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

Level-n Recursion Scheme $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$

Fix a set Var of simply-typed variables.

- $\mathcal{N}$: Simply-typed non-terminals of level (= order) at most n

$$D : A_1 \to \cdots \to A_m \to o$$

  including a distinguished start symbol $S : o$.

- Σ: Terminals $f : \underbrace{o \to \cdots \to o}_{k} \to o$ (written $o^k \to o$) with $k \geq 0$

- $\mathcal{R}$: Equations for non-terminals $D : A_1 \to \cdots \to A_m \to o$ of the shape

$$D\,\varphi_1 \cdots \varphi_m \quad = \quad e$$

  where the applicative term $e : o$ is constructed from

  - terminals f, g, a, etc. from Σ
  - variables $\varphi_1 : A_1, \cdots, \varphi_m : A_m$ from Var,
  - non-terminals D, F, G, etc. from $\mathcal{N} - \{ S \}$

Examples

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

1. A level-0 example: No variables.

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

2. A level-2 example.

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

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

$[\![G]\!]$: Value Tree (or Denotation) of a Recursion Scheme G

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

$$G_1 \; : \; \left\{ \begin{array}{rcl} S & = & F\,a \\ F\,x & = & f\,x\,(F\,(g\,x)) \end{array} \right.$$

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

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

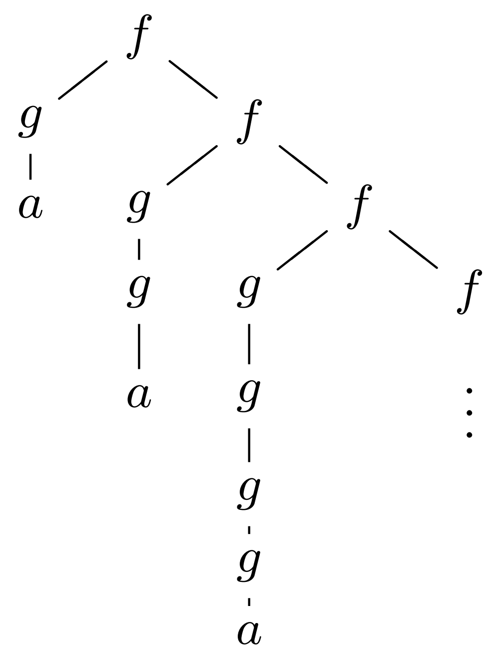
Formally a ***Σ-tree*** is a function $t : T \longrightarrow \Sigma$ such that $T \subseteq \{ 1, \cdots, m \}^*$ is prefix-closed, and for all occurrences $\alpha \in T$, the symbol $t(\alpha) \in \Sigma$ has arity k iff α has k children, which must be $\alpha\,1, \cdots, \alpha\,k \in T$.

A level-2 example.

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

$$G_2 \ : \ \left\{ \begin{array}{rcl} S & = & F\, g \\ B\, \varphi\, \psi\, x & = & \varphi\, (\psi\, x) \\ F\, \varphi & = & f\, (\varphi\, a)\, (F\, (B\, \varphi\, \varphi)) \end{array} \right.$$

The ***value tree***, $[\![\, G_2 \,]\!] : \{ 1, 2 \}^* \longrightarrow \Sigma$, is: $\left\{ \begin{array}{ccccccc} \epsilon & \mapsto & f & \qquad & 11 & \mapsto & a \\ 1 & \mapsto & g & & 21 & \mapsto & g \\ 2 & \mapsto & f & & 22 & \mapsto & f \\ & \cdots & & & \cdots & & \end{array} \right.$



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, and the Safe Lambda Calculus
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 Σ-TREES)

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

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 Σ-trees $t : T \longrightarrow \Sigma$)

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: $\mathbf{d}_i(x, y) \equiv$ "y is i-child of x"
2. Node labelling: $\mathbf{p}_f(x) \equiv$ "x has label f" where f is a Σ-symbol
3. Set-membership: $x \in X$

and closed under

- boolean connectives: $\neg, \vee, \wedge, \rightarrow$
- 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 $t \vDash \varphi \iff t, \epsilon \vDash p_\varphi$.

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. $T_a(i, t)$ = "i-th cell of the semi-infinite tape contains $a \in \Sigma$ at time t".

