Infinite Trees, Higher-Order Recursion Schemes and Game Semantics

Luke Ong

Oxford University Computing Laboratory

www.comlab.ox.ac.uk/oucl/work/luke.ong/
Abstract

Higher-order recursion schemes are a natural (and old) model of programs. They define a family of finitely branching infinite term-trees, which forms an infinite hierarchy according to their type-theoretic level.

Building on the famous work of Rabin 1969 and others, Knapik et al. (FOSSACS 2002) proved that the MSO theories of all such trees are decidable, provided the generating recursion scheme satisfies a syntactic constraint called safety. Is the safety assumption really necessary?

We prove that

(i) The modal mu-calculus model-checking problem for trees generated by level-$n$ recursion schemes is $n$-EXPTIME complete, for all $n \geq 0$.

(ii) Hence trees generated by recursion schemes of every level, whether safe or not, have decidable MSO theories.

In this talk, we survey the area, explain the result, and briefly sketch a game-semantic proof.
Outline of Talk

1. Level-$\eta$ Recursion Schemes and their Value Trees
2. A Model-Checking Problem
4. The Theorem and Proof Outline
Level-n Recursion Scheme $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$

Fix a set $\text{Var}$ of simply-typed variables.

- $\mathcal{N}$: Simply-typed non-terminals of level (= order) at most $n$
  \[ D : A_1 \rightarrow \cdots \rightarrow A_m \rightarrow o \]
  including a distinguished start symbol $S : o$.

- $\Sigma$: Terminals $f : o \rightarrow \cdots \rightarrow o \rightarrow o$ (written $o^k \rightarrow o$) with $k \geq 0$

- $\mathcal{R}$: Equations for non-terminals $D : A_1 \rightarrow \cdots \rightarrow A_m \rightarrow o$ of the shape
  \[ D \varphi_1 \cdots \varphi_m = e \]
  where the applicative term $e : o$ is constructed from
  - terminals $f, g, a$, etc. from $\Sigma$
  - variables $\varphi_1 : A_1, \cdots, \varphi_m : A_m$ from $\text{Var}$,
  - non-terminals $D, F, G$, etc. from $\mathcal{N} - \{ S \}$
Examples

Set $\Sigma = \{ f, f^\prime : o^2 \to o, \ g : o \to o, \ a : o \}$.

1. A level-0 example: No variables.

$$G_1 : \begin{cases} S &= f \, T \, T \\ T &= f^\prime \, U \, U \\ U &= f \, T \, T \end{cases}$$

2. A level-2 example.

$$B : (o \to o) \to (o \to o) \to o \to o, \quad F : (o \to o) \to o$$

$$G_2 : \begin{cases} S &= F \, g \\ B \, \varphi \psi \, x &= \varphi (\psi \, x) \\ F \, \varphi &= f (\varphi \, a) \left(F \left(B \, \varphi \, \varphi \right)\right) \end{cases}$$
The value tree $[G]$ of a (deterministic) recursion scheme $G$ is a possibly infinite applicative term constructed from the terminals, which is obtained by unfolding the equations ad infinitum, replacing formal by actual parameters each time, starting from $S$.

Example. $\Sigma = \{ f, g, a \}$. Take

$$G_1 : \begin{cases} S &= F \, a \\ F \, x &= f \, x \,(F \,(g \, x)) \end{cases}$$

We have $[G_1] = f \, a \, (f \,(g \, a) \,(f \,(g \,(g \, a)))\,(\cdots))$.

We view the infinite term $[G]$ as a $\Sigma$-tree (generated by $G$).

