Automata, Logic and Games: Theory and Application

Higher-Order Model Checking 1 / 2

Luke Ong

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

TACL Summer School
University of Salerno, 14-19 June 2015

Themes in Logic in Computer Science

Interface between (Computer-Aided Formal) Verification and Semantics of Computation

Logical and Algorithmic Foundations of Verification

- Automata on infinite trees as computational models of state-based systems
- Logical systems for describing correctness properties
- Two-person games as an abstract model of interaction between a reactive system and its environment

Semantics of Higher-Order Computation

- Lambda calculus as a definitional device
- Game semantics as accurate, intensional model
- Type systems: compositional, syntax-directed inference systems of behavioural properties

Model checking: an approach to program verification that promises accurate analysis with push-button automation.

Verification Problem: Given system Sys (e.g. OS), and correctness property $Spec$ (e.g. deadlock freedom), does Sys satisfy $Spec$?

The model checking approach:

1. Find an abstract model $\mathcal{M}$ of the system Sys.
2. Describe property $Spec$ as a formula φ of a decidable logic.
3. Exhaustively check if φ is violated by $\mathcal{M}$.

Major progress in verification of 1st-order imperative programs. Many tools: SLAM, Blast, Terminator, SatAbs, etc.

Two key techniques: State-of-the-art tools use

1. abstraction refinement techniques: CEGAR (Counter-Example Guided Abstraction Refinement)
2. acceleration methods such as SAT- and SMT-solvers.

Examples of Higher-Order / Functional Languages: OCaml, F#, Haskell, Lisp/Scheme, JavaScript, and Erlang; even C++.

Why Higher-Order / Functional Languages?

1. Functional programs are succinct, less error-prone, easy to write and maintain, good for prototyping.
2. Lambdas (closures) and streams now standard in today's leading languages (TIOBE Index): Java8, C++11, C#5.0, Javascript, Perl5, Python, Scala.
3. FL support domain-specific languages and organise data parallelism well; increasingly prevalent in scientific applications and financial modelling
4. Attractive for concurrent programminng (multicore, GPU-processing and cloud computing), thanks to absence of mutable variables and monadic structuring principles

Verifying Functional Programs

Two Standard Approaches

1. Type-Based Program Analysis.
   - sound and scalable but often imprecise ("curse of false positives")
   E.g. type-and-effect system (region-based memory management), qualifier types, linear types, intersection types, resource usage (sized types), etc.
2. Theorem Proving and Dependent Types
   - accurate, typically requires human intervention; does not scale well
   E.g. Coq, Agda, etc.

Model-Checking Higher-Order Programs is Intrinsically Hard

1. Infinite-state and extremely complex: Even without recursion, higher-order programs over a finite base type are infinite-state.
2. Many other sources of infinity: data structures, control structures (with recursion), concurrency, distribution and asynchronous communication, real-time and embedded systems, systems with parameters, etc.
3. Models of higher-order features as studied in semantics – are typically too "abstract" to support any algorithmic analysis.

   A notable exception is game semantics.

Higher-Order Model Checking is the model checking of infinite trees which are generated by recursion schemes (equivalently, $\lambda\mathbf{Y}$-calculus), with a view to formally analysing higher-order computation.

Outline of Part 2

1. Relating Families of Generators of Infinite Trees / Graphs: Recursion Schemes and (Collapsible) Pushdown Automata
2. Algorithmics and Expressivity
3. Reducing Model Checking to Type Inference
4. Compositional Model Checking of Higher-Type Böhm Trees
5. Practical Algorithms for Higher-Order Model Checking
6. Conclusions and Further Directions

Types $\quad \kappa \quad ::= \quad \mathsf{o} \quad | \quad (\kappa \to \kappa')$

Every type can be written uniquely as

$$\kappa_1 \to (\kappa_2 \to \cdots \to (\kappa_n \to \mathsf{o}) \cdots), \quad n \geq 0$$

often abbreviated to $\kappa_1 \to \kappa_2 \to \cdots \to \kappa_n \to \mathsf{o}$. Arrows associate to the right.

Order of a type: measures "nestedness" on LHS of $\to$.

$$\begin{array}{rcl} \mathsf{order}(\mathsf{o}) & := & 0 \\ \mathsf{order}(\kappa \to \kappa') & := & \max(\mathsf{order}(\kappa) + 1, \mathsf{order}(\kappa')) \end{array}$$

Examples. $\mathbb{N} \to \mathbb{N}$ and $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ both have order 1;
$(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ has order 2.

