Model Checking Higher-Order Functional Programs

Luke Ong

Oxford University Computing Laboratory

Joint work with: Naoki Kobayashi (Tohoku);
Martin Lester, Robin Neatherway and Steven Ramsay (Oxford).

Nordic Workshop on Programming Theory
10-12 November 2010, Turku, Finland

Software Model Checking

The development of techniques (notably model checking) for the computer-aided verification of computing systems is a success story in computer science.

2007 ACM Turing Award (Clarke, Emerson and Sifakis) "for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

What is Model Checking?

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

The model checking approach:

1. Find an abstract (usually finite-state) model M of the system *Sys*.
2. Describe the property *Spec* as a formula φ of a suitable logic.
3. Exhaustively check if φ is violated by M.

Software Model Checking

The development of techniques (notably model checking) for the computer-aided verification of computing systems is a success story in computer science.

2007 ACM Turing Award (Clarke, Emerson and Sifakis) "for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

What is Model Checking?

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

The model checking approach:

1. Find an abstract (usually finite-state) model M of the system *Sys*.
2. Describe the property *Spec* as a formula φ of a suitable logic.
3. Exhaustively check if φ is violated by M.

Verification of Functional Programs

Significant advances in the engineering of scalable software model checkers (especially 1st-order imperative programs such as C) in the past decade.

These techniques appear much less useful for higher-order functional programs.

Verifying functional programs: two standard approaches

1. static (especially, type-based) analysis of programs
   - sound, scalable but often imprecise
2. theorem proving using dependent types
   - accurate, but requires human intervention; does not scale well

In this talk, we present an approach to verifying functional programs by reduction to the model checking of higher-order recursion schemes.

Outline

Outline

Recall: Church's Simple Types

Simple Types $A ::= \mathsf{o} \mid (A \to B)$

The **order** of a type is a measure of "nestedness" on LHS of $\to$.

$$\begin{cases} \mathsf{order}(\mathsf{o}) & := & 0 \\ \mathsf{order}(A \to B) & := & \max(\mathsf{order}(A) + 1, \mathsf{order}(B)) \end{cases}$$

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.

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

(Deterministic) recursion schemes are grammars for defining infinite trees.

Example. An order-1 recursion scheme G_1. Fix a ranked alphabet of terminal symbols $\Sigma = \{ f, g, a \}$ (with arities 2, 1 and 0 respectively).

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

Unfolding from the start symbol S:

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

The (term-)tree thus generated, written $[\![\, G_1 \,]\!]$, is

$$f\, a\, (f\, (g\, a)\, (f\, (g\, (g\, a))(\cdots))).$$

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

(Deterministic) recursion schemes are grammars for defining infinite trees.

Example. An order-1 recursion scheme G_1. Fix a ranked alphabet of terminal symbols $\Sigma = \{ f, g, a \}$ (with arities 2, 1 and 0 respectively).

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

Unfolding from the start symbol S:

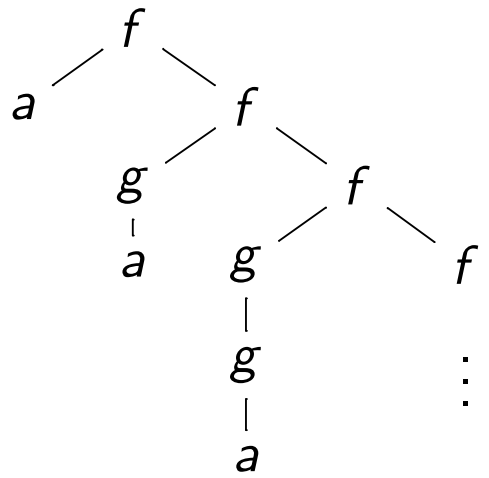
$$\begin{array}{rcl} S & \to & F\, a \\ & \to & f\, a\, (F\, (g\, a)) \\ & \to & f\, a\, (f\, (g\, a)\, (F\, (g\, (g\, a)))) \\ & \to & \cdots \end{array}$$