Formally a $\Sigma$-tree is a function $t : T \longrightarrow \Sigma$ such that $T \subseteq \{1, \cdots, m\}^*$ is prefix-closed, and for all occurrences $\alpha \in T$, the symbol $t(\alpha) \in \Sigma$ has arity $k$ iff $\alpha$ has $k$ children, which must be $\alpha \, 1, \cdots, \alpha \, k \in T$. 
A level-2 example.

\[ \Sigma = \{ f, g, a \} \]. \( B : (o \to o) \to (o \to o) \to o \to o, \quad F : (o \to o) \to o \)

\[
G_2 : \left\{ \begin{array}{l}
S = Fg \\
B \phi \psi x = \phi(\psi x) \\
F \phi = f(\phi a)(F(B \phi \phi))
\end{array} \right.
\]

The value tree, \([ G_2 ] : \{ 1, 2 \}^* \longrightarrow \Sigma\), is:

\[
\epsilon \mapsto f \\
1 \mapsto g \\
2 \mapsto f \\
\ldots \\
11 \mapsto a \\
21 \mapsto g \\
22 \mapsto f \\
\ldots \\
\ldots
\]
Outline of Talk

1. Level-$n$ Recursion Schemes and their Value Trees

2. A Model-Checking Problem


4. The Theorem and Proof Outline
Model Checking Problem

Parametrized over logical language $\mathcal{L}$ and level $n$.

**MODEL CHECKING PROBLEM ($\mathcal{L}$, level-$n$ $\Sigma$-trees)**

\[
\begin{align*}
\text{INSTANCE:} & \quad \text{A level-$n$ recursion scheme } G, \text{ and a formula } \varphi \in \mathcal{L} \\
\text{QUESTION:} & \quad \text{Does the } \Sigma\text{-tree } [G] \text{ satisfy } \varphi?
\end{align*}
\]

We consider $\mathcal{L} =$

- Monadic Second-Order (MSO) Logic, and
- Modal mu-calculus.

A fundamental direction in Verification:

Find classes of finitely-presentable infinite structures (e.g. trees, graphs, etc.) whose MSO model-checking problem is decidable.
Monadic Second-Order Logic (for $\Sigma$-trees $t : T \rightarrow \Sigma$)

First-order variables: $x, y, z$, etc. (ranging over nodes, which are finite words over $\{1, \cdots, m\}$, for a fixed $m$)

Second-order variables: $X, Y, Z$, etc. (ranging over sets of nodes i.e. monadic relations)

MSO formulas are built up from atomic formulas:

1. Parent-child relationship between nodes: $d_i(x, y) \equiv \text{“}y \text{ is } i\text{-child of } x\text{”} $
2. Node labelling: $p_f(x) \equiv \text{“}x \text{ has label } f\text{” }$ where $f$ is a $\Sigma$-symbol
3. Set-membership: $x \in X$

and closed under

- boolean connectives: $\neg, \lor, \land, \rightarrow$
- first-order quantifications: $\forall x. -, \exists x. -$
- second-order quantifications: $\forall X. -, \exists X. -$. 

Why MSO Logic?

It is a kind of gold standard!

**MSO is very expressive.** Over graphs, MSO is strictly more expressive than modal mu-calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL*, etc.) can embed.

Over trees, modal mu-calculus is as expressive as (but algorithmically more tractable than) **MSO**: For every MSO ϕ, there is a modal mu-calculus formula p_ϕ s.t. for every Σ-tree t, we have t ⊩ ϕ ⇐⇒ t, ε ⊩ p_ϕ.

Any obvious extension of MSO would break decidability. Either of the following would permit an encoding of a Turing machine:

- Second-order quantification over binary relations.
- Freely interpretable binary relations in the vocabulary.

E.g. \( T_a(i, t) = \text{“}i\text{-th cell of the semi-infinite tape contains } a \in \Sigma \text{ at time } t\text{”} \).
Examples of MSO-definable properties of trees

Several useful relations are definable:

1. **Set inclusion** (and hence equality): $X \subseteq Y \equiv \forall x . x \in X \rightarrow x \in Y$.

2. **“Is-an-ancestor-of” or prefix ordering** $x \leq y$ (and hence $x = y$):
   \[
   \text{PrefCl}(X) \equiv \forall xy . y \in X \land \bigvee_{i=1}^{m} d_i(x, y) \rightarrow x \in X
   \]
   
   $x \leq y \equiv \forall X . \text{PrefCl}(X) \land y \in X \rightarrow x \in X$

**Examples:**

- **Reachability property**: “$X$ is a path”
- **“$X$ is a cut”** i.e. no two nodes in it are $\leq$-compatible, and it has a non-empty intersection with every maximal path; “$X$ is finite”.
- **Recurrence condition**: “There are infinitely many occurrences of the symbol $f : o \rightarrow o$.”