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 $x = y$):

$$\begin{array}{rcl} \mathsf{PrefCl}(X) & \equiv & \forall xy \,.\, y \in X \wedge \bigvee_{i=1}^{m} \mathbf{d}_i(x, y) \;\rightarrow\; x \in X \\ x \leq y & \equiv & \forall X \,.\, \mathsf{PrefCl}(X) \wedge y \in X \;\rightarrow\; x \in X \end{array}$$

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
3. **Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees, and the Safe Lambda Calculus**
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.
3. Caucal (ICALP 1996): Prefix-recognizable graphs (= ϵ-*closures* of configuration graphs of pushdown automata, Stirling 2000).
4. Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): Σ-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 Σ-trees generated by ***unsafe*** recursion schemes have decidable MSO theories? If so, at which levels?

Trees generated by *Safe* recursion schemes

	Trees
Level 0	Regular trees
Level 1	Generated by DPDAs

(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?

W. Damm: Derived types in "IO and OI Hierarchies", TCS 1982.

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: $f : o^2 \to o\ G, H : o$.

$$\begin{array}{rcl} G\,x & = & H\,\underline{(f\,x)} \\ F\,\varphi\,x\,y & = & f\,(F\,\underline{(F\,\varphi\,y)}\,y\,(\varphi\,x))\,a \end{array}$$

Safety (as presented above) seems syntactically awkward and semantically unnatural, but has important algorithmic value.

In what sense is a safe λ-term *safe*?

A basic idea in lambda calculus / logic:

When performing β-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:

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

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

$$\underbrace{x_1 : A_1, \cdots, x_i : A_i}_{\text{level } l_1} \mid \quad \cdots \quad \mid \underbrace{x_l : A_l, \cdots, x_n : A_n}_{\text{level } l_m} \vdash M : B$$

- 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 $(A_1 \to \cdots \to A_n \to o)$ is homogeneous just if $level(A_1) \geq level(A_2) \geq \cdots \geq level(A_n)$, and each A_i is homogeneous.

Safe λ-calculus makes algorithmic sense

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

1. $(F\varphi)x : o \to o$ is not safe.
2. $\boldsymbol{\lambda} x\, y.F\varphi x y$ is safe but not $\boldsymbol{\lambda} x.F\varphi x y$.

Theorem. "Safe λ-calculus = α-conversion-free λ-calculus"
I.e. when performing β-reductions on a safe (recursively-defined) λ-term, there is no need to rename bound variables when substituting.

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

Safe λ-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 Σ-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 decidabiliy? No, not at level 2.

> **Theorem**. (A+deM+O TLCA 2005) Σ-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
3. Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees, and the Safe Lambda Calculus
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 φ, it is decidable whether $[\![G]\!] \vDash \varphi$.

Thanks to Rabin, Emerson + Jutla, etc., equivalent to deciding if $[\![G]\!]$ is accepted by an alternating parity tree automaton $\mathcal{B}$ – call $\mathcal{B}$ the property APT.

Recall: $\mathcal{B}$ accepts $[\![G]\!]$ iff $\mathcal{B}$ has an accepting run-tree over $[\![G]\!]$.

Proof approach: Transfer algorithmic analysis from value tree $[\![G]\!]$ 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 $\mathcal{C}$.

Computation trees, *concretely*

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

1. Expand each RHS to its η-long form, including ground-type subterm in operand position. Thus $e : o$ η-expands to $\boldsymbol{\lambda}.e$ (“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.