Notation. $\quad e : \kappa \quad$ means "expression e has type κ".
Applications associate to the left: write $f\,g\,h\,a$ to mean $((f\,g)\,h)\,a$.

Higher-order recursion schemes [Par68, Niv72, NC78, Dam82,...]

An order-n recursion scheme = closed, ground-type term definable in order-n fragment of $\lambda\mathbf{Y}$-calculus (i.e. simply-typed λ-calculus with recursion and uninterpreted order-1 constant symbols).

We use recursion schemes to define infinite trees.

Example: An order-1 recursion scheme. Fix an alphabet of 1st-order constants $\Sigma = \{ f : o \to o \to o,\ g : o \to o,\ a : o \}$.

$$G \ : \begin{cases} S & \to \quad F\,a \\ F\,x & \to \quad f\,x\,(F\,(g\,x)) \end{cases}$$

Unfolding from the start symbol S:

$$\begin{array}{lll} S & \to & F\,a \\ & \to & f\,a\,(F\,(g\,a)) \\ & \to & f\,a\,(f\,(g\,a)\,(F\,(g\,(g\,a)))) \\ & \to & \cdots \end{array}$$

The term-tree thus generated, tree(G), is $f\,a\,(f\,(g\,a)\,(f\,(g\,(g\,a))(\cdots)))$.

A recursion scheme is a quadruple $G = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ where

- Σ is a set of first-order constant symbols (tree constructors); write elements of Σ as $a, b, \ldots$
- $\mathcal{N}$ is a set of function symbols; write elements of $\mathcal{N}$ as $F, H, \ldots$
- $S \in \mathcal{N}$ is a distinguished *start symbol* where $S : o$
- $\mathcal{R}$ is a set of rewrite rules of the form, one for each $F \in \mathcal{N}$:

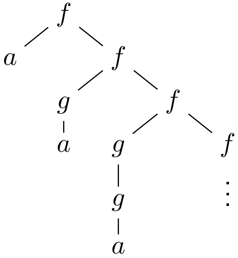
$$F\, x_1 \cdots x_k \;\rightarrow\; e$$

where $F : \kappa_1 \rightarrow \cdots \rightarrow \kappa_k \rightarrow o$, and $e : o$ is an applicative term built up from $\mathcal{N} \cup \{ x_1, \cdots, x_k \}$.

The order of G is the highest order of the function symbols in $\mathcal{N}$.

Representing the term-tree $\mathsf{tree}(G)$ as a Σ-labelled tree

$\mathsf{tree}(G) = f\,a\,(f\,(g\,a)\,(f\,(g\,(g\,a))(\cdots)))$ is the term-tree



Formally the term-tree, $\mathsf{tree}(G)$, is a map $T \longrightarrow \Sigma$, where T is a prefix-closed subset of $\{1, \cdots, m\}^*$, and m is the maximal arity of symbols in Σ.

$\mathsf{tree}(G)$ is ranked and ordered trees.

(Think of $\mathsf{tree}(G)$ as the Böhm tree of G.)

An Order-3 Example: Fibonacci Numbers

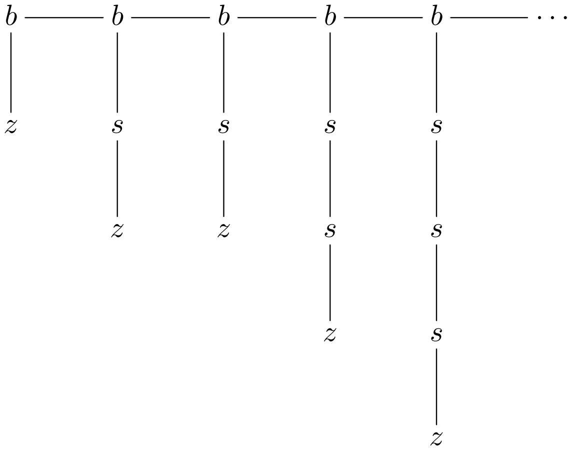
`fib` generates an infinite spine, with each member (in unary) of the Fibonacci sequence appearing in turn as a left branch from the spine.