**But “MSO cannot count”**: E.g. “$X$ has twice as many elements as $Y$”.
Outline of Talk

1. Level-\(n\) Recursion Schemes and their Value Trees
2. A Model-Checking Problem
4. The Theorem and Proof Outline
Structures with decidable MSO theories: some milestones

1. **Rabin 1969**: Regular trees. “Mother of all decidability results”

2. **Muller and Schupp 1985**: Configuration graphs of pushdown automata.


4. **Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002)**: \(\Sigma\)-trees generated by *safe* recursion schemes of all finite levels.

5. **Caucal (MFCS 2002)**. Hierarchies of trees \((\mathcal{T}_n)_{n \in \omega}\) and graphs \((\mathcal{G}_n)_{n \in \omega}\):
   - \(\mathcal{G}_0\) are the finite graphs; \(\mathcal{T}_0\) are the finite trees.
   - Trees in \(\mathcal{T}_{n+1}\) are the *unfoldings* of graphs in \(\mathcal{G}_n\) (= KNU safe trees)
   - Graphs in \(\mathcal{G}_n\) are the *inverse rational images* of trees in \(\mathcal{T}_n\).

**Question.** Do \(\Sigma\)-trees generated by *unsafe* recursion schemes have decidable MSO theories? If so, at which levels?
Trees generated by *Safe* recursion schemes

<table>
<thead>
<tr>
<th>Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
</tr>
<tr>
<td>Level 1</td>
</tr>
</tbody>
</table>

(All level-0 and level-1 trees are safe.)

**Safety seems a robust definition: several characterisations**

1. Hierarchy of *higher-order pushdown trees* generated by higher-order pushdown automata (KNU 2002)
   E.g. A level-2 stack is a stack of level-1 stacks.

2. **Caucal Tree Hierarchy** 2002: generated from finite trees by iterated transformations that preserve MSO decidability.

3. **Indexed grammars** of level $n + 1$ have exponents / indices that are grammars of level $n$. *(Maslov 1976)*
What is the safety constraint?


**Definition** [KNU02]. A level-2 equation is **unsafe** if the RHS has a subterm $P$ such that

(i) $P$ is level 1

(ii) $P$ occurs in an operand position (i.e. as 2nd argument of the application operator)

(iii) $P$ contains a level-0 parameter.

**Examples of unsafe equations**: $f : o^2 \rightarrow o \ G, H : o$.

$$
G \ x \ = \ H \ (f \ x) \\
F \ \varphi \ x \ y \ = \ f \ (F \ (F \ \varphi \ y) \ y \ (\varphi \ x)) \ a
$$

Safety (as presented above) seems syntactically awkward and semantically unnatural, but has important algorithmic value.
In what sense is a safe $\lambda$-term safe?

A basic idea in lambda calculus / logic:

When performing $\beta$-reduction, one must use capture-avoiding substitution, which is standardly implemented by renaming bound variables afresh upon each substitution.

There is an algorithmic price to pay for renaming:

Any machine that correctly computes:

\[
\begin{cases}
\text{INPUT: } & \text{A simply-typed } \lambda\text{-term } M \\
\text{OUTPUT: } & \text{A } \beta\text{-reduction sequence from } M
\end{cases}
\]

needs an unbounded supply of names, and hence unbounded memory.

Safety lets us get away with no renaming of bound variables!
Safety reformulated as a simply-typed theory

We reexpress (and generalize) the safety constraint as a simply-typed theory. Sequents have the form

\[
\begin{align*}
x_1 : A_1, \ldots, x_i : A_i & \quad \cdots \quad | \quad \cdots \quad | \quad x_l : A_l, \ldots, x_n : A_n \vdash M : B \\
\text{level } l_1 & \quad \cdots \quad | \quad \cdots \quad | \quad \text{level } l_m
\end{align*}
\]

- Each \(A_i\) and \(B\) are homogeneous\(^a\).
- Typing context partitioned according to levels with \(l_1 \geq \cdots \geq l_m\).

