Game Semantics and its Algorithmic Applications

(Lecture 3: Infinite Trees, Recursion Schemes and Game Semantics)

Luke Ong
Oxford University Computing Laboratory

www.comlab.ox.ac.uk/oucl/work/luke.ong/
Synopsis of Application 1

**Problem**: Find classes of finitely-presentable infinite-state systems with decidable MSO theories.

The hierarchy of trees generated by higher-order recursion schemes is an example of such a class, and it is a unifying framework.

We survey old and more recent work on a related hierarchy of recursion schemes satisfying the safety constraint, which are equivalent (as tree generators) to higher-order pushdown automata.

*Digression. Safe Lambda Calculus: Questions and Possible Directions*

Main Theorem and its Game-Semantic Proof:

**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 each \(n \geq 0\).

Many further directions.
Outline of Talk

1. **Level-\(n\) Recursion Schemes and their Value Trees**
2. A Model-Checking Problem
3. Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. The Safe Lambda Calculus
5. The Theorem and Proof Outline
Order of a Type

Types are ranged over by $A, B, \cdots$.

$$A ::= o \mid (A \to B)$$

Every type can be written uniquely as

$$A_1 \to \cdots \to A_n \to o, \quad n \geq 0$$

(arrows associate to the right), which is then abbreviated to $(A_1, \cdots, A_n, o)$. The order of a type measures how nested it is on the LHS of the arrow.

$$\text{order}(o) = 0$$

$$\text{order}(A \to B) = \max(\text{order}(A) + 1, \text{order}(B))$$

Notation. $e : A$ means “expression $e$ has type $A$”.
Order-\(n\) (Deterministic) Recursion Scheme \(G = (\mathcal{N}, \Sigma, \mathcal{R}, S')\)

Fix a set \(Var\) 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: \(f \in \Sigma\) 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 \(Var\),
- non-terminals \(D, F, G, \text{etc.}\) from \(\mathcal{N} - \{S\}\)
Examples

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

1. An order-0 example: No variables!

$$G_1 : \begin{cases} 
S &= f \top \top \\
T &= f' \top \top \\
U &= f \top \top 
\end{cases}$$

2. An order-2 example.

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

$$G_2 : \begin{cases} 
S &= F \varphi x \\
B \varphi \psi x &= \varphi(\psi x) \\
F \varphi &= f(\varphi a)(F(B \varphi \varphi)) 
\end{cases}$$
The \textit{value tree} $[G]$ of a recursion scheme $G$ is a possibly infinite applicative term \textit{constructed from the terminals}, which is obtained by unfolding the equations \textit{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 $\Sigma$-\textit{labelled (ranked and ordered) tree} (generated by $G$).

Formally a $\Sigma$-\textit{labelled tree} is a function $t : \text{dom}(t) \rightarrow \Sigma$ such that $\text{dom}(t) \subseteq \{1, \cdots, m\}^*$ is prefix-closed, and for all nodes $\alpha \in T$, the $\Sigma$-symbol $t(\alpha) \in \Sigma$ has arity $k$ iff $\alpha$ has $k$ children, namely $\alpha 1, \cdots, \alpha k \in T$. 
An order-2 example.

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

\[
\begin{align*}
G_2 : & \quad S = F g \\
& \quad B \varphi \psi x = \varphi (\psi x) \\
& \quad F \varphi = f (\varphi a) (F (B \varphi \varphi))
\end{align*}
\]

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

\[
\begin{align*}
\epsilon & \mapsto f & 11 & \mapsto a \\
1 & \mapsto g & 21 & \mapsto g \\
2 & \mapsto f & 22 & \mapsto f \\
\ldots & \mapsto & \ldots & \\
\end{align*}
\]
Outline of Talk

1. Level-$n$ Recursion Schemes and their Value Trees
2. A Model-Checking Problem
3. Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. The Safe Lambda Calculus
5. The Theorem and Proof Outline
A Model Checking Problem

Parametrized over logical language $\mathcal{L}$ and $n \geq 0$.

**Model Checking Problem ($\mathcal{L}$, order-$n$ $\Sigma$-labelled trees)**

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

Here we only consider $\mathcal{L} =$

- Monadic Second-Order Logic, and
- Modal mu-calculus.

Many other possibilities for further investigations: complexity of sublogics.
Monadic Second-Order Logic (for $\Sigma$-labelled 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: $\neg$, $\lor$ etc.

- first-order quantifications: $\forall x.\neg$, $\exists x.\neg$

- second-order quantifications: $\forall X.\neg$, $\exists X.\neg$. 


Game Semantics and its Algorithmic Applications: Lecture 3, Newton Institute, February 2006. 11
Why MSO Logic?

It is a kind of gold standard!

