Game Semantics and its Algorithmic Applications

Lecture 4: On Model-Checking Trees generated by Higher-Order Recursion Schemes

Luke Ong
Oxford University Computing Laboratory

www.comlab.ox.ac.uk/oucl/work/luke.ong/
A quick recap

**A Problem in Verification.** Find classes of finitely-presentable infinite structures that have decidable MSO theories.

**Study:** Hierarchies of structures ($\Sigma$-labelled trees - for this talk) generated by higher-order recursion schemes. E.g. for each $n \geq 0$

\[
\text{SafeRecSch}_n = \begin{cases} 
\Sigma\text{-labelled trees generated by} \\
\text{order-}n \text{ safe recursion schemes}
\end{cases}
\]

**Why?**

- Rich and unifying class - subsumes several major results
- Robust framework - admits different characterizations.

**Theorem** (Knapik-Niwiński-Urzyczyn-Caucal 2002). For each $n \geq 0$,

\[
\text{SafeRecSch}_n = \text{PushdownTree}_n = \text{CaucalTree}_n.
\]
Order-\(n\) (Deterministic) Recursion Scheme \(G = (\mathcal{N}, \Sigma, \mathcal{R}, S')\)

Fix a set \(\text{Var}\) of typed variables.

- \(\mathcal{N}\): Typed non-terminals of order at most \(n\), \(D : A_1 \to \cdots \to A_m \to o\), including a distinguished start symbol \(S : o\).

- \(\Sigma\): Ranked alphabet of terminals: Each \(f \in \Sigma\) has arity \(\text{ar}(f) \geq 0\), with \(f : o \to \cdots \to o \to o\) (written \(o^{\text{ar}(f)} \to o\))

- \(\mathcal{R}\): An equation for each non-terminal \(D : A_1 \to \cdots \to A_m \to o\) of the shape

\[
D \varphi_1 \cdots \varphi_m = e
\]

where the applicative term \(e : o\) is constructed from

- terminals \(f, g, a, \text{etc.}\) from \(\Sigma\)

- variables \(\varphi_1 : A_1, \cdots, \varphi_m : A_m\) from \(\text{Var}\),

- non-terminals \(D, F, G, \text{etc.}\) from \(\mathcal{N}\)
E.g. (order-2). $B : (o \to o) \to (o \to o) \to o \to o$, $F : (o \to o) \to o$

$$G_2 : \begin{cases} S & = Fg \\ B \varphi \psi x & = \varphi (\psi x) \\ F \varphi & = f (\varphi a) (F (B \varphi \varphi)) \end{cases}$$

The value tree, $\left[ G_2 \right] : \{ 1, 2 \}^* \to \Sigma$, is:

$$\begin{align*}
\epsilon & \mapsto f \\
1 & \mapsto g \\
2 & \mapsto f \\
\ldots & \ldots
\end{align*}$$
Background

**Theorem** (Knapik, Niwiński + Urzyczyn FOSSACS02). The MSO model checking problem for trees generated by order-\(n\) safe recursion schemes is decidable, for each \(n \geq 0\).

Recall: Safety is a rather awkward syntactic condition.

**Question (KNU02).** Is the safety constraint necessary for MSO decidability?

Some partial answers:

1. Aehlig, de Miranda + Ong TLCA05: MSO model checking problem is decidable for arbitrary order-2 scheme (i.e. whether safe or not, and whether homogeneously typed or not).

2. Knapik, Niwiński, Urzyczyn, Walukiewicz ICALP05: Modal mu-calculus model checking problem for homogeneously-typed order-2 schemes (whether safe or not) is 2-EXPTIME complete.
Definition: Homogeneously-Typed and Safe Recursion Scheme

- $o$ is homogeneous; and $(A_1 \to \cdots \to A_n \to o)$ is homogeneous just if $\text{order}(A_1) \geq \text{order}(A_2) \geq \cdots \geq \text{order}(A_n)$, and each $A_i$ is homogeneous.