Formation rules must respect (unicity of) the partition:

- When forming abstraction, all variables of the lowest type-partition must be abstracted in an atomic step.
- When forming application, the operator-term must be applied to all operand-terms (one for each type) of the highest type-partition, in one atomic step.

\(^a\)o is homogeneous; and \((A_1 \rightarrow \cdots \rightarrow A_n \rightarrow o)\) is homogeneous just if \(\text{level}(A_1) \geq \text{level}(A_2) \geq \cdots \geq \text{level}(A_n)\), and each \(A_i\) is homogeneous.
Safe $\lambda$-calculus makes algorithmic sense

**Examples.** Set $\Gamma = F : (o \rightarrow o) \rightarrow o \rightarrow o \rightarrow o \mid \varphi : o \rightarrow o \mid x : o, y : o$

1. $(F\varphi)x : o \rightarrow o$ is not safe.

2. $\lambda x y. F\varphi xy$ is safe but not $\lambda x. F\varphi xy$.

**Theorem.** “Safe $\lambda$-calculus = $\alpha$-conversion-free $\lambda$-calculus” I.e. when performing $\beta$-reductions on a safe (recursively-defined) $\lambda$-term, there is no need to rename bound variables when substituting.

Thus when reducing a safe $\lambda$-term, we do not need any supply of fresh name.

Safe $\lambda$-calculus seems of independent interest, and deserves further investigations. E.g. what kind of reasoning principles does it support (via Curry-Howard)? Does it have interesting models?

Nevertheless, we shall prove that safety is *not* necessary for decidability.
Two questions about safety

Is safety a genuine or spurious constraint for:

1. **Expressiveness.** Are there *inherently* unsafe $\Sigma$-trees?
   
   I.e. Is there an unsafe recursion scheme whose value tree is not the value tree of any safe recursion scheme? If so, at what level?

   **Conjecture.** Yes, at level 2. But note:

   **Theorem.** (A+deM+O FOSSACS 2005) There is no inherently unsafe word language at level 2.

2. **Decidability.** Is safety necessary for decidability? No, not at level 2.

   **Theorem.** (A+deM+O TLCA 2005) $\Sigma$-trees denoted by level-2 recursion schemes, whether safe or not, have decidable MSO theories.

   **Question.** What about higher levels?

   Yes: Decidability result extends to all levels - main result of this talk.
Outline of Talk

1. Level-$n$ Recursion Schemes and their Value Trees
2. A Model-Checking Problem
4. The Theorem and Proof Outline

Theorem.

(i) The modal mu-calculus model checking problem for trees generated by level-$n$ recursion schemes is $n$-EXPTIME complete, for all $n \geq 0$.

(ii) Hence trees generated by recursion schemes of every level, whether safe or not, have decidable MSO theories.

The level-2 case has also been obtained, independently, by Knapik, Niwiński, Urzyczyn + Walukiewicz (ICALP 2005) using a new kind of machines called “panic automata”.

**Theorem.** For every level-$n$ (deterministic) recursion scheme $G$, for every modal mu-calculus formula $\varphi$, it is decidable whether $\llbracket G \rrbracket \vDash \varphi$.

Thanks to Rabin, Emerson + Jutla, etc., equivalent to deciding if $\llbracket G \rrbracket$ is accepted by an alternating parity tree automaton $B$ – call $B$ the property APT.

Recall: $B$ accepts $\llbracket G \rrbracket$ iff $B$ has an accepting run-tree over $\llbracket G \rrbracket$.

**Proof approach:** Transfer algorithmic analysis from value tree $\llbracket G \rrbracket$ to an auxiliary computation tree $\lambda(G)$.

**Two major technical components:**

1. Strong 1-1 correspondence between paths in value tree and traversals over the computation tree, established using game semantics.

2. Recognition of (accepting) traversals.

   - P-views of a traversal over a computation tree are paths in the same tree.

   - Thus we can simulate traversals by certain paths over the computation tree, as formalised by the notion of traversal-simulating APT $C$. 