The (term-)tree thus generated, written $[\![G_1]\!]$, is

$$f\, a\, (f\, (g\, a)\, (f\, (g\, (g\, a))(\cdots))).$$

The tree $[\![G_1]\!]$ generated by recursion scheme G_1

$[\![G_1]\!] = f\ a\ (f\ (g\ a)\ (f\ (g\ (g\ a))(\cdots)))$ is the (term-)tree

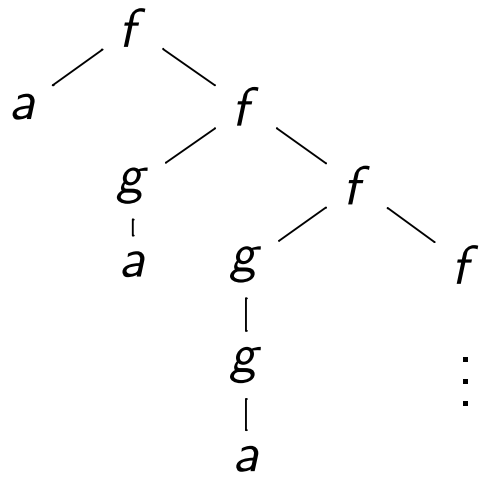


We view the infinite term $[\![G_1]\!]$ as a Σ-labelled tree.

Σ-labelled trees such as $[\![G_1]\!]$ are ranked and ordered.

The tree $[\![G_1]\!]$ generated by recursion scheme G_1

$[\![G_1]\!] = f\ a\ (f\ (g\ a)\ (f\ (g\ (g\ a))(\cdots)))$ is the (term-)tree



We view the infinite term $[\![G_1]\!]$ as a Σ-labelled tree.

Σ-labelled trees such as $[\![G_1]\!]$ are ranked and ordered.

The Hierarchy of Trees generated by Recursion Schemes

For $n \geq 0$, let $\mathbf{RecSchTree}_n$ be the class of Σ-labelled trees generated by order-n recursion schemes.

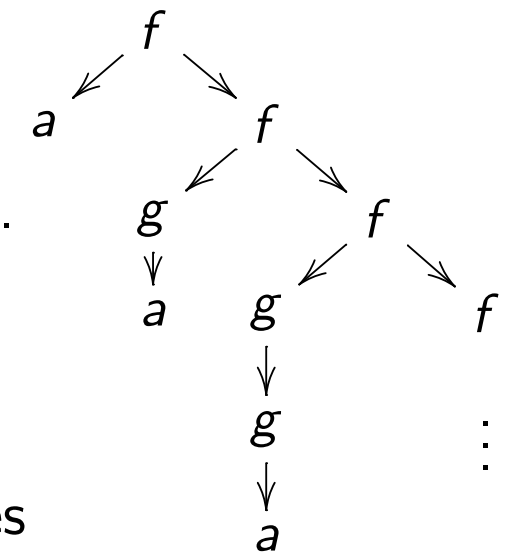
Some Nice Properties

1. Hierarchy Theorem (Damm 1982) for $\langle\, \mathbf{RecSchTree}_n \mid n \in \omega \,\rangle$
2. The hierarchy is highly expressive: order-0 are the regular trees, order-1 are the algebraic trees (Courcelle 1995); order-2 are the hyperalgebraic trees (Knapik et al. 2001).
3. Machine characterization: order-n trees are exactly those generated by order-n collapsible pushdown automata (HMOS LiCS 2008)
4. Decidable MSO theories (to date, the "largest" known such hierarchy of trees). More anon...

A Challenge Problem in Verification

Example: Consider $[\![G_1]\!]$ on the right

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



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

We use monadic second-order logic (MSOL) (equivalently modal mu-calculus) to describe properties such as $\varphi_1 \vee \varphi_2$.

Is the "MSO Model-Checking Problem for $\mathbf{RecSchTree}_n$" decidable?

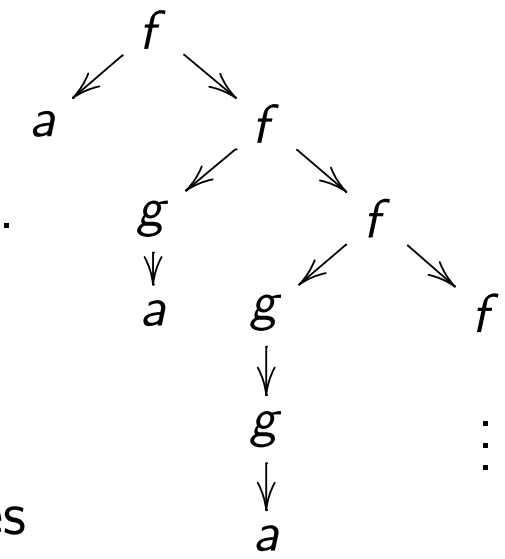
- INSTANCE: An order-n recursion scheme G, and an MSO formula φ
- QUESTION: Does the Σ-labelled tree $[\![G]\!]$ satisfy φ?