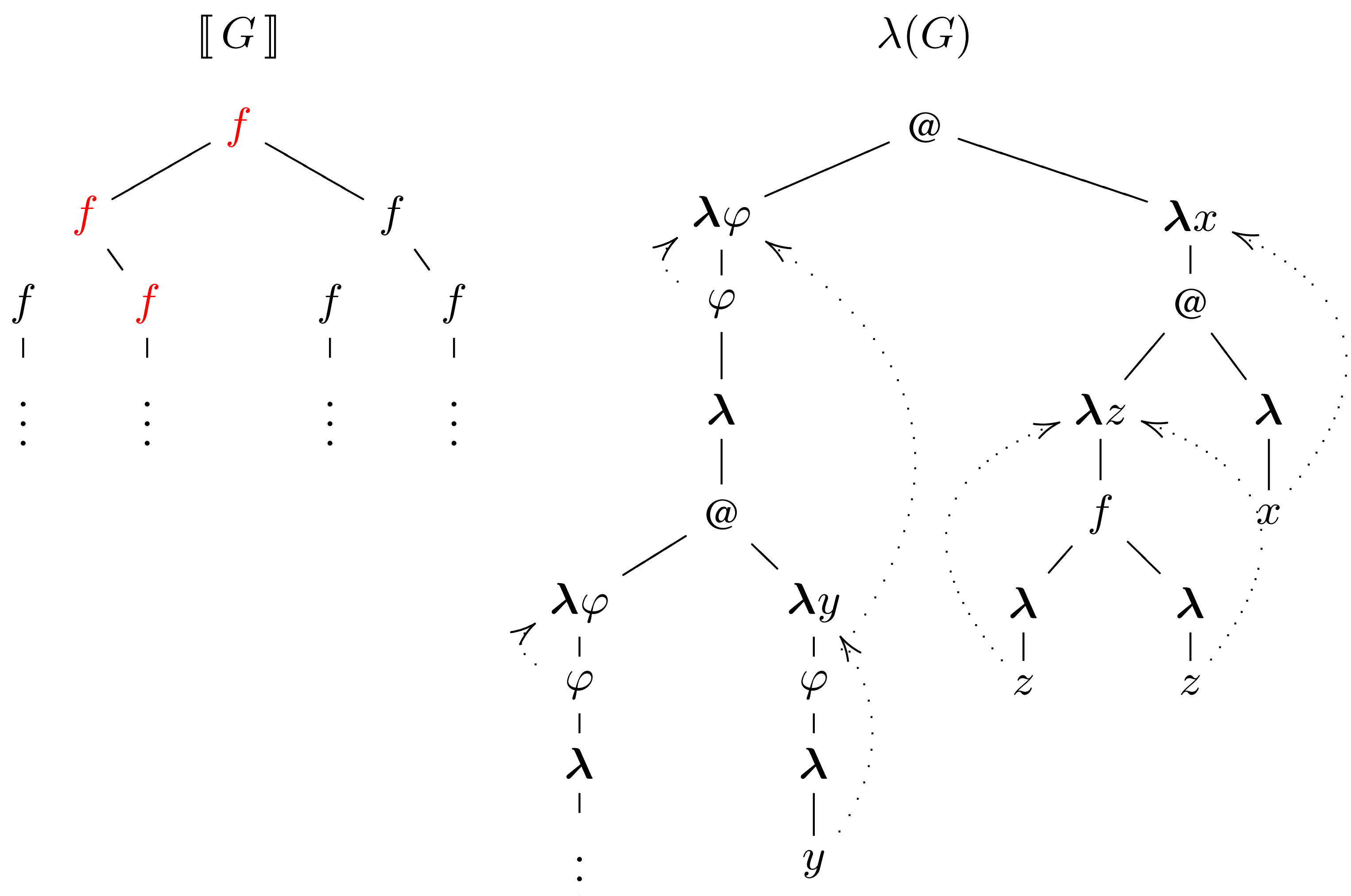
The ***computation tree*** $\lambda(G)$ is the infinite term-tree obtained by unfolding the transformed equations in $\overline{G}$ – a level-0 recursion scheme! – *ad infinitum*.

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.

$$G : \left\{ \begin{array}{rcl} S & = & F\,H \\ F\,\varphi & = & \varphi\,(F\,\varphi) \\ H\,z & = & f\,z\,z \end{array} \right. \qquad \mapsto \qquad \overline{G} : \left\{ \begin{array}{rcl} S & = & @\,F\,(\boldsymbol{\lambda} x.@\,H\,\boldsymbol{\lambda}.x) \\ F & = & \boldsymbol{\lambda}\varphi.\varphi(\boldsymbol{\lambda}.@\,F\,(\boldsymbol{\lambda} y.\varphi(\boldsymbol{\lambda}.y)))) \\ H & = & \boldsymbol{\lambda} z.f(\boldsymbol{\lambda}.z)(\boldsymbol{\lambda}.z) \end{array} \right.$$

The **computation tree** $\lambda(G)$ is (the abstract syntax tree of) the 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 (Σ-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, Σ-projects onto) path p.