Computation trees, concretely

Fix a level-\(n\) recursion scheme \(G\). Transform \(G \mapsto \overline{G} \) :

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

2. **Explicit “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**.

The **computation tree** \(\lambda(G)\) is the infinite term-tree obtained by unfolding the transformed equations in \(\overline{G}\) – a level-0 recursion scheme! – *ad infinitum*.

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

**Semantically**: The computation tree \(\lambda(G)\) is just (a representation of) the uncovering (aka *fully revealed strategy*) of the value tree \([G]\), which is an innocent strategy.
\[
G : \left\{ \begin{array}{ll}
S &= FH \\
F \varphi &= \varphi(F \varphi) \\
H z &= fzz
\end{array} \right. \quad \mapsto \quad \overline{G} : \left\{ \begin{array}{ll}
S &= \@ F (\lambda x. H \lambda x) \\
F &= \lambda \varphi. \varphi (\@ F (\lambda y. \varphi (\lambda y))) \\
H &= \lambda z. f (\lambda z) (\lambda z)
\end{array} \right.
\]

The \textbf{computation tree} \( \lambda(G) \) is (the abstract syntax tree of) the \textbf{unfolding} of \( \overline{G} \):

Theorem. (Correspondence) Let $G$ be a level-$n$ recursion scheme.

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

(ii) Further, traversal $t_p$ is the uncovering of (and, hence, $\Sigma$-projects onto) path $p$.

Thus: Property APT $\mathcal{B}$ has an accepting run-tree over $[G]$ 

\[
\exists \text{ certain set of } \delta_\mathcal{B}\text{-respecting, state-annotated maximal paths in } [G] \text{ satisfying parity condition}
\]

Thm (Corr)

\[
\exists \text{ certain set of } \delta_\mathcal{B}\text{-respecting, state-annotated maximal traversals over } \lambda(G) \text{ satisfying parity condition}
\]

by def.

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

Problem: How to recognise such state-annotated traversals?

Higher-order traversals can be very complex!

A level-3 example:
Definition. **Traversals** over $\lambda(G)$ are justified sequences defined by induction:

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

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

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

- **(Var)** If $tn\lambda\xi\cdot\cdot\cdot\xi$ is a traversal, so is $tn\lambda\xi\cdot\cdot\cdot\xi\lambda\eta$.

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

A traversal jumps all over the comp. tree, and can visit a node infinitely often!

**Key lemma:** P-views of traversals are paths in the computation tree.
Simulate traversals by *paths* – A level-2 illustration

**Idea.** Simulate a traversal by the respective P-views of all its prefixes, which can be shown to be a set of 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 higher-order recursion scheme $G$, and a property APT $\mathcal{B} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ with $p$ priorities. Write $[p] = \{1, \ldots, p\}$.

\[
\begin{align*}
\mathbf{VP}_G^B(o) & = Var_G^o \times Q \times [p] \times 2^\emptyset \\
\mathbf{VP}_G^B(A_1 \rightarrow \cdots \rightarrow A_n \rightarrow o) & = Var_G^A \times Q \times [p] \times 2^{(\bigcup_{i=1}^n \mathbf{VP}_G^B(A_i))}
\end{align*}
\]

Asserting

\[(\varphi, q, m, c) \in \mathbf{VP}_G^B(A)\]

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 $c$”.
**Traversal-simulating APT C**: simulates $B$-states and verifies guesses

$C$-automata descend the computation tree with states $q \rho$ where $q$ is the $B$-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 level-0 variable $x$.
  
  If $\rho = \{ (x, q, m, \emptyset) \}$, succeed; otherwise abort.

- $l$ is a $\Sigma$-symbol $f : o^k \rightarrow o$.
  
  Guess a set $\{ (i_1, q_1), \cdots, (i_l, q_l) \}$ satisfying $\delta_B(q, f)$ (abort, if impossible), and guess environments $\rho_1, \cdots, \rho_l$ 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 $\lambda \eta_1, \cdots, \lambda \eta_k$.
  