(Equivalently: for each order-n G, is the MSO theory of $[\![G]\!]$ decidable?)

A Challenge Problem in Verification

Example: Consider $[\![G_1]\!]$ on the right

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



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

We use monadic second-order logic (MSOL) (equivalently modal mu-calculus) to describe properties such as $\varphi_1 \vee \varphi_2$.

Is the "MSO Model-Checking Problem for **RecSchTree**$_n$" decidable?

- INSTANCE: An order-n recursion scheme G, and an MSO formula φ
- QUESTION: Does the Σ-labelled tree $[\![G]\!]$ satisfy φ?

(Equivalently: for each order-n G, is the MSO theory of $[\![G]\!]$ decidable?)

Why study MSO logic?

MSOL is the gold standard of logics for describing model-checking properties.

- MSOL is *very* expressive.
  - MSOL is more expressive than the modal mu-calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL$*$, etc.) can embed.
  - But note that, over trees, modal mu-calculus is equi-expressive with (but more tractable than) MSOL.
- Hard to do better than MSOL.
  It is hard to extend MSOL meaningfully without sacrificing decidability where it holds.

A (selective) survey of MSO-decidable structures

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs (ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-n pushdown automata. $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-n safe rec. schemes.
- Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

Theorem (O. LiCS 2006)

For $n \geq 0$, trees in $\mathbf{RecSchTree}_n$ have decidable MSO theories.

Proof is by game semantics.

A (selective) survey of MSO-decidable structures

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs (ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-n pushdown automata. $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-n safe rec. schemes.
- Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

Theorem (O. LiCS 2006)

For $n \geq 0$, trees in $\mathbf{RecSchTree}_n$ have decidable MSO theories.

Proof is by game semantics.

A (selective) survey of MSO-decidable structures

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs (ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-n pushdown automata. $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-n safe rec. schemes.
- Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

Theorem (O. LiCS 2006)

For $n \geq 0$, trees in $\mathbf{RecSchTree}_n$ have decidable MSO theories.

Proof is by game semantics.

A (selective) survey of MSO-decidable structures

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs (ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): **PushdownTree**$_n\Sigma$ = Trees generated by order-n pushdown automata. **SafeRecSchTree**$_n\Sigma$ = Trees generated by order-n safe rec. schemes.
- Caucal (MFCS 2002). **CaucalTree**$_n\Sigma$ and **CaucalGraph**$_n\Sigma$.

Theorem (O. LiCS 2006)

For $n \geq 0$, trees in **RecSchTree**$_n$ *have decidable MSO theories.*

Proof is by game semantics.

A (selective) survey of MSO-decidable structures

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs (ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): **PushdownTree**$_n\Sigma$ = Trees generated by order-n pushdown automata. **SafeRecSchTree**$_n\Sigma$ = Trees generated by order-n safe rec. schemes.
- Caucal (MFCS 2002). **CaucalTree**$_n\Sigma$ and **CaucalGraph**$_n\Sigma$.

Theorem (O. LiCS 2006)

For $n \geq 0$, trees in **RecSchTree**$_n$ *have decidable MSO theories.*

Proof is by game semantics.

A (selective) survey of MSO-decidable structures

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs (ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-n pushdown automata. $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-n safe rec. schemes.
- Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

Theorem (O. LiCS 2006)

For $n \geq 0$, trees in $\mathbf{RecSchTree}_n$ have decidable MSO theories.

Proof is by game semantics.