Constants: $b : o \to o \to o,\ s : o \to o,\ z : o$

Functions: Write Church as a shorthand for $(o \to o) \to o \to o$

$$\begin{array}{rcl} S & : & o \\ \mathit{Zero} & : & \mathit{Church} \\ \mathit{One} & : & \mathit{Church} \\ \mathit{Show} & : & \mathit{Church} \to \mathit{Church} \to o \\ \mathit{Add} & : & \mathit{Church} \to \mathit{Church} \to (o \to o) \to o \to o \end{array}$$

$$\texttt{fib} \left\{ \begin{array}{rcl} S & \to & \mathit{Show}\, \mathit{Zero}\, \mathit{One} \\ \mathit{Zero}\, \varphi\, x & \to & x \\ \mathit{One}\, \varphi\, x & \to & \varphi\, x \\ \mathit{Show}\, n_1\, n_2 & \to & b\, (n_1\, s\, z)\, (\mathit{Show}\, n_2\, (\mathit{Add}\, n_1\, n_2)) \\ \mathit{Add}\, n_1\, n_2\, \varphi\, x & \to & n_1\, \varphi\, (n_2\, \varphi\, x) \end{array} \right.$$



Using recursion schemes as generators of word languages

Idea: A word is just a linear tree.

Represent a finite word "$a\,b\,c$" (say) as the applicative term $a\,(b\,(c\,e))$, viewing a, b and c as Σ-symbols of arity 1, where e is a distinguished nullary end-of-word marker.

1. A word language is regular iff it is generated by an order-0 recursion scheme.
2. A word language is context-free iff it is generated by an order-1 recursion scheme.

What class of word languages do order-2 recursion schemes define?

Higher-order pushdown automata (HOPDA) [Maslov 74]

Order-2 pushdown automata

A 1-stack is an ordinary stack. A 2-stack (resp. $(n+1)$-stack) is a stack of 1-stacks (resp. n-stack).

Operations on 2-stacks: s_i ranges over 1-stacks.

$$
\begin{array}{rcll}
push_2 & : & [s_1 \cdots s_{i-1} \underbrace{[\gamma_1 \cdots \gamma_n]}_{s_i}] & \mapsto \quad [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]] & \mapsto \quad [s_1 \cdots s_{i-1}] \\
push_1\, \gamma & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]] & \mapsto \quad [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n\, \gamma]] \\
pop_1 & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n\, \gamma_{n+1}]] & \mapsto \quad [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]]
\end{array}
$$

Idea extends to all finite orders: an order-n PDA has an order-n stack, and has $push_i$ and pop_i for each $1 \leq i \leq n$.

Example: $L := \{ a^n\, b^n\, c^n : n \geq 0 \}$ is recognisable by an order-2 PDA

L is not context free—thanks to the "$uvwxy$ Lemma".

Idea: Use top 1-stack to process $a^n\, b^n$, and height of 2-stack to remember n.

$$
\begin{tikzcd}
q_1\ \texttt{[[]]} \arrow[r, "a"] & q_1\ \texttt{[[][}z\texttt{]]} \arrow[r, "a"] & q_1\ \texttt{[[][}z\texttt{][}z\ z\texttt{]]} \arrow[d, "b"] \\
 & & q_2\ \texttt{[[][}z\texttt{][}z\texttt{]]} \arrow[d, "b"] \\
q_3\ \texttt{[[]]} & q_3\ \texttt{[[][}z\texttt{]]} \arrow[l, "c"'] & q_2\ \texttt{[[][}z\texttt{][]]} \arrow[l, "c"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
q_1 \arrow[loop above, "- \xrightarrow{a} push_2\, ;\, push_1 z"] \arrow[r, "z \xrightarrow{b} pop_1 z"'] & q_2 \arrow[loop above, "z \xrightarrow{b} pop_1"] \arrow[r, "\perp \xrightarrow{c} pop_2"'] & q_3 \arrow[loop above, "z \xrightarrow{c} pop_2"]
\end{tikzcd}
$$

'read a' 'read b' 'read c'

Some properties of the Maslow Hierarchy (Maslov 74, 76)

1. HOPDA define an infinite hierarchy of word languages.
2. Low orders are well-known: orders 0, 1 and 2 are the regular, context free, and indexed languages (Aho 68). Higher-order languages are poorly understood.
3. For each $n \geq 0$, the order-n languages form an abstract family of languages (closed under $+, \cdot, (-)^*$, intersection with regular languages, homomorphism and inverse homo.)
4. For each $n \geq 0$, the emptiness problem for order-n PDA is decidable.