- “Formally a homogeneously-typed term of order $k > 0$ is said to be unsafe if it contains an occurrence of a parameter of order strictly less than $k$, otherwise the term is safe. An occurrence of an unsafe term $t$ as a subexpression of a term $t'$ is safe if it is in the context $\cdots (ts) \cdots$, otherwise the occurrence is unsafe. A recursion scheme is safe if no unsafe term has an unsafe occurrence in the RHS of any equation.”

Knapik et al. FOSSACS02

Example. $\Sigma$-symbol $f$ has arity 2.

(i) $G x^o = H (f x)$

(ii) $F \phi^{(o,o)} x^o y^o = f (F (F \phi y) y (\phi x)) x$
Theorem. “Safety is not necessary for MSO decidability.”

The modal mu-calculus model checking problem for trees generated by arbitrary order-\(n\) recursion schemes is \(n\)-EXPTIME complete, for each \(n \geq 0\).

Three major ingredients:

1. A transference principle from tree generated from the recursion scheme – value tree – to an auxiliary computation tree.
   A strong correspondence between paths in the value tree and traversals in the computation tree – proof is by game semantics.
   Thus an alternating parity tree automaton (APT) has accepting run-tree over value tree iff it has accepting traversal-tree over computation tree.

2. Simulation of (accepting) traversal-tree by a certain set of annotated paths over computation tree; the latter recognised by traversal-simulating APT.

3. Application of Jurdziński’s complexity bound for solving acceptance parity game to a finite graph (which generates the computation tree) and the traversal-simulating APT.
**Fact.** (i) Over trees, MSO and modal mu-calculus are equi-expressive.
(ii) Modal mu-calculus and alternating parity tree automata are equivalent.

Positive boolean formulas over $P$: $p$ ranges over $P$

$$B^+(P) \ni \theta ::= \text{true} \mid \text{false} \mid p \mid \theta \land \theta \mid \theta \lor \theta$$

For $S \subseteq P$ and $\theta \in B^+(P)$, we say $S$ satisfies $\theta$ if assigning true to elements in $S$ and false to elements in $P \setminus S$ makes $\theta$ true.

**Alternating Parity Tree Automaton (APT)** \( \mathcal{B} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle \)

- $\Sigma$ is the input ranked alphabet
- $Q$ is a finite set of states, and $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \longrightarrow B^+(\lfloor ar(\Sigma) \rfloor \times Q)$ is the transition function where, for each $f \in \Sigma$ and $q \in Q$, we have $\delta(q, f) \in B^+(\lfloor ar(f) \rfloor \times Q)$
  
  *Notation:* $[m] = \{1, \cdots, m\}$; $ar(\Sigma) = \max\{ ar(f) : f \in \Sigma \}$.
- $\Omega : Q \longrightarrow \mathbb{N}$ is the priority function.
When does an APT $B$ accept a $\Sigma$-labelled tree $t : \text{dom}(t) \rightarrow \Sigma$?

An APT $B$ accepts a $\Sigma$-labelled tree $t$ just if it has an accepting run-tree over $t$ i.e.

“There is a certain set of ‘$\delta_B$-respecting’, state-annotated maximal paths in $t$ such that the infinite paths satisfy the parity condition.”

$\delta_B$-respecting:

Automaton reads root $\epsilon$ with initial state $q_0$.

Suppose automaton reads node $\alpha$ of $\text{dom}(t)$ with state $q$.

- Guess a set $S \subseteq [\text{ar}(t(\alpha))] \times Q$ that satisfies the positive boolean formula $\delta_B(q, t(\alpha))$.
  
  Recall: $\delta_B : Q \times \Sigma \rightarrow \mathbb{B}^+([\text{ar}(\Sigma)] \times Q)$.

- For each $(i, q') \in S$, spawn automaton to read $i$-child of $\alpha$ with state $q'$.