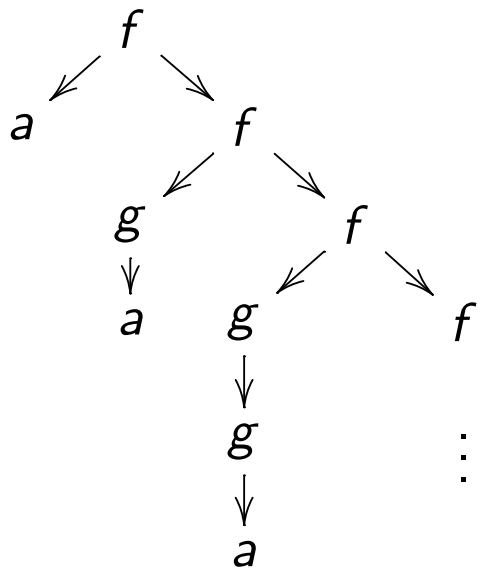
Outline

Model Checking Problem for HORS

Given: order-n recursion scheme G, and alternating parity tree automaton (equivalently, modal mu-calculus formula) $\mathcal{A}$; Qn: Does $\mathcal{A}$ accept $[\![G]\!]$?

Alternating parity tree automata (APT) for infinite trees

$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, \Omega \rangle$



Transition function

$$\begin{aligned}
\delta(q_0, f) &= ((1, q_0) \wedge (2, q_0)) \vee (1, q_1) \\
\delta(q_0, g) &= (1, q_1) \\
\delta(q_1, g) &= (1, q_1) \\
\delta(q_0, a) &= \delta(q_1, a) = \text{true}
\end{aligned}$$

Priority map:

$$\begin{aligned}
\Omega(q_0) &= 0 \\
\Omega(q_1) &= 1
\end{aligned}$$

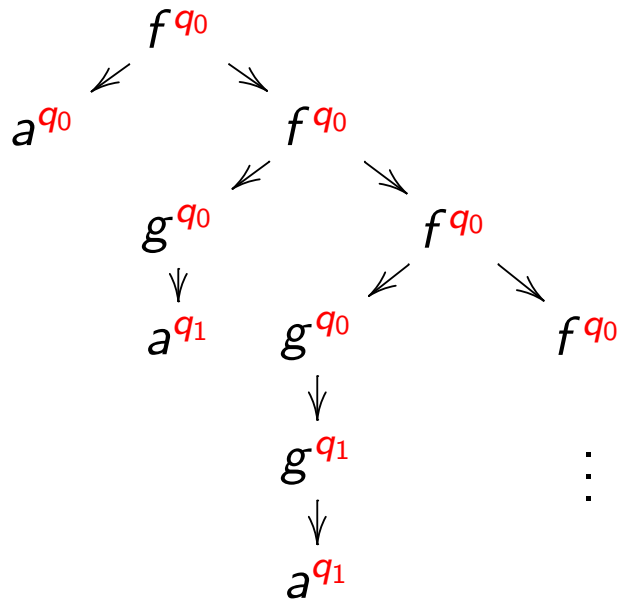
Parity condition: For every infinite path of the run-tree, the least priority visited infinitely often must be even.

Model Checking Problem for HORS

Given: order-n recursion scheme G, and alternating parity tree automaton (equivalently, modal mu-calculus formula) $\mathcal{A}$; Qn: Does $\mathcal{A}$ accept $[\![G]\!]$?

Alternating parity tree automata (APT) for infinite trees

$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, \Omega \rangle$



Transition function

$$\begin{aligned}
\delta(q_0, f) &= ((1, q_0) \wedge (2, q_0)) \vee (1, q_1) \\
\delta(q_0, g) &= (1, q_1) \\
\delta(q_1, g) &= (1, q_1) \\
\delta(q_0, a) &= \delta(q_1, a) = \text{true}
\end{aligned}$$

Priority map:

$$\begin{aligned}
\Omega(q_0) &= 0 \\
\Omega(q_1) &= 1
\end{aligned}$$

Parity condition: For every infinite path of the run-tree, the least priority visited infinitely often must be even.

N.B. $\mathcal{A}$ accepts t iff in every path of t, if g occurs than a eventually occurs.

A type theory characterising APT / modal mu-calculus theories

Theorem (Characterisation. Kobayashi + O. LiCS 2009)

Given an alternating parity tree automaton $\mathcal{A}$ there is a type system $\mathcal{K}_{\mathcal{A}}$ such that for every recursion scheme G, the tree $[\![G]\!]$ is accepted by $\mathcal{A}$ iff G is $\mathcal{K}_{\mathcal{A}}$-typable.

Theorem (Parameterised Complexity. Kobayashi + O. LiCS 2009)

There is a type-inference algorithm polytime in size of recursion scheme, assuming the other parameters are fixed.
The runtime is

$$O(r^{1+\lfloor m/2 \rfloor}\, \mathbf{exp}_n((a\,|Q|\,p)^{1+\epsilon}))$$

where r is the number of rules of the recursion scheme, a is largest arity of the types, p the number of priorities and $|Q|$ the number of states.