A recent breakthrough

Theorem (Inaba + Maneth FSTTCS08)

All languages of the Maslov Hierarchy are context-sensitive.

Proof uses macro tree transducers (Engelfriet); order-n languages $\subseteq$ image of n iterates of MTTs

Theorem (Engelfriet 1991)

Let $s(n) \geq \log(n)$.

(i) *For* $k \geq 0$, *word acceptance problem of non-det. order-*k *PDA augmented with a two-way work-tape with* $s(n)$ *space is* k*-EXPTIME complete.*

(ii) *For* $k \geq 1$, *the acceptance problem of alternating order-*k *PDA augmented with a two-way work-tape with* $s(n)$ *space is* $(k-1)$*-EXPTIME complete.*

(iii) *For* $k \geq 0$, *the acceptance problem of alternating order-*k *PDA is* k*-EXPTIME complete.*

(iv) *For* $k \geq 1$, *the emptiness problem of non-det. order-*k *PDA is* $(k-1)$*-EXPTIME complete.*

Relating the two generator-families: word-language case

Theorem (Equi-expressivity)

For each $n \geq 0$, the three formalisms

1. *order-n pushdown automata (Maslov 76)*
2. *order-n safe recursion schemes (Damm 82, Damm + Goerdt 86)*
3. *order-n indexed grammars (Maslov 76)*

generate the same class of word languages.

What is safety?

Two Families of Generators of Infinite Structures

Summary

Higher-order pushdown automata can be used as recognising/generating device for

1. finite-word languages (Maslov 74) and ω-word languages
2. possibly-infinite ranked trees (KNU01) and, more generally, languages of such trees
3. possibly infinite graphs (Muller+Schupp 86, Courcelle 95, Cachat 03), *qua* configuration graphs of these pushdown systems

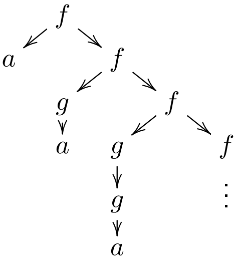
Higher-order recursion schemes can also be used to generate word languages, potentially-infinite trees (and languages there of) and graphs.

The two families are closely related.

A challenge problem in verification of higher-order computation

Example: Consider tree(G) on the right

- $\varphi_1 =$ "Infinitely many f-nodes are reachable".
- $\varphi_2 =$ "Only finitely many g-nodes are reachable".



Every node on the tree satisfies $\varphi_1 \vee \varphi_2$.

Monadic second-order (MSO) logic can describe properties such as $\varphi_1 \vee \varphi_2$.

MSO Model-Checking Problem for Order-n Recursion Schemes

- INSTANCE: An order-n recursion scheme G, and an MSO formula φ
- QUESTION: Does the Σ-labelled tree tree(G) satisfy φ?

Is the above problem decidable?

Monadic Second-Order Logic (for Σ-labelled trees)

where Σ is a ranked alphabet.

First-order variables: x, y, z, etc. (ranging over nodes)

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

MSO formulas are built up from **atomic formulas**:

- Parent-child relationship between nodes: $\mathbf{d}_i(x, y)$, for $1 \leq i \leq m$
- Node labelling: $\mathbf{p}_f(x)$, for $f \in \Sigma$
- Set-membership: $x \in X$

and closed under boolean connectives, first and second-order quantifications.

Why study monadic second-order (MSO) logic?

Because it is the gold standard of logics for describing correctness properties of reactive systems.

- MSO is *very* expressive.
  Over graphs, MSO is more expressive than the modal mu calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL$*$, etc.) can embed.
- It is hard to extend MSO meaningfully without sacrificing decidability where it holds.

A (selective) survey of MSO-decidable structures: up to 2002

- Rabin 1969: Infinite binary trees and regular trees. "Mother of all decidability results in algorithmic verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognisable graphs (ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): $\textbf{PushdownTree}_n\Sigma$ = Trees generated by order-n pushdown automata.

  $\textbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-n safe rec. schemes.
- Subsuming all the above:
  Caucal (MFCS 2002). $\textbf{CaucalTree}_n\Sigma$ and $\textbf{CaucalGraph}_n\Sigma$.

Theorem (KNU-Caucal 2002)

For $n \geq 0$, $\textbf{PushdownTree}_n\Sigma = \textbf{SafeRecSchTree}_n\Sigma = \textbf{CaucalTree}_n\Sigma$; and they have decidable MSO theories.