  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 $\rho_1, \cdots, \rho_l$ with $\bigcup_{j=1}^l \rho_j = \rho$. For each $j$, spawn automaton with state $q_j (\rho_j \cup c_j)$ in direction $i_j$. 

**Theorem (Simulation).** The following are equivalent:

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

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

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

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

“(ii) $\Rightarrow$ (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.
Key Steps of Decidability Proof

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

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

$\iff$ \{ Theorems of Rabin, Muller + Schupp, Emerson + Jutla, etc. \}

Property APT $B_\varphi$ accepts the value tree $\llbracket G \rrbracket$

$\iff$ \{ Definition of APT \}

$B_\varphi$ has an accepting run-tree over the value tree $\llbracket G \rrbracket$

$\iff$ \{ Correspondence Theorem \}

$B_\varphi$ has an accepting traversal-tree over the computation tree $\lambda(G)$

$\iff$ \{ Simulation Theorem \}

Traversal-simulating APT $C$ has an accepting run-tree over $\lambda(G)$
Complexity of Modal Mu-Calculus Model Checking

Mu-calculus model checking of level-$n$ 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

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

**Theorem.** Given a property APT $\mathcal{B} = \langle Q, \Sigma, \delta, q_0, \Omega \rangle$ with $p$ priorities, and a level-$n$ recursion schemes $G$ (whether safe or not), acceptance of $\llbracket G \rrbracket$ by $\mathcal{B}$ is decidable in time $\exp_n O(|G| \cdot |Q| \cdot p)$.

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

1. **Conjecture**: There are *inherently* unsafe trees (at level 2) - Urzyczyn’s tree.

2. What is the automata-theoretic counterpart of (possibly unsafe) higher-order recursion schemes. E.g. Stirling’s *pointer machines*.

3. Definition of hierarchy of graphs generated by high-order recursion schemes? Are their MSO theories decidable? Relationship with CaucaH Hierarchy?

4. “Mixing semantic and verification games”: Denotational semantics of λ-calculus “relative to an alternating parity tree automaton (APT)”. Construct a CCC, parameterized by an APT, with maps witnessed by profiles (“guesses”).

5. Algorithmic properties of Σ-trees generated by stateful (Algol-like) rec. schemes.

6. Given a μ-formula over $[[G]]$, is its “winning region” computable?

7. Identify properties and/or subclasses of trees that are “feasibly” model-checkable.

**Safe λ-calculus, safe word and tree languages, higher-order PDAs:**

1. Safe λ-calculus (Idealised Algol?): Models? Proof theory (via Curry-Howard)?

2. Are safe word languages context-sensitive?

3. Higher-order (visibly) PDA; hot topic - 6 recent PhD theses! Equiv. problem.
Safe Lambda Calculus: System $S$ Typing Rules

\begin{align*}
\frac{(A_1 | \cdots | A_n | o) \text{ homogeneous}}{\overline{x}_1 : A_1 | \cdots | \overline{x}_n : A_n \vdash b : B}
\end{align*}

\begin{align*}
\frac{(A_1 | \cdots | A_n | o) \text{ homogeneous}}{\overline{x}_1 : A_1 | \cdots | \overline{x}_n : A_n \vdash x_{ij} : A_{ij}}
\end{align*}

\begin{align*}
\frac{\overline{x}_1 : A_1 | \cdots | \overline{x}_{n+1} : A_{n+1} \vdash M : B \quad (A_{n+1} | B) \text{ homogeneous}}{\overline{x}_1 : A_1 | \cdots | \overline{x}_n : A_n \vdash \lambda \overline{x}_{n+1}.M : (A_{n+1} | B)}
\end{align*}

\begin{align*}
\frac{\Gamma \vdash M : (B_1 | \cdots | B_m | o) \quad \Gamma \vdash N_1 : B_{11} \cdots \Gamma \vdash N_{l_1} : B_{1l_1}}{\Gamma \vdash MN_1 \cdots N_{l_1} : (B_2 | \cdots | B_m | o)}
\end{align*}

When forming abstraction, all variables of the (right-most) type-partition must be abstracted. When forming application, the operator-term must be applied to all operand-terms (one for each type) of the left-most type-partition.