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

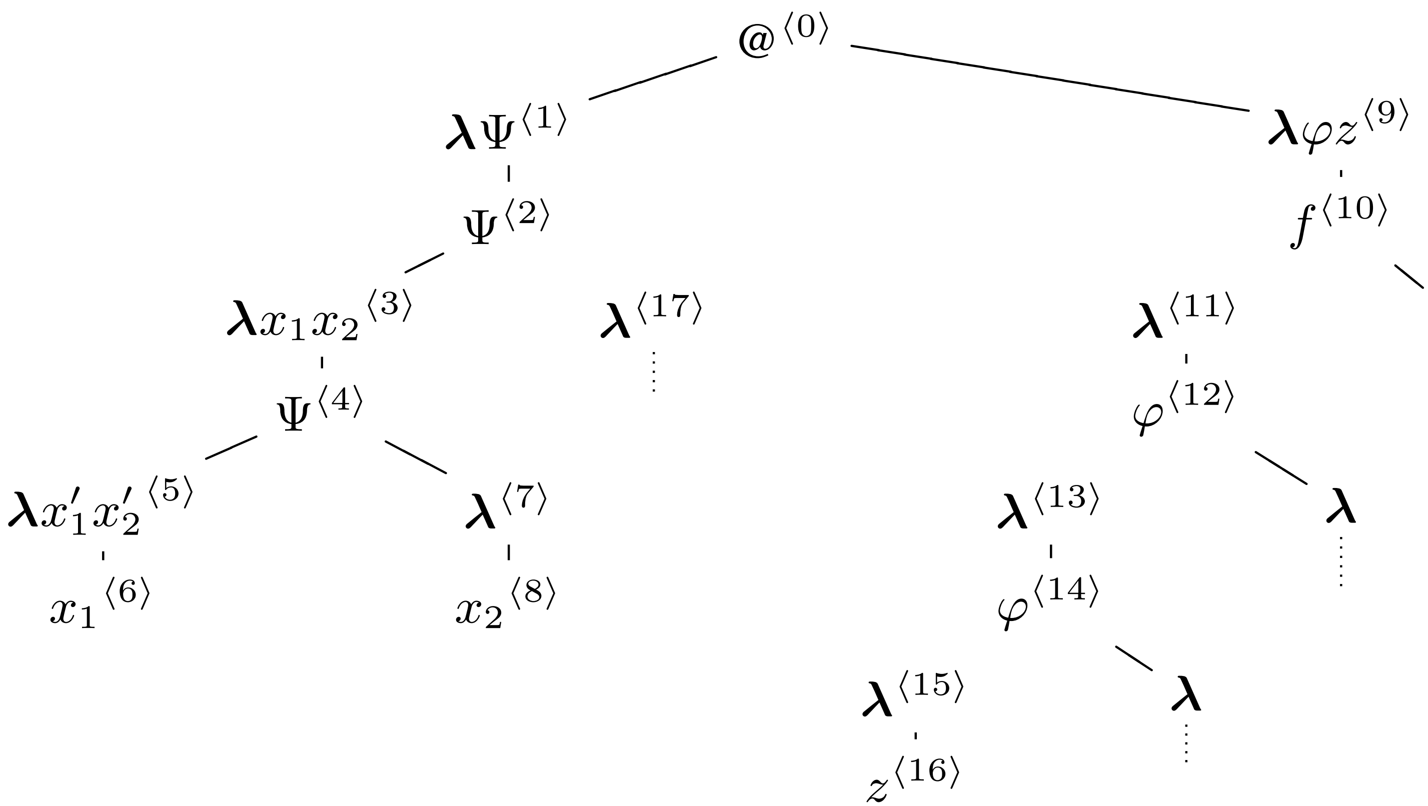
$\overset{\text{by def.}}{\iff}$ $\exists$ certain set of $\delta_{\mathcal{B}}$-respecting, state-annotated maximal paths in $[\![G]\!]$ satisfying parity condition

$\overset{\text{Thm (Corr)}}{\iff}$ $\exists$ certain set of $\delta_{\mathcal{B}}$-respecting, state-annotated maximal traversals over $\lambda(G)$ satisfying parity condition

