Verifying Pure Functional Programs

Luke Ong

Oxford University Computing Laboratory

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

(Joint work with Klaus Aehlig and Jolie de Miranda)

Abstract

Recursion schemes are an old model of computation for recursively-defined procedural programs. Higher-order recursive schemes are very natural models of (pure) functional programs. They can viewed as a means of defining a family of (finitely branching) infinite term trees, which forms an infinite hierarchy according to the type-theoretic level.

By building on the famous work of Rabin 1969, and others, Knapik *et al* (in FOSSACS 2002) proved that the monadic second-order theories of all such trees are decidable, provided the generating recursion schemes satisfy a syntactic condition called *safety*. They asked if the safety assumption is necessary for the decidability result. We resolve the question, negatively, for trees at level 2. I.e. trees generated by all level-2 recursion schemes have decidable MSO theories.

In this talk, we survey the area and explain the result.

Motivation

Goal. Foundations of verification (as opposed to inference) of behavioural and structural properties of functional computation.

Examples. Take a typed recursive-defined functional program M.

1. Termination analysis: Does M evaluate to a WHNF? (Of course undecidable in general.)
2. Usage analysis: Does M use a given value only finitely often?
3. Strictness analysis. Does M compute a strict function?

Model-checking paradigm. Fix a class of properties.

- Identify an appropriate model of computation for a class of useful functional computation.
- Study algorithmic properties of the model.

Approach. Model is simple (and reasonably expressive), but problems are hard.

Outline of Talk

1. **Level-2 Recursion Schemes and their Tree Denotations**
2. MSO Logic of Trees
3. Safety and Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. Tree Automata and Equivalence with MSO Logic
5. The Theorem and Proof Outline

Level-2 Recursion Scheme $G = (N, \Sigma, V, \mathcal{R}, S)$

- N: Non-terminals (at most level 2)

$$D : (o \to \cdots \to o) \to \cdots \to o \to \cdots \to o$$

- Σ: Terminals $f : \underbrace{o \to \cdots \to o}_{k} \to o$ (written $o^k \to o$) with $k \geq 0$
- V: Variables $x : o, \varphi : o^k \to o$
- $\mathcal{R}$: Equations for each non-terminal D

$$D\,\varphi_1 \cdots \varphi_m\, x_1 \cdots x_n \quad = \quad e$$

where e is constructed from

  - terminals f, g, a, etc.
  - variables $\varphi_1, \cdots, \varphi_m, x_1, \cdots, x_n$
  - non-terminals D, M, N, etc. from $N - \{ S \}$

- a distinguished *start* non-terminal $S : o$.

Examples

1. A level-1 example:

$F : o \to o$

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

2. A level-2 example.

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

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

$[\![\, G \,]\!]$: Denotation of a Recursion Scheme G

The ***denotation*** $[\![\, G \,]\!]$ of a (deterministic) recursion scheme G is a possibly infinite applicative term constructed from terminals (from Σ), which is obtained by unfolding the equations *ad infinitum*, replacing formal by actual parameters each time, starting from S.

Example. Take

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

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

Regard denotation $[\![\, G \,]\!]$ as a Σ-tree.

The syntax tree of $[\![\, G \,]\!]$ is a Σ-labelled tree, or Σ-tree for short.

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

Recursion scheme *vs* "real functional programs"

How much do recursion schemes fall short?

Recursion schemes are pure, uninterpreted functional programs, minimalist in design.

Certain features can be added "without comprising decidability".