Note: $\mathbf{exp}_n(x)$ is tower-of-exponentials function of height n. I.e.

$$\mathbf{exp}_0(x) := x \qquad \mathbf{exp}_{i+1}(x) := 2^{\mathbf{exp}_i(x)}$$

Intersection types embedded with states and priorities

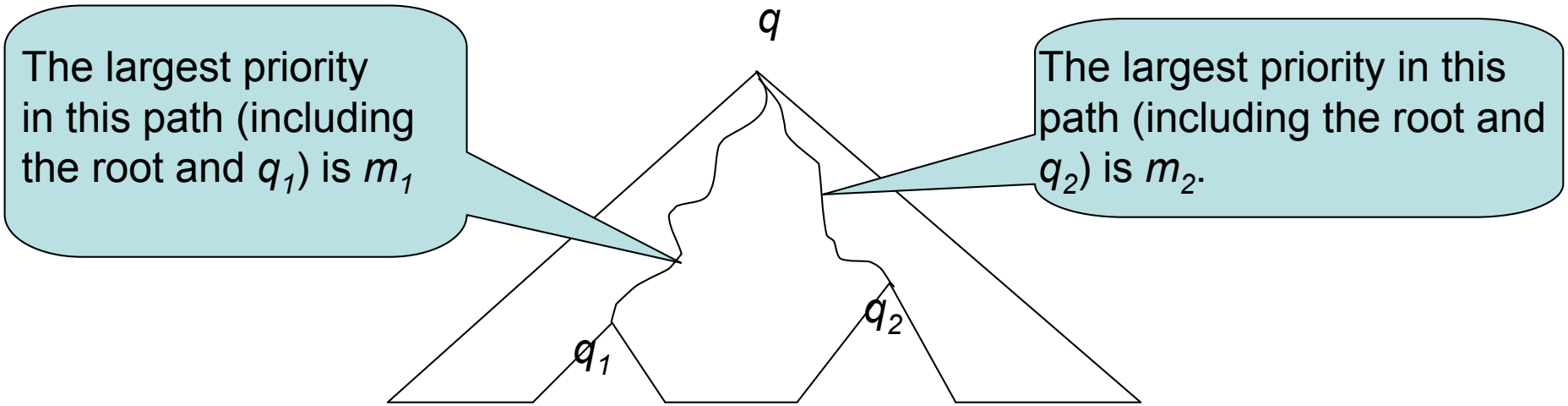
Intersection types: Long history. First used to construct filter models for untyped λ-calculus (Dezani, Barendregt, et al. in early 80s).

Fix an alternating parity tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$.

Idea: Refine intersection types with APT states $q \in Q$ and priorities m_i.

$$\begin{array}{rrcl} \textit{Types} & \theta & ::= & q \mid \tau \to \theta \\ & \tau & ::= & \bigwedge \{ (\theta_1, m_1), \cdots, (\theta_k, m_k) \} \end{array}$$

Intuition. A tree function described by $(q_1, m_1) \wedge (q_2, m_2) \to q$.