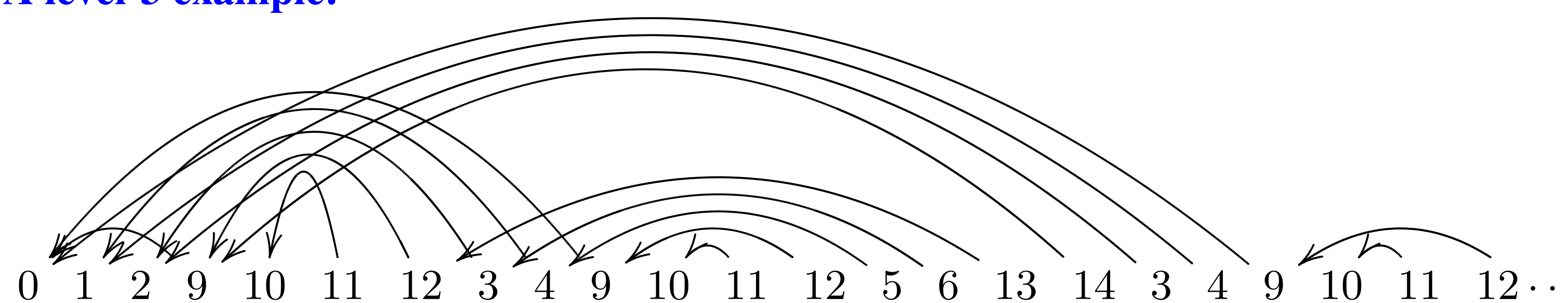
$\overset{\text{by def.}}{\iff}$ Property APT $\mathcal{B}$ has an accepting traversal-tree over $\lambda(G)$.

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 ϵ) is a traversal.

(App) If $t\ @$ is a traversal, so is $t\ @\ \boldsymbol{\lambda}\overline{\xi}$.

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

(Var) If $t\, n\ \boldsymbol{\lambda}\overline{\xi}\ \cdots\ \xi$ is a traversal, so is

$$t\ n\ \boldsymbol{\lambda}\overline{\xi}\ \cdots\ \xi\ \boldsymbol{\lambda}\overline{\eta}\ .$$

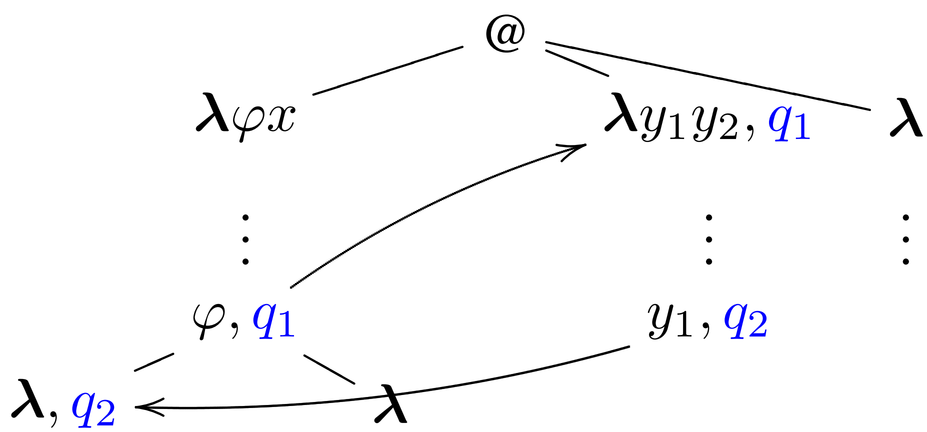
(Lam) If $t\ \boldsymbol{\lambda}\overline{\xi}$ is a traversal, so is $t\ \boldsymbol{\lambda}\overline{\xi}\ n$, such that $\ulcorner t\, \boldsymbol{\lambda}\overline{\xi}\, n \urcorner$ is a path in $\lambda(G)$.

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 φ at simulating state q_1 to a sibling subtree rooted at $\boldsymbol{\lambda} y_1 y_2$, subsequently exits it at y_1 and rejoins the original subtree at first $\boldsymbol{\lambda}$-child of φ with state q_2.

Simulate the traversal by paths:

- At φ with q_1, guess that the detour will return at first $\boldsymbol{\lambda}$-child with state q_2
- Spawn an automaton at $\boldsymbol{\lambda} y_1 y_2$ to verify the guess.