What is the safety constraint on recursion schemes? (1)

There is another notion of "higher-order" types:

Safe types

$D_0 \ := \ \{\, trees \,\}$

Order-0 objects are trees.

$D_{i+1} \ := \ \bigcup_{k \geq 0} [\underbrace{D_i \times \cdots \times D_i}_{k} \to D_i]$

Order $(i+1)$-objects are functions from (tuples of) order-i objects to order-i objects.

Define s-order$(t) := i \quad$ if $t \in D_i$

Safe (or "Derived") types

- OI-Hierarchy (Damm 82)
- Higher-level tree transducer (Engelfriet & Vogler 88)

What is the safety constraint on recursion schemes? (2)

Safety is a set of constraints on where variables may occur in a term.

Definition (Damm TCS 82, KNU FoSSaCS'02)

No order-k subterm of a safe term can contain free variables of order $<k$.

Example (unsafe rule).

$$F : (o \to o) \to o \to o \to o,\ f : o \to o \to o,\ x, y : o.$$

$$F\,\varphi\,x\,y \quad = \quad f\,(F\,\underline{(F\,\varphi\,y)}\,y\,(\varphi\,x))\,a$$

The subterm $F\ \varphi\ y$ has order 1, but the free variable y has order 0.

What is the point of safety?

Safety does have an important algorithmic advantage!

Theorem (KNU 02, Blum + O. TLCA 07, LMCS 09)

Substitution (hence β-reduction) in safe λ-calculus can be safely implemented without renaming bound variables! Hence no fresh names needed!

Theorem (Expressivity)

1. *(Schwichtenberg 76) The numeric functions representable by simply-typed λ-terms are multivariate polynomials with conditional.*
2. *(Blum + O. LMCS 09) The numeric functions representable by simply-typed safe λ-terms are the multivariate polynomials.*

(See (Blum + O. LMCS 09) for a study on the safe lambda calculus .)

1. MSO decidability: Is safety a genuine constraint for decidability?
   I.e. do trees generated by (arbitrary) recursion schemes have decidable MSO theories?
2. Machine characterisation: Find a hierarchy of automata that characterise the expressive power of recursion schemes.
3. Expressivity: Is safety a genuine constraint for expressivity?
   I.e. are there inherently unsafe word languages / trees / graphs?
4. Graph families:
   1. Definition: What is a good definition of "graphs generated by recursion schemes"?
   2. Model-checking properties: What are the decidable theories of the graph families?

A Tale of Two Higher-Order Systems

Damm's Safe Types (TCS 82) $D_{i+1} \ := \ \bigcup_{k \geq 0} [\underbrace{D_i \times \cdots \times D_i}_{k} \to D_i]$	**Church's Simple Types** (JSL 40) $\kappa := o \mid \kappa \to \kappa'$
MSO model checking is decidable (KNU 02)	**Q1**?
Safe RS equi-expressive with HOPDA (Damm 82, KNU 02)	**Q2**: Equi-expressive with HOPDA++?
	Q3: Are there inherently unsafe word languages / trees / graphs?
Hierarchy is strict (Damm 82)	?
Word languages are context-sensitive (Inaba & Maneth 08)	?

Q1. Do trees generated by HORS have decidable MSO theories?

Theorem (Aehlig, de Miranda + O. TLCA 2005)

Trees generated by order-2 recursion schemes (whether safe or not) have decidable MSO theories.

Theorem (Knapik, Niwinski, Urczyczn + Walukiewicz, ICALP 2005)

Modal mu-calculus model checking problem for homogenously-typed order-2 schemes (whether safe or not) is 2-EXPTIME complete.

What about higher orders?

Yes: MSO decidability extends to all orders (O. LICS06).

Q1. Do trees generated by HORS have decidable MSO theories? Yes

- (Rabin 69): MSOL and parity tree automata are effectively equi-expressive for tree languages.
- (EJ 91): mu-calculus and alternating parity tree automata (APT) are effectively equi-expressive for tree languages.
- Mu-calculus and MSOL are equi-expressive for tree languages. (JW 96): mu-calculus is the bisimulation-invariant fragment of MSOL.

Theorem (O. LICS 2006)

For $n \geq 0$, the mu-calculus model-checking problem for trees generated by order-n recursion schemes is n-EXPTIME complete. Thus these trees have decidable MSO theories.

Remarks on the Proof (1)

$\lambda(G)$ is the tree-unravelling of the underlying (finite) syntax graph of G.