**Parity condition**: largest priority that occurs infinitely often is even.
Outline of the Lecture

1. **Transfer Principle**: Correspondence between paths in value tree and traversals over computation tree

2. **Simulation** of traversals by annotated paths over the computation tree

3. **Complexity Analysis**: Succinctness property and Jurdziński’s bound

4. Further directions
Transference Principle: from value tree to computation tree

Direct algorithmic analysis of value tree $[G]$ (i.e. tree generated by the given recursion scheme) is futile:

- Value tree has no useful structure for our purpose.
- It is the “extensional” outcome of a (potentially infinite) computational process, the algorithmics of which are what we should analyse.

Our approach:

We consider an auxiliary computation tree $\lambda(G)$ which recovers useful intensional information about the computational process behind the construction of the value tree.

Indeed it is static view of the very computational process.

Roughly speaking, “evaluating the computation tree (by contracting the $\beta$-redexes) returns the value tree”.

The Long Transform: from $G$ to $\overline{G}$

$\overline{G}$-rules are obtained by: For each $G$-rule

1. Expand RHS to its $\eta$-long form, including ground-type subterm in operand position. Thus $e : o \eta$-expands to $\lambda.e$ (“dummy lambdas”).

2. Insert long-apply symbol @: Replace every ground-type subterm $D e_1 \cdots e_n$ by $@ D e_1 \cdots e_n$, where $D$ ranges over non-terminals.

3. Curry each equation.

4. Rename (bound) variables afresh. Only finitely many new names.

Example.

\[
G : \begin{cases} 
S &= F H \\
F \phi &= \phi (F \phi) \\
H z &= f z z 
\end{cases} \quad \Rightarrow \quad \overline{G} : \begin{cases} 
S &= @ F (\lambda x. @ H \lambda x) \\
F &= \lambda \phi. \phi (\lambda. @ F (\lambda y. \phi (\lambda. y)))) \\
H &= \lambda z. f (\lambda. z)(\lambda. z) 
\end{cases}
\]
Computation tree $\lambda(G)$

Long transform $\overline{G}$ is an order-0 recursion scheme:

- Terminals are those of $G$ plus variables, lambdas (of form $\lambda x$) and long-apply symbols @.
- Non-terminals are those of $G$ but all have a new type $o$.
- No variables (=: currying) - hence order-0!

The **computation tree**, written $\lambda(G)$, is defined to be $[\overline{G}]$ – the infinite term-tree obtained by unfolding $\overline{G}$-rules *ad infinitum*

Computation trees are **regular** trees - very useful!

Labels in $\lambda(G)$ from a **finite** set – **no renaming** of bound variables.

Nodes at levels 0, 2, 4, etc. are labelled by non-lambdas (= P-moves); nodes at levels 1, 3, 5, etc. are labelled by lambdas (= O-moves).

[Note: Computation trees do not correspond to arenas!]
\[
G : \begin{cases}
S &= F H \\
F \varphi &= \varphi (F \varphi) \\
H z &= f z z
\end{cases}
\quad \mapsto \quad \overline{G} : \begin{cases}
S &= \@ F (\lambda x. @ H \lambda . x) \\
F &= \lambda \varphi. \varphi (\lambda . @ F (\lambda y. \varphi (\lambda . y)))) \\
H &= \lambda z. f (\lambda . z) (\lambda . z)
\end{cases}
\]

The **computation tree** \(\lambda(G)\) is the term-tree obtained by infinitely unfolding \(\overline{G}\):

\[
\lambda(G')
\]

---

*Game Semantics and its Algorithmic Applications: Lecture 4, Newton Institute, February 2006.*
Definition of traversals over $\lambda(G)$

Recall: $\lambda(G)$ is tree generated by order-0 recursion scheme $\langle \Lambda_G, N, S, \mathcal{R} \rangle$

Define a family of binary relations $\vdash_i$, where $i \in [ar(\Lambda_G)]$, between nodes of a computation tree $\lambda(G)$, called **enabling**:

- A lambda node $l$ is $i$-enabled by its parent node $m$ where $l$ is the $i$-child of $m$; we write $m \vdash_i l$

- A variable node $\xi_i$ is $i$-enabled by its **binder**, which is defined to be the node, labelled $\lambda\xi_1 \cdots \xi_n$ where $1 \leq i \leq n$, that is closest to $\xi_i$ enroute to the root. We write $\lambda\xi \vdash_i \xi_i$.