Formalising the guesses as Variable Profiles $\mathbf{VP}_G^{\mathcal{B}}(A)$

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 $[p] = \{\, 1, \cdots, p \,\}$.

$$\begin{array}{rcl} \mathbf{VP}_G^{\mathcal{B}}(o) & = & \mathit{Var}_G^o \times Q \times [p] \times 2^{\varnothing} \\ \mathbf{VP}_G^{\mathcal{B}}(A_1 \to \cdots \to A_n \to o) & = & \mathit{Var}_G^A \times Q \times [p] \times 2^{(\bigcup_{i=1}^n \mathbf{VP}_G^{\mathcal{B}}(A_i))} \end{array}$$

Asserting

$$(\varphi, q, m, c) \quad \in \quad \mathbf{VP}_G^{\mathcal{B}}(A)$$

at node α of computation tree means: the traversal being simulated will reach some descendant-node that is labelled φ

(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 φ) will eventually return to the children of the node labelled φ "in accord with c".

Traversal-simulating APT $\mathcal{C}$: simulates $\mathcal{B}$-states and verifies guesses

$\mathcal{C}$-automata descend the computation tree with states $q\,\rho$ where q is the $\mathcal{B}$-state being simulated, and environment ρ 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 $\rho = \{\, (x, q, m, \varnothing)\,\}$, succeed; otherwise abort.

- l is a Σ-symbol $f : o^k \to o$.

  Guess a set $\{\,(i_1, q_1), \cdots, (i_l, q_l)\,\}$ satisfying $\delta_{\mathcal{B}}(q, f)$ (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 direciton i_j.

- l is an @ with children labelled by $\boldsymbol{\lambda}\overline{\varphi}$ and $\boldsymbol{\lambda}\overline{\eta_1}, \cdots, \boldsymbol{\lambda}\overline{\eta_k}$.

  Guess $\rho' = \{\,(\varphi_{i_j}, q_j, m_j, c_j) : 1 \le j \le l\,\}$, 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 φ a modal mu-calculus formula.

$$
\begin{array}{ll}
 & \text{Value tree } [\![\, G \,]\!] \text{ satisifes } \varphi \\
\Longleftrightarrow & \{ \text{ Theorems of Rabin, Muller + Schupp, Emerson + Jutla, etc.} \} \\
 & \text{Property APT } \mathcal{B}_\varphi \text{ accepts the value tree } [\![\, G \,]\!] \\
\Longleftrightarrow & \{ \text{ Definition of APT } \} \\
 & \mathcal{B}_\varphi \text{ has an accepting run-tree over the value tree } [\![\, G \,]\!] \\
\Longleftrightarrow & \{ \text{ Correspondence Theorem } \} \\
 & \mathcal{B}_\varphi \text{ has an } \textit{accepting traversal-tree} \text{ over the computation tree } \lambda(G) \\
\Longleftrightarrow & \{ \text{ Simulation Theorem } \} \\
 & \text{Traversal-simulating APT } \mathcal{C} \text{ has an accepting run-tree over } \lambda(G)
\end{array}
$$

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 $[\![\, G \,]\!]$ by $\mathcal{B}$ is decidable in time $\mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$.

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 Caucal 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 $\mathcal{S}$ Typing Rules

$$\frac{(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid o) \text{ homogeneous} \qquad b \text{ is a type-}B \text{ constant}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash b : B}$$

$$\frac{(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid o) \text{ homogeneous}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash x_{ij} : A_{ij}}$$

$$\frac{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_{n+1}} : \overline{A_{n+1}} \vdash M : B \qquad (\overline{A_{n+1}} \mid B) \text{ homogeneous}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash \boldsymbol{\lambda} \overline{x_{n+1}}.M : (\overline{A_{n+1}} \mid B)}$$

$$\frac{\Gamma \vdash M : (\overline{B_1} \mid \cdots \mid \overline{B_m} \mid o) \qquad \Gamma \vdash N_1 : B_{11} \ \cdots \ \Gamma \vdash N_{l_1} : B_{1l_1}}{\Gamma \vdash M N_1 \cdots N_{l_1} : (\overline{B_2} \mid \cdots \mid \overline{B_m} \mid o)}$$

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.