Intersection types embedded with states and priorities

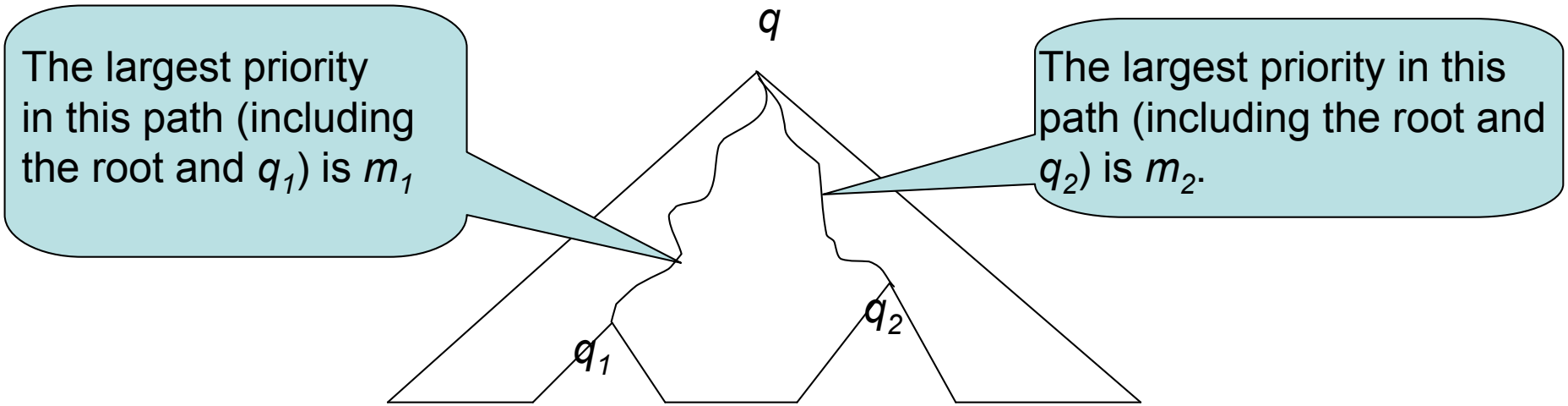
Intersection types: Long history. First used to construct filter models for untyped λ-calculus (Dezani, Barendregt, et al. in early 80s).

Fix an alternating parity tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$.

Idea: Refine intersection types with APT states $q \in Q$ and priorities m_i.

$$\begin{array}{rrcl} \textit{Types} & \theta & ::= & q \mid \tau \to \theta \\ & \tau & ::= & \bigwedge \{ (\theta_1, m_1), \cdots, (\theta_k, m_k) \} \end{array}$$

Intuition. A tree function described by $(q_1, m_1) \wedge (q_2, m_2) \to q$.



Defining Typability of Recursion Scheme G as Parity Game

Typing judgements: $\Gamma \vdash t : \theta$. Only four rules - one per term-constructor.

Definition. G is typable just if Verifier has a winning strategy in a **parity game** determined by the APT $\langle Q, \Sigma, \delta, q_0, \Omega \rangle$ as follows.

Idea: Verifier tries to prove that G is typable; Refuter tries to disprove it.

- Start vertex: $S : (q_0, \Omega(q_0))$.
- Verifier: Given $F : (\theta, m)$, choose Γ such that $\Gamma \vdash rhs(F) : \theta$ is valid.
- Refuter: Given Γ, choose $F : (\theta, m) \in \Gamma$; then challenge Verifier to prove that F has type (θ, m).

Verifier wins just if (i) he does not get stuck, and (ii) if play is infinite, parity condition is satisfied.

Intuition: The game is a way to construct an infinite type derivation, in a form suitable for reasoning about parity.

Defining Typability of Recursion Scheme G as Parity Game

Typing judgements: $\Gamma \vdash t : \theta$. Only four rules - one per term-constructor.

Definition. G is typable just if Verifier has a winning strategy in a **parity game** determined by the APT $\langle Q, \Sigma, \delta, q_0, \Omega \rangle$ as follows.

Idea: Verifier tries to prove that G is typable; Refuter tries to disprove it.

- Start vertex: $S : (q_0, \Omega(q_0))$.
- Verifier: Given $F : (\theta, m)$, choose Γ such that $\Gamma \vdash rhs(F) : \theta$ is valid.
- Refuter: Given Γ, choose $F : (\theta, m) \in \Gamma$; then challenge Verifier to prove that F has type (θ, m).

Verifier wins just if (i) he does not get stuck, and (ii) if play is infinite, parity condition is satisfied.

Intuition: The game is a way to construct an infinite type derivation, in a form suitable for reasoning about parity.

Rules of the Type System $\mathcal{K}_{\mathcal{A}}$

$$\frac{}{x : (\theta, m) \vdash x : \theta} \quad \text{(T-Var)}$$

$$\frac{\{\, (i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i \,\} \text{ satisfies } \delta_{\mathcal{A}}(q, a)}{\varnothing \vdash a : \bigwedge_{j=1}^{k_1}(q_{1j}, m_{1j}) \to \cdots \to \bigwedge_{j=1}^{k_n}(q_{nj}, m_{nj}) \to q} \quad \text{(T-Const)}$$
$$\text{where } m_{ij} = max(\Omega(q_{ij}), \Omega(q))$$