A **justified sequence** over $\lambda(G)$ is a possibly infinite, lambda / non-lambda alternating sequence of nodes satisfying the **pointer condition**:

Each node $n$ that occurs in it, except those labelled by @ or a $\Sigma$-symbol, has a pointer to some earlier node-occurrence $n_0$ in the sequence such that $n_0 \vdash_j n$, for some $j$. 
**Definition.** *Traversals* over $\lambda(G)$ are justified sequences defined by induction:

**(Root)** The singleton sequence (comprising $\epsilon$) is a traversal.

**(App)** If $t \ @$ is a traversal, so is $t \ @ \ \lambda \xi$.

**(Sig)** If $t f$ is a traversal, so is $t f \lambda$ where $1 \leq i \leq \text{arity}(f)$.

**(Var)** If $t n \lambda \xi \cdots \xi$ is a traversal, so is $t n \lambda \xi \cdots \xi \lambda \eta$.

**(Lam)** If $t \lambda \xi$ is a traversal, so is $t \lambda \xi \ n$, such that $\gamma t \lambda \xi \ n \gamma$ is a path in $\lambda(G)$.

**Key lemma:**

(i) Traversals are justified sequences that satisfy Visibility.

(ii) P-views of traversals are paths in the computation tree.
**Theorem.** (Correspondence) Let $G$ be an order-$n$ recursion scheme.

(i) There is a 1-1 correspondence between maximal paths $p$ in ($\Sigma$-labelled) value tree $\llbracket G \rrbracket$ and maximal traversals $t_p$ over computation tree $\lambda(G)$.

(ii) Further for each $p$, we have $p \upharpoonright \Sigma = t_p \upharpoonright \Sigma$.

Proof is by game semantics.

**Idea:** Traversals over computation tree are a concrete representation of the dynamics of the computational process.

Value tree $\llbracket G \rrbracket$ is a representation of the strategy-denotation of $G$ (in game semantics of PCF).

Paths in $\llbracket G \rrbracket$ corresponds to plays in the strategy-denotation.

Traversals $t_p$ over computation tree $\lambda(G)$ are just (representations of) the uncoverings of the plays (= path) $p$ in the strategy-denotation of $G$. 
Strategy composition is “parallel composition of two processes $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ synchronizing on $B$, followed by hiding of $B$-moves.”

The play in $\sigma ; \tau$ thus constructed is $c_1 a_1 a_2 c_2 c_3 a_3$. 

---

*Game Semantics and its Algorithmic Applications: Lecture 4, Newton Institute, February 2006.*
From run-tree over $[G]$ to traversal-tree over $\lambda(G)$

Thus: Property APT $B$ has an accepting run-tree over $[G]$ by def.

$\iff$

$\exists$ certain set of $\delta_B$-respecting, state-annotated maximal paths in $[G]$ satisfying parity condition

Thm (Corr)

$\iff$

$\exists$ certain set of $\delta_B$-respecting, state-annotated maximal traversals over $\lambda(G)$ satisfying parity condition

new def.

$\iff$

Property APT $B$ has an accepting traversal-tree over $\lambda(G)$.

Higher-order traversals can be very complex - they jump all over the tree, and can visit certain nodes infinitely often.

See order-3 example!

**Problem**: Find a device to recognise an accepting traversal-tree.
An order-3 example:
Outline of the Lecture

1. **Transfer Principle**: Correspondence between paths in value tree and traversals over computation tree

2. **Simulation of traversals by annotated paths over the computation tree**

3. **Complexity Analysis**: Succinctness property and Jurdziński’s bound

4. Further directions
Simulate traversals by *paths* – an order-2 illustration

**Idea.** Simulate an annotated traversal by the respective P-views of all its prefixes, which are a set of annotated paths in the computation tree.

Suppose a traversal jumps from $\varphi$ at simulating state $q_1$ to a sibling subtree rooted at $\lambda y_1 y_2$, subsequently exits it at $y_1$ and rejoins the original subtree at first $\lambda$-child of $\varphi$ with state $q_2$.