Standard Functional Features	**Present in Recursion Schemes?**
Basic data types	Yes: only finite types
Dynamic data types	No, but can be included if bounded
Basic arithmetics (Presburger)	No, but extendable
Conditionals	No, but extendable
Recursion	Yes
Pattern matching	No. Probably extendable

Outline of Talk

1. Level-2 Recursion Schemes and their Tree Denotations
2. **MSO Logic of Trees**
3. Safety and Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. Tree Automata and Equivalence with MSO Logic
5. The Theorem and Proof Outline

Desiderata

Seek

1. A language $\mathcal{L}$ to describe properties φ
2. A class $\mathcal{T}$ of (finitely presentable) infinite trees

such that

$$\begin{cases} \text{INSTANCE:} & \text{A tree } t \in \mathcal{T}, \text{ and a formula } \varphi \in \mathcal{L} \\ \text{QUESTION:} & \text{Does "} t \vDash \varphi \text{" hold?} \end{cases}$$

should be *decidable*.

Aim. $\mathcal{L}$ should capture as many computationally meaningful properties, and $\mathcal{T}$ should include tree denotations of as many useful functional programs as possible.

Monadic Second-Order Logic (for Σ-trees $t : T \longrightarrow \Sigma$)

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**:

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", for f ranging over Σ
3. Set-membership: $x \in X$

and closed under

- boolean connectives $\neg, \vee, \wedge, \rightarrow$
- first-order $\forall x.-, \exists x.-$ quantifications
- second-order $\forall X.-, \exists X.-$ quantifications.

Why MSO Logic?

In a netshell, there is no "better" candidate around (for describing tree properties)!

- Any obvious extension would yield an undecidable logic.

  Unrestricted interpretation of *binary* relation would permit an encoding of a Turing machine.

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

  Logics with first-order or higher expressivenss can only be decidable if
  - Second-order quantification over binary relations is prohibited
  - No freely interpretable binary relations in the vocabulary.
- MSO is *very* expressive.

  MSO is strictly more expressive than the modal μ-calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL$*$, etc.) are embeddable.

Examples of MSO-definable properties

First, 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. Prefix ordering $x \leq y$ (and hence node equality $x = y$):

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

Example. "There are finitely many occurrences of the terminal $f : o \rightarrow o$."

- "X is a path (in a tree)"

$$\begin{array}{rcl} \mathsf{Path}(X) & \equiv & \forall xy \in X \,.\, x \leq y \vee y \leq x \\ & \wedge & \forall xyz \,.\, x \in X \wedge z \in X \wedge x \leq y \leq z \;\rightarrow\; y \in X \end{array}$$

- $\mathsf{MaxPath}(X) \equiv \mathsf{Path}(X) \wedge \forall Y \,.\, \mathsf{Path}(Y) \wedge X \subseteq Y \;\rightarrow\; Y \subseteq X$.

Example (finite occurrence)

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

$$\begin{array}{rcl} \mathsf{Cut}(X) & \equiv & \forall xy \in X \,.\, \neg(x \leq y \vee y \leq x) \\ & \wedge & \forall Z \,.\, \mathsf{MaxPath}(Z) \;\rightarrow\; \exists z \in Z \,.\, z \in X \end{array}$$

- **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.*

$$\mathsf{Finite}(X) \;\equiv\; \exists Y \,.\, \mathsf{Cut}(Y) \;\wedge\; \forall x \in X \,.\, \exists y \in Y \,.\, x \leq y$$

  Note: By König's Lemma, every cut is finite.

Hence, "there are finitely many nodes labelled by f" is expressible by

$$\exists X \,.\, \mathsf{Finite}(X) \;\wedge\; \forall x \,.\, \mathbf{p}_f(x) \rightarrow x \in X$$

Outline of Talk

1. Level-2 Recursion Schemes and their Tree Denotations
2. MSO Logic of Trees
3. **Safety and Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees**
4. Tree Automata and Equivalence with MSO Logic
5. The Theorem and Proof Outline

Timeline of major decidability results

1. Rabin 1969 "Mother of all decidability results": $S2S$, second-order theory of two successors of infinite binary trees, is decidable.
2. Muller and Schupp 1985: Pushdown graphs have decidable MSO theories.
3. Courcelle 1995: Σ-trees denoed by level-1 recursion schemes have decidable MSO theories.
4. Knapik, Niwiński and Urzyczyn; TLCA 2001: Σ-trees denoted by level-2 safe recursion schemes have decidable MSO theories.
5. KNU, FOSSACS 2002: For all $n \geq 3$, the MSO theories of Σ-trees denoted by level-n safe recursion schemes are decidable.