$$\frac{\begin{array}{c}\Gamma_0 \vdash t_0 : (\theta_1, m_1) \wedge \cdots \wedge (\theta_k, m_k) \to \theta \\ \Gamma_i \vdash t_1 : \theta_i \text{ for each } i \in \{\, 1, \ldots, k \,\}\end{array}}{\Gamma_0 \cup (\Gamma_1 \uparrow m_1) \cup \cdots \cup (\Gamma_k \uparrow m_i) \vdash t_0\ t_1 : \theta} \quad \text{(T-App)}$$

where $\Gamma \uparrow m = \{\, F : (\theta, \min(m, m')) \mid F : (\theta, m') \in \Gamma \,\}$

$$\frac{\Gamma, x : \bigwedge_{i \in I}(\theta_i, m_i) \vdash t : \theta \qquad I \subseteq J}{\Gamma \vdash \lambda x.t : \bigwedge_{i \in J}(\theta_i, m_i) \to \theta} \quad \text{(T-Abs)}$$

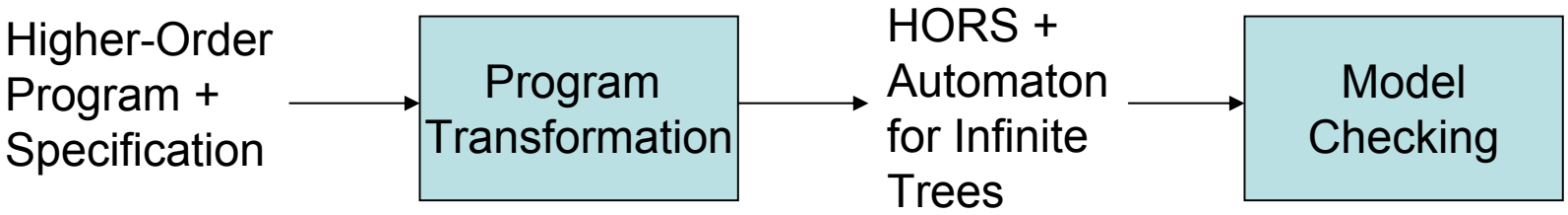
Outline

Verification by Reduction to Model Checking HORS

Verification Problem: Given a functional program P and a temporal specification φ, does P satisty φ?

Our approach:

1. Transform program P to a recursion scheme $\widetilde{P}$ that generates a tree representing all possible event sequences in P.
2. Transform property φ to an automaton of infinite trees $\widetilde{\varphi}$.
3. Model check recursion scheme $\widetilde{P}$ against (transformed) property $\widetilde{\varphi}$, so that $P \models \varphi$ iff $[\![\widetilde{P}]\!]$ is accepted by $\widetilde{\varphi}$.



This method is fully automatic, sound and complete (for Resource Usage Verification Problem).

Resource Usage Verification Problem

Scenario. Higher-order recursive functional programs generated from finite base types, with dynamic resource creation and access primitives.

Resources model stateful objects such as files, locks and memory cells.

Question. Does program D access each resource ρ in accord with φ, where φ is a formula (e.g. LTL formula or regular expression) or an automaton (e.g. alternating parity automaton).

Example. A simple resource specification: $\varphi =$ "An opened file is eventually closed, and after which it is not read". E.g. set $\varphi = \mathtt{r}^* \, \mathtt{c}$.

```
let rec g x = if b then close(x)
                   else read(x) ; g(x) in
let r = open_in "foo" in g(r)
```

Does program access resource `foo` in accord with φ?

Are questions of this kind decidable?

Resource Usage Verification Problem

Scenario. Higher-order recursive functional programs generated from finite base types, with dynamic resource creation and access primitives.

Resources model stateful objects such as files, locks and memory cells.

Question. Does program D access each resource ρ in accord with φ, where φ is a formula (e.g. LTL formula or regular expression) or an automaton (e.g. alternating parity automaton).

Example. A simple resource specification: $\varphi =$ "An opened file is eventually closed, and after which it is not read". E.g. set $\varphi = \mathtt{r}^* \, \mathtt{c}$.

```
let rec g x = if b then close(x)
                   else read(x) ; g(x) in
let r = open_in "foo" in g(r)
```

Does program access resource `foo` in accord with φ?