Simulate the traversal by *paths*:

- At $\varphi$ with $q_1$, **guess** that the detour will return at first $\lambda$-child with state $q_2$.
- **Spawn** an automaton at $\lambda y_1 y_2$ to **verify the guess.**
Formalising the guesses as Variable Profiles $\mathbf{VP}_G^B(A)$

Fix a recursion scheme $G$, and a property APT $B = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ with $p$ priorities. Write $[p] = \{1, \ldots, p\}$.

$$\mathbf{VP}^B_G(o) = \varnothing_G^o \times Q \times [p] \times 2^\varnothing$$

$$\mathbf{VP}^B_G(A_1 \rightarrow \cdots \rightarrow A_n \rightarrow o) = \varnothing_G^A \times Q \times [p] \times 2^\left(\bigcup_{i=1}^n \mathbf{VP}^B_G(A_i)\right)$$

Asserting $(\varphi, q, m, c) \in \mathbf{VP}^B_G(A)$ at node $\alpha$ of computation tree means: the traversal being simulated will reach some descendant-node that is labelled $\varphi$

(i) with state $q$, such that

(ii) $m$ is the highest priority that will have been encountered up to that point

(iii) further, the traversal (which will then jump to the root of a subtree that denotes the actual argument of $\varphi$) will eventually return to the children of the node labelled $\varphi$ “in accord with $c$”.
Traversing-simulating APT $\mathcal{C}$: simulate $\mathcal{B}$-states + verify guesses

$\mathcal{C}$-states: $q \rho$ where $q$ is $\mathcal{B}$-state being simulated, and environment $\rho$ is the set of profiles of variable (within current scope) to be verified.

**Suppose automaton with state $q \rho$ reading node with label $l$:** Some cases (verification of priorities omitted)

- $l$ is a $\Sigma$-symbol $f : o^k \rightarrow o$.

  Guess a set $\{ (i_1, q_1), \ldots, (i_l, q_l) \}$ satisfying $\delta_B(q, f)$ (abort, if impossible), and guess environments $\rho_1, \ldots, \rho_l$ such that $\bigcup_{j=1}^{l} \rho_j = \rho$.

  For each $j$, spawn automata with state $q_j \rho_j$ in direction $i_j$.

- $l$ is an $@$ with children labelled by $\lambda \varphi$ and $\lambda \eta_1, \ldots, \lambda \eta_k$.

  Guess $\rho' = \{ (\varphi_{i_j}, q_j, m_j, c_j) : 1 \leq j \leq l \}$, and spawn automaton with state $q \rho'$ in direction 0. Guess $\rho_1, \ldots, \rho_l$ with $\bigcup_{j=1}^{l} \rho_j = \rho$. For each $j$, spawn automaton with state $q_j (\rho_j \cup c_j)$ in direction $i_j$.

**Question.** Simulating traversal-tree by 2-way APT?
Theorem (Simulation). The following are equivalent:

(i) Property APT $B$ has an accepting traversal-tree over the computation tree $\lambda(G)$.

(ii) Traversal-simulating APT $C$ has an accepting run-tree over the computation tree $\lambda(G)$.

“(i) ⇒ (ii)”: From the traversal-tree annotated only by $B$-states, we perform a succession of annotation operations, transforming it to a traversal-tree annotated by $C$-states.

The set of P-views of all such $C$-state-annotated traversals is precisely an accepting run-tree of $C$.

“(ii) ⇒ (i)”: Reconstruct each traversal (of the putative traversal-tree) as a sequence of segments of paths (=P-views) in the accepting run-tree, thus inheriting an accepting state-annotation.

Satisfication of parity condition tricky!
Key Steps of the Decidability Proof

Let $G$ be any order-$n$ recursion scheme, and $\varphi$ a modal mu-calculus formula.