Hence (combining 1, 3, 4 and 5)

Theorem. Safe trees of all levels have decidable MSO theories.

Knapik-Niwiński-Urzyczyn Hierarchy of *Safe* Trees

For $n \geq 0$, ***level-n safe trees*** are trees denoted by level-n recursion schemes satisfying the *safety* condition. (Safety only "bites" from level 2 onwards.)

- Level 0: Regular trees
- Level 1: Trees generated by deterministic pushdown automaton

Several Characterizations

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 hierarchies of trees $(\mathcal{T}_n : n \geq 0)$, and graphs $(\mathcal{G}_n : n \geq 0)$ (Caucal 2002)
   - $\mathcal{G}_0$ are the Σ-labelled finite graphs; $\mathcal{T}_0$ the Σ-labelled finite trees
   - Trees in $\mathcal{T}_{n+1}$ are the tree-unfoldings of graphs in $\mathcal{G}_n$
   - Graphs in $\mathcal{G}_{n+1}$ are the inverse rational images of trees in $\mathcal{T}_n$.

What is safety?

An awkward syntactic constraint. Semantically unnatural, but has great algorithmic value.

Idea goes back to W. Damm: *derived types* in "IO and OI Hierarchies" TCS 1982.

Definition. A level-2 equation is ***unsafe*** if the RHS contains a level-1 subterm that occurs in an operand position and contains a level-0 parameter.

Examples.

1. Unsafe equation: underlined subterm level-1 is at *operand* position

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

2. Safe equation: underlined subterm level-1 is at *operator* position

$$G\,\varphi\,x\,y \;=\; g\,(\underline{G\,\varphi\,y}\,x)\,(g\,a)$$

What does *safe* mean?

A basic idea in functional programming:

When performing β-reduction, one must use *capture-avoiding* substitution, which is often implemented by *renaming bound variables* afresh upon each substitution.

An algorithmic price to pay:

Any *reduction machine* that correctly computes:

$$\begin{cases} \text{INPUT:} & \text{a (recursively defined) simply-typed } \lambda\text{-term } M \\ \text{OUTPUT:} & \text{a } \beta\text{-reduction sequence from } M \end{cases}$$

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

Safety lets us get away with not renaming bound variables!

Lemma. *When performing β-reductions on a safe λ-term, it is safe not to rename bound variables afresh upon substitution.*

Two natural questions about safety

Is the safety constraint spurious?

1. **Expressiveness**. Are there inherently unsafe Σ-trees?

   I.e. Is there an unsafe recursion scheme whose tree-denotation is not the denotation of any safe recursion scheme? If so, at what level?

   Conjecture. Yes, at level 2. But *cf.* our FOSSACS05 paper

   Theorem. (A+deM+O) *There is no inherently unsafe word language.*

2. **Decidability**. Is safety necessary for decidabiliy?

   No, not at level 2.

The main result of this talk (TLCA 2005):

> **Theorem**. (A+deM+O) *Σ-trees denoted by level-2 recursion schemes have decidable MSO theories.*

What about higher levels? Don't know.

Outline of Talk

1. Level-2 Recursion Schemes and their Tree Denotations
2. MSO Logic of Trees
3. Safety and Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. **Tree Automata and Equivalence with MSO Logic**
5. The Theorem and Proof Outline

Tree Automata $A = (Q, \Sigma, \Delta, q_0, Acc)$

- a finite set Q of control-states, with initial state $q_0 \in Q$
- a finite alphabet Σ comprising symbols $f : o^k \to o$ of arity $k \geq 0$
- transition relation:

$$\Delta \subseteq Q \times \Sigma \times (Q + Q^2 + \cdots + Q^m)$$

  where m is the maximum arity of symbols in Σ, and Δ has elements of the form

$$(q,\ f : o^k \to o,\ (q_1, \cdots, q_k))$$

- $Acc \subseteq Q^\omega$ (for defining acceptance).