Theorem (Transfer)

Given Σ, there is an effective transformation of APT, α, such that for every HORS G, we have $\mathcal{B}$ accepts $\mathsf{tree}(G)$ *iff $\alpha(\mathcal{B})$ accepts $\lambda(G)$.*

$\lambda(G)$ is regular; and APT acceptance problem of regular trees is decidable. Hence:

Corollary

The modal mu-calculus model checking problem for HORS is decidable.

$\lambda(G)$ is the tree-unravelling of the underlying (finite) syntax graph of G.

Theorem (Transfer)

Given Σ, there is an effective transformation of APT, α, such that for every HORS G, we have $\mathcal{B}$ accepts $\mathrm{tree}(G)$ *iff $\alpha(\mathcal{B})$ accepts $\lambda(G)$.*

Extension to infintary HORS

Given finite sets $\mathcal{T}$ and $\mathcal{V}$ of types and variables respecively, $G = \langle\, \Sigma, \mathcal{N}, \mathcal{R}, S \,\rangle \in \mathbf{RS}^{\infty}(\mathcal{T}, \mathcal{V})$ just if types of all subterms are in $\mathcal{T}$, and rules may only use variables from $\mathcal{V}$, but $\mathcal{N}$ and $\mathcal{R}$ may be infinite.

Theorem (O. 2013)

Given $\Sigma, \mathcal{T}, \mathcal{V}$, there is an effective transformation of APT, α, s.t. for every $G \in \mathbf{RS}^{\infty}(\mathcal{T}, \mathcal{V})$, we have $\mathcal{B}$ accepts $\mathrm{tree}(G)$ *iff $\alpha(\mathcal{B})$ accepts $\lambda(G)$.*

Cf. Same result in [Salvati & Walukiewicz 13] but they use infinitary λY-calculus.

Remarks on the Proof (3)

$\lambda(G)$ is the tree-unravelling of the underlying (finite) syntax graph of G.

Theorem (Transfer)

Given Σ, there is an effective transformation of APT, α, such that for every HORS G, we have $\mathcal{B}$ accepts tree(G) *iff $\alpha(\mathcal{B})$ accepts $\lambda(G)$.*

Proof Idea. Two key ingredients:

APT $\mathcal{B}$ has accepting run-tree over tree(G)

$\iff$ { **I. Traversal-Path Correspondence**}

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

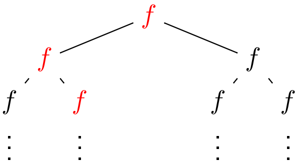
$\iff$ { **II. Simulation of traversals by paths** }

APT $\alpha(\mathcal{B})$ has an accepting run-tree over $\lambda(G)$

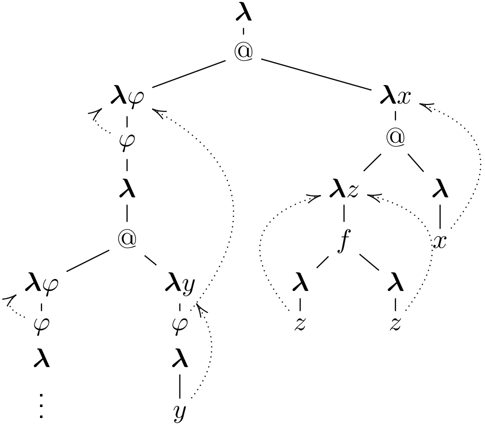
Transference principle, based on a theory of traversals

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

tree(G)



$\lambda(G)$



Idea: β-reduction is global (i.e. substitution changes the term being evaluated); game semantics gives an equivalent but local view.
A traversal (over the computation tree $\lambda(G)$) is a trace of the local computation that produces a path (over $\mathsf{tree}(G)$).

Theorem (Path-traversal correspondence)

Let G be an order-n recursion scheme.

(i) *There is a 1-1 correspondence between maximal paths p in (Σ-labelled) generated tree* $\mathsf{tree}(G)$ *and maximal traversals t_p over computation tree $\lambda(G)$.*

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

Proof is by game semantics.

Explanation (for game semanticists):

- Term-tree $\mathsf{tree}(G)$ is (a representation of) the game semantics of G.
- Paths in $\mathsf{tree}(G)$ correspond to P-views in the strategy-denotation.
- Traversals t_p over computation tree $\lambda(G)$ are just (representations of) the uncoverings of the P-views ($=$ path) p in the game semantics of G.