**MSO is very expressive.** Over graphs, MSO is strictly more expressive than the 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 $\varphi$, there is a modal mu-calculus formula $p_\varphi$ s.t. for every $\Sigma$-labelled tree $t$, we have $t \models \varphi \iff t, \varepsilon \models p_\varphi$.

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

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 $x = y$):

   
   \[
   \text{PrefCl}(X) \equiv \forall xy . y \in X \land \bigvee_{i=1}^{m} d_i(x, y) \rightarrow x \in X
   \]
   
   \[
   x \leq y \equiv \forall X . \text{PrefCl}(X) \land y \in X \rightarrow x \in X
   \]

**Reachability property:** “$X$ is a path”

\[
\text{Path}(X) \equiv \forall xy \in X . x \leq y \lor y \leq x
\]

\[
\land \forall xyz . x \in X \land z \in X \land x \leq y \leq z \rightarrow y \in X
\]

\[
\text{MaxPath}(X) \equiv \text{Path}(X) \land \forall Y . \text{Path}(Y) \land X \subseteq Y \rightarrow Y \subseteq X.
\]
**Recurrence Property**

A set of nodes is a cut if no two nodes in it are $\leq$-compatible, and it has a non-empty intersection with every maximal path.

$$\text{Cut}(X) \equiv \forall xy \in X . \neg(x \leq y \lor y \leq x)$$

$$\land \forall Z . \text{MaxPath}(Z) \rightarrow \exists z \in Z . z \in X$$

**Fact.** A set $X$ of nodes in a finitely-branching tree is finite iff there is a cut $C$ such that every $X$-node is a prefix of some $C$-node.

$$\text{Finite}(X) \equiv \exists Y . \text{Cut}(Y) \land \forall x \in X . \exists y \in Y . x \leq y$$

Hence “there are finitely many nodes labelled by $f$” is expressible in MSO by

$$\exists X . \text{Finite}(X) \land \forall x . \mathbf{p}_f(x) \rightarrow x \in X$$

**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
3. Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. The Safe Lambda Calculus
5. The Theorem and Proof Outline
Structures with decidable MSO theories: some milestones

1. Rabin 1969: Regular trees. “Mother of all decidability results”
   \[
   \text{PushdownTree}_n = \text{Trees generated by order-} n \text{ pushdown automata.}
   \]
   \[
   \text{SafeRecSch}_n = \text{Trees generated by order-} n \text{ safe recursion schemes.}
   \]
5. Cauca (MFCS 2002). \( T_n = \) Trees obtained from the regular \( \Sigma \)-labelled trees by \( n \)-fold inverse deterministic rational mappings, each followed by an unfolding.

**Theorem** (KNU-C). \( \forall n \geq 0, \text{PushdownTree}_n = \text{SafeRecSch}_n = T_n \)

**Question.** Do \( \Sigma \)-labelled trees generated by unsafe recursion schemes have decidable MSO theories? If so, at which orders?
Hierarchies of Finitely-Presentable Infinite Structures

Safety seems a robust definition: several characterisations

<table>
<thead>
<tr>
<th>Equivalent Higher-Order Generating Devices</th>
<th>Classes of Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Languages</td>
<td>Trees</td>
</tr>
<tr>
<td>Pushdown Automata</td>
<td>Maslov 74, 76</td>
</tr>
<tr>
<td>Safe Recursion Schemes</td>
<td>Damm 82</td>
</tr>
<tr>
<td>Indexed Grammars</td>
<td>Maslov 76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word Languages</th>
<th>Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order 0</td>
<td>Regular languages</td>
</tr>
<tr>
<td>Order 1</td>
<td>Context-free languages; e.g. $a^n b^n$</td>
</tr>
<tr>
<td>Order 2</td>
<td>Indexed languages; e.g. $a^n b^n c^n$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Regular trees (Rabin, etc.)
Algebraic trees (Bourcelle, etc.)
Hyperalgebraic trees (KNU 01)
Open Problems about the Maslov (= Damm) Hierarchy

Not much is known about level-3 and above.

1. **Pumping Lemma** (or Myhill-Nerode-type results)
   There are “pumping lemmas” for levels 0, 1 and 2 ([Hay73,Gil96]).
   *Pace* [Blumensath04] for whole Maslov Hierarchy – runs are pumpable, conditions given as lengths of runs and configuration size.

2. **Logical Characterization.**
   Regular languages are exactly those that are MSO definable (Büchi ’60).
   There is a characterization of context-free languages using quantification over matchings [LST94].

3. **Complexity-Theoretic Characterization.**
   Engelfriet ’83, ’91: characterizations of languages accepted by alternating / two-way / multi-head / space-auxiliary order-\(n\) PDA in terms of time-complexity classes (but no result for Maslov Hierarchy itself).

4. **Relationship with Chomsky Hierarchy.** E.g. is level 3 context-sensitive?
What is the safety constraint?