We use tree automata A as accepting devices to define tree languages.

Tree language recognised by a tree automaton

A ***run*** of tree automaton A over a Σ-tree $t : T \longrightarrow \Sigma$ is just an assignment of states to nodes of t that respects the transition relation.

Formally it is a function $r : T \longrightarrow Q$ such that $r(\epsilon) = q_0$ and for each $\alpha \in T \subseteq \{\, 1, \cdots, m \,\}^*$, we have

$$(r(\alpha),\ t(\alpha) : o^k \to o,\ (r(\alpha\, 1), \cdots, r(\alpha\, k))) \ \in\ \Delta$$

A run $r : T \longrightarrow Q$ is ***accepting*** if every maximal path $\beta_0\beta_1 \cdots$ (i.e. an element of $\{\, 1, \cdots, m \,\}^\omega$) in T, $r(\beta_0)r(\beta_1) \cdots \in Acc$.

A Σ-tree t is accepted by A just if there is an accepting run of A over t.

Define the ***tree language*** of A, $L(A) = \{\, T \in \Sigma\text{-trees} :\ A \text{ accepts } T \,\}$.

Parity condition: Acc consists of $p_0p_1p_2 \cdots \in Q^\omega$ such that

$$\min\{\, \Omega(q) :\ q \text{ occurs infinitely often in } p_0p_1p_2 \cdots \,\}$$

is even.

MSO logic and tree automata are expressively equivalent

1. There is an algorithm that transforms a MSO formula φ to a tree automaton A_φ such that for all Σ-trees t

$$t \vDash \varphi \iff A_\varphi \text{ accepts } t$$

2. There is an algorithm that transforms a tree automaton A to an MSO formula φ_A such that for all Σ-trees t

$$A \text{ accepts } t \iff t \vDash \varphi_A$$

Recall:

Rabin's Theorem 1969. For any tree automaton, it is decidable if its tree language (i.e. set of Σ-trees accepted by it) is empty.

Outline of Talk

1. Level-2 Recursion Schemes and their Tree Denotations
2. MSO Logic of Trees
3. Safety and Knapik-Niwiński-Urzyczyn Hierarchy of Safe Trees
4. Tree Automata and Equivalence with MSO Logic
5. **The Theorem and Proof Outline**

Safety is not necessary for decidability at level 2. Precisely

Theorem (A+deM+O 2004). *Σ-trees denoted by level-2 recursion schemes have decidable MSO theories.*

(Also obtained by Knapik, Niwinski, Urzyczyn + Walukiewicz, but by a different proof.)

An idea that works for safe trees ...

... but doesn't carry over to the general case.

Level-1 recursion schemes are well understood (recall: safety only bites from level 2 onwards).

Transform any level-2 G to an equivalent level-1 recursion scheme (essentially by partially evaluation).

Example

Take level-2 $G_3 = \left\{ \begin{array}{rcl} S & = & F\,f\,a\,b \\ F\,\varphi\,x\,y & = & F\,(F\,\varphi\,y)\,y\,(\varphi\,x) \end{array} \right.$

Consider

$$G'_3 = \left\{ \begin{array}{rcl} S & = & F\,f\,a\,b \\ F\,\varphi & = & \boldsymbol{\lambda} x\,y.F\,(F\,\varphi\,y)\,y\,(\varphi\,x) \end{array} \right.$$

By regarding φ as a level-0 parameter, and $\boldsymbol{\lambda}xy.$ and x and y as new terminals, G'_3 "becomes" level-1!