Value tree $\llbracket G \rrbracket$ satisfies $\varphi$

$\iff \{\text{Emerson + Jutla 1991}\}$

Property APT $B$ has accepting run-tree over $\llbracket G \rrbracket$

$\iff \{\text{Correspondence Theorem}\}$

$B$ has an accepting traversal-tree over computation tree $\lambda(G)$

$\iff \{\text{Simulation Theorem}\}$

Traversal-simulating APT $C$ has an accepting run-tree over $\lambda(G)$

$\iff \{\text{Stirling(?), etc.}\}$

Eloise has winning strategy in acceptance parity game $G(\text{Gr}(G), C)$

for finite graph $\text{Gr}(G)$ with $\text{unfold}(\text{Gr}(G)) = \lambda(G)$
Outline of the Lecture

1. **Transfer Principle**: Correspondence between paths in value tree and traversals over computation tree

2. **Simulation** of traversals by annotated paths over the computation tree

3. **Complexity Analysis**: Succinctness property and Jurdziński’s bound

4. Further directions
Complexity of Modal Mu-Calculus Model Checking

Mu-calculus model checking of order-$n$ trees is $n$-EXPTIME hard, because it is already so for safe trees (T. Cachat ICALP’04).

Use parity game to show problem is decidable in $n$-EXPTIME.

**Theorem.** (Jurdzinski 2000) Eloise’s winning regions and strategy in a parity game with $|V|$ vertices and $|E|$ edges and $p \geq 2$ priorities is computed in time

\[
O \left( p \cdot |E| \cdot \left( \frac{|V|}{\lfloor p/2 \rfloor} \right)^{\lfloor p/2 \rfloor} \right)
\]

Fix an order-$n$ recursion scheme $G$, a property APT $B = \langle Q, \Sigma, \delta, q_0, \Omega \rangle$ with $p$ priorities and its corresponding traversal-simulating APT $C = \langle Q_C, \Lambda_G, \delta_C, q_0 \emptyset, \Omega_C \rangle$.

Define a parity game $G(G, C)$ such that Eloise has a winning strategy iff the APT $C$ accepts $\lambda(G)$ (iff $B$ accepts $\llbracket G \rrbracket$).
\[
\mathbf{VP}_G^B = \bigcup_{i=0}^{n-1} \mathbf{VP}_G^B(i)
\]
\[
\text{Env}_G = \mathcal{P}(\mathbf{VP}_G^B)
\]
\[
|\mathbf{VP}_G^B(i)| = \exp_i O(|G| \cdot |Q| \cdot p)
\]

By a succinctness property of traversal-simulating APT \( C \): If \( C \) has an accepting run-tree, then it has one with small branching factor.

\[
|V| = O(|G| \cdot |Q_C| \cdot (|G| \cdot |Q_C|)^{|\mathbf{VP}_G^B|}) = \exp_n O(|G| \cdot |Q| \cdot p)
\]

Since \(|E|\) is at most \(|V|^2\), time complexity is

\[
O \left( p \cdot (|V|)^{\lceil p/2 \rceil + 2} \right) = O \left( p \cdot (\exp_n O(|G| \cdot |Q| \cdot p))^{\lceil p/2 \rceil + 2} \right)
\]
\[= \exp_n O(|G| \cdot |Q| \cdot p)
\]

**Theorem.** The modal mu-calculus model checking problem for trees generated by order-\( n \) recursion schemes (whether safe or not) is \( n \)-EXPTIME complete, for all \( n \geq 0 \).

Hence MSO theories of these trees are decidable (non-elementarily).
Further directions

1. Is safety a genuine constraint on expressiveness?

   **Conjecture.** SafeRecSch$_2$ ⊂ RecSch$_2$ I.e. There are inherently unsafe trees (at order 2).
   Candidate: Urzyczyn’s tree.


3. What is the corresponding hierarchy of graphs generated by high-order recursion schemes? Are their MSO theories decidable?

4. “Mixing semantic and verification games”: Denotational semantics of λ-calculus “relative to an alternating parity tree automaton (APT)”.

   **Problem.** Construct a cartesian closed category (= model of the lambda calculus), parameterized by an APT, whose maps are witnessed by profiles (“guesses”).