**Definition** [KNU02]. An order-2 equation is *unsafe* if the RHS has a sub-term $P$ such that

(i) $P$ is order 1

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

(iii) $P$ contains an order-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 (we shall see shortly) it has important algorithmic value.
Outline of Talk

1. Level-\(n\) Recursion Schemes and their Value Trees
2. A Model-Checking Problem
3. Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. The Safe Lambda Calculus
5. The Theorem and Proof Outline
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 a price to pay for naming:

Any machine that correctly computes:

\[
\begin{cases}
\text{INPUT:} & \text{A simply-typed \(\lambda\)-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

\[ \underbrace{x_1 : A_1, \ldots, x_i : A_i}_{\text{order } l_1} \quad \cdots \quad \underbrace{x_l : A_l, \ldots, x_n : A_n}_{\text{order } l_m} \vdash M : B \]

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

Formation rules must respect 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 \((A_1 \to \cdots \to A_n \to o)\) 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.
Safe $\lambda$-Calculus: System S Typing Rules

$$\frac{\text{(A}_1|\cdots|\text{A}_n|o) \text{ homogeneous}}{\text{x}_1: \text{A}_1|\cdots|\text{x}_n: \text{A}_n \vdash b : B}$$

$$\frac{\text{(A}_1|\cdots|\text{A}_n|o) \text{ homogeneous}}{\text{x}_1: \text{A}_1|\cdots|\text{x}_n: \text{A}_n \vdash x_{ij} : \text{A}_{ij}}$$

$$\frac{\text{x}_1: \text{A}_1|\cdots|\text{x}_{n+1}: \text{A}_{n+1} \vdash M : B \quad \text{(A}_{n+1}|B) \text{ homogeneous}}{\text{x}_1: \text{A}_1|\cdots|\text{x}_n: \text{A}_n \vdash \lambda\text{x}_{n+1}.M : (\text{A}_{n+1}|B)}$$

$$\frac{\Gamma \vdash M : (\text{B}_1|\cdots|\text{B}_m|o) \quad \Gamma \vdash N_1 : \text{B}_{11} \cdots \Gamma \vdash N_{l_1} : \text{B}_{1l_1}}{\Gamma \vdash MN_1\cdots N_{l_1} : (\text{B}_2|\cdots|\text{B}_m|o)}$$

When forming abstraction, all variables of the lowest-order type-partition must be abstracted. When forming application, the operator-term must be applied to all operand-terms (one for each type) of the highest-order type-partition.
Safe $\lambda$-calculus makes algorithmic sense

**Example.** Suppose $f : o^2 \rightarrow o$. Contracting the $\beta$-redex without renmaing

$$(\lambda \varphi^{(o,o)}.(\lambda x.\varphi x))(f x)$$

leads to variable capture. The term is *not* safe.

**Theorem.** “Safe $\lambda$-calculus = (a) $\alpha$-conversion-free $\lambda$-calculus”

In the safe lambda calculus, there is no need to rename bound variables when performing substitution $M[N_1/\varphi_1, \cdots, N_n/\varphi_n]$ provided the substitution is performed simultaneously on all free variables of the same order in $M$.

**Proof idea.** Suppose $\varphi$ free in $M$, and $x$ free in $N$, and $x$ captured in (capture permitting) $M[N/\varphi]$. Then $M$ looks like $\cdots (\lambda x.\cdots \varphi \cdots) \cdots$.

Case analysis by comparing $order(x)$ with $order(\varphi)$.

**Lemma.** A free variable in a safe term has order as least that of the term. □

Thus when reducing a safe $\lambda$-term, we do not need any supply of fresh name.
What is the right way to think of the Safe Lambda Calculus?

**Safe λ-calculus** seems of independent interest, and we don’t understand it.

**Design issues:** Is the homogeneity assumption really necessary?

**Proof theory:** What kind of reasoning principles does it support (via Curry-Howard)? Is it useful to automated deduction / theorem proving?

What is a **model** of safe λ-calculus? Does it have interesting models?

**Game semantics:** What kind of **pointer economy** does safety determine? Ans: Pointers are redundant in safe view-functions!

E.g. Kierstead terms: \( \lambda f. f(\lambda x. f(\lambda y. y)) \) is safe, but \( \lambda f. f(\lambda x. f(\lambda y. x)) \) is unsafe.

**Implicit complexity.** Simply-typed λ-calculus characterize polytime-computable numeric functions (Leivant-Marion 93). What about the safe terms?

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

Is safety a genuine or spurious constraint for:

1. **Expressiveness.** Are there *inherently* unsafe \( \Sigma \)-labelled 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 order?

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

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

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

**Theorem.** (A+deM+O 05 / KNUW 05) \( \Sigma \)-labelled trees denoted by order-2 recursion schemes, *whether safe or not*, have decidable MSO theories.

**Question.** What about higher orders?

Yes: Decidability result extends to all orders - main result of Part 2.