Problem: Unfolding and only replace formal parameter φ by actual

$$\begin{array}{rcl} S & = & F\,f\,a\,b \\ & = & \boldsymbol{\lambda}xy.F\,(F\,f\,y)\,y\,(f\,x)\,a\,b \\ & = & \boldsymbol{\lambda}xy.(\boldsymbol{\lambda}x'y'.F\,(F\,(F\,f\,y)\,y')\,y'(F\,f\,y\,x'))\,y\,(f\,x)\,a\,b \end{array}$$

We need y and y' to be distinct to avoid name capture. So need infinitely many distinct names (and hence terminal symbols)!

Doing it precisely!

We transform $G_3 = \left\{ \begin{array}{rcl} S & = & F\,f\,a\,b \\ F\,\varphi\,x\,y & = & F\,(F\,\varphi\,y)\,y\,(\varphi\,x) \end{array} \right.$ to

$$G'_3 = \left\{ \begin{array}{rcl} S & = & @\,(@\,(F\,f)\,a)\,b \\ F\,\varphi & = & \boldsymbol{\lambda}x.\boldsymbol{\lambda}y.@(@\,(F\,(@\,(F\,\varphi)\,y))\,y)(@\,\varphi\,x) \end{array} \right.$$

where $\varphi : o$ and $F : o \to o$ with

$$\left\{ \begin{array}{rcl} @ & : & o \to o \to o \\ \boldsymbol{\lambda}x., \boldsymbol{\lambda}y. & : & o \to o \\ x,\ y & : & o \end{array} \right.$$

are new symbols in the augmented signature Σ'.

Now G'_3 *is* a level-1 recursion scheme. Intuitively G'_3 corresponds to unfolding G_3-rules *ad infinitum*, but only contracting the level-1 redexes.

An idea that does work: Never contract any β-redex

Take a level-2 G. Consider the curried (and η-expanded) versions of the G-rules.

Example Thus from G_3 we obtain:

$$G_3'' = \begin{cases} S & = & F\ f\ a\ b \\ F & = & \boldsymbol{\lambda}\varphi\ x\ y.F\ (\boldsymbol{\lambda}z.F\ (\boldsymbol{\lambda}x'.\varphi x')\ y\ z)\ y\ (\varphi\ x) \end{cases}$$

This is a *level-0* recursion scheme. Call the corresponding syntax tree of $[\![\, G_3'' \,]\!]$ the ***lambda tree*** of G_3.

No danger of name capture, as there are no formal parameters!

Intuitively the lambda tree is obtained by unfolding the rules (and hence replicating the β-redexes), but never contracting any.

We prove a more general result:

> For any level-2 (non-deterministic) recursion scheme G, for any MSO formula φ, it is decidable if $t \vDash \varphi$ for some $t \in [\![\, G \,]\!]$.

Outline of Argument

INPUT: G, φ

1. Define the tree language $\lambda\mathsf{Trees}(G)$ of lambda trees generated from G.
   Note: Every $L \in \lambda\mathsf{Trees}(G)$ "evaluates" (by potentially infinite β-reduction) to some Σ-tree $\mathsf{Eval}(L) \in [\![\, G \,]\!]$.
   $\lambda\mathsf{Trees}(G)$ is regular i.e. recognisable by a tree automaton B_G.
2. From φ construct the tree automaton A_φ that accepts all Σ-trees satisfying φ.
   Simulation of A_φ: Construct a tree automaton C_φ that accepts precisely those lambda trees that "evaluates to" Σ-trees accepted by A_φ i.e.
   $$C_\varphi \text{ accepts } L \iff A_\varphi \text{ accepts } \mathsf{Eval}(L).$$
3. Construct the intersection automaton of B_G and C_φ, and check for non-emptiness.

OUTPUT: Yes iff $t \vDash \varphi$ for some $t \in [\![\, G \,]\!]$.