are only finitely many refinement types).
Theorem (Reduction)

Given an APT \( \mathcal{A} \) there is a type system \( \mathcal{K}_\mathcal{A} \) such that for every HORS \( G \), \( \text{tree}(G) \) 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 \( \mathcal{K}_\mathcal{A} \).

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

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

where

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

\(^1\) \( \exp_0(x) = x; \exp_{k+1}(x) := 2^{\exp_k(x)} \).
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, control flow analysis and strictness 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>$o \rightarrow o$</td>
<td>$2^2 \times 2 = 8$</td>
</tr>
<tr>
<td>2</td>
<td>$(o \rightarrow o) \rightarrow o$</td>
<td>$2^8 \times 2 = 512$</td>
</tr>
<tr>
<td>3</td>
<td>$((o \rightarrow o) \rightarrow o) \rightarrow o$</td>
<td>$2^{513} \approx 10^{154} &gt;&gt; #$ atoms in universe!</td>
</tr>
</tbody>
</table>

Note: $\rho(\kappa_1 \rightarrow \kappa_2) := 2^{\rho(\kappa_1)} \times \rho(\kappa_2)$; $\rho(o) = 2$

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

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

---

On realistic examples, algorithm terminates 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]
(K) Krivine machine [Salvati & Walukiewicz ICALP11]

G, C and T are basis of attempts to build practical algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Basis</th>
<th>Properties</th>
<th>Propagation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRecS</td>
<td>T</td>
<td>trivial automata</td>
<td>forward</td>
<td>Tohoku, 2009</td>
</tr>
<tr>
<td>GTRRecS1, 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>LIAFA/RHL/TUM ’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. 

**Idea:** Converge on the solution from both sides, by reasoning about acceptance by \( \mathcal{A} \) and acceptance by \( \neg \mathcal{A} \) *simultaneously*.

**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 environment pairs

\[
\langle \Gamma^0_\exists, \Gamma^0_\forall \rangle, \langle \Gamma^1_\exists, \Gamma^1_\forall \rangle, \langle \Gamma^2_\exists, \Gamma^2_\forall \rangle, \cdots
\]

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 \( (\Gamma^k_\exists, \mathcal{A}) \).
- Verifier has a winning strategy in typing parity game \( (\Gamma^k_\forall, \neg \mathcal{A}) \).
## Category 1 Benchmarks from MoCHi (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 of F# 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 out of time (set to 2 mins) or other resources.

**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?
Further Directions

1. **Composing Parity Games**
   Model checking is typically a whole program analysis: in higher-order computation, model checking only addresses terms of ground type, and not arbitrary higher types.

   **Goal**: Build a cartesian closed category of parity games, by analysing [Kobayashi & O LICS09].

2. **HOMC Algorithm Design**
   Redesign the Preface algorithm so that type extraction is uniformly more aggressive in each iteration.
   Extend the algorithm to model check HORS with respect to APT.
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 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.
- Broadbent & O: On global model checking trees generated by higher-order recursion schemes. FoSSaCS 2009.
- Kobayashi & O: A type theory equivalent to the model checking of higher-order recursion schemes. LiCS 2009.
- O & Tsukada: Two-Level Game Semantics, Intersection Types, and Recursion Schemes. ICALP, 2012
- Neatherway, Ramsay & O: A traversal-based algorithm for higher-order model checking. ICFP, 2012
- O & Ramsay: Verifying higher-order functional programs with pattern-matching algebraic data types. POPL 2011
- Broadbent, Carayol, O & Serre: Recursion Schemes and Logical Reflection. LICS 2010