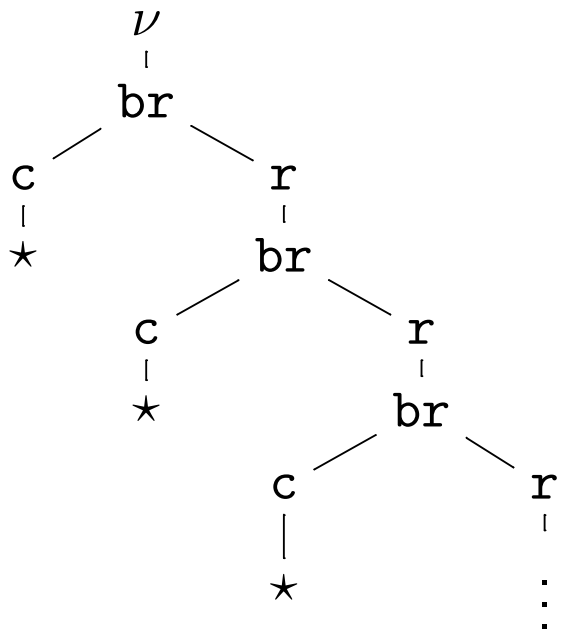
Are questions of this kind decidable?

An approach to verifying Resource Usage (Kobayashi, POPL 2009)

1. Transform ML-like source program (by CPS and λ-lifting) to recursion scheme

$$\begin{cases} S & \to & \nu\,(G\,d\,\star) \\ G\,x\,k & \to & \mathtt{br}\,(c\,k)\,(\mathtt{r}\,(G\,x\,k)) \end{cases}$$

that generates an infinite tree, each of whose path (from root) corresponds to a possible access sequence to resource in question.



2. Reduce resource usage problem to model checking the scheme against a transformed property given by an APT (in this case, a trivial automaton).

3. Further reduce model checking problem to a type inference problem.

Resource Usage Verification Problem

Resource Usage Verification Problem

Instance: A functional program P using resources ($\lambda^{\rightarrow}$ + recursion + booleans + resource creation / access primitives), and specification φ as a parity word automaton.

Question: Does P use resources in accord with φ?

Resource usage properties translate into alternating parity tree automata. Thus we have:

Theorem (Lester, Neatherway, O. + Ramsay 2010; submitted)

The Resource Usage Verification Problem is decidabe (n-EXPTIME complete where n is order of source programs).

Outline

Experiments with Thors (Ramsay, Lester, Neatherway + O. 2010)

Brute-force search will not work!

Order	Types	# Intersection Types (assume 2 states)
1	$o \to o$	$2^2 \times 2 = 8$
2	$(o \to o) \to o$	$2^8 \times 2 = 512$
3	$((o \to o) \to o) \to o$	$2^{512} \times 2 = 2^{513} \approx 10^{154} \gg$ # atoms in univ.!

Thors (Types for Higher-Order Recursion Schemes)

- An implementation of the type-inference algorithm for alternating weak tree automata (equivalently alternation-free mu-calculus). Hence it can check all CTL properties.
- Builds on and extends Kobayashi's "hybrid algorithm".
- Uses partial evaluation, symmetry reduction and many other tricks to drastically reduce search space.

Available at `https://mjolnir.comlab.ox.ac.uk/thors`

Example 1: A network-oriented OCaml program `intercept`

This program reads an arbitrary amount of data from a network socket into a queue and is then responsible for forwarding the data on to another socket.

```
let rec f_o y n= for i in 1 to n do { write(y) } ;
                 done ; close(y)
let rec f_i x y n = if b then read(x) ; f_i(x,y,n+1)
                        else close(x) ; f_o(y,n)
let s_o = open_out "socket_out"
in let s_i = open_in "socket_in"
   in f_i(s_i,s_o,0)
```

An recursion scheme (order 4, 15 rules) is obtained by "slicing" the source program[1] (about 110 LOC of ML), and then CPS-transforming it.
Correctness property: If the "in" socket stops transmitting data then the "out" socket is eventually closed i.e. $AG\,(close_{in} \implies AF\,close_{out})$.
The APT has 2 states.
Execution time: 35 ms. `https://mjolnir.comlab.ox.ac.uk/thors`

[1] `intercept.ml` at ML systems programming repository `http://abaababa.ouvaton.org/caml`.

Example 1: A network-oriented OCaml program `intercept`

This program reads an arbitrary amount of data from a network socket into a queue and is then responsible for forwarding the data on to another socket.

```
let rec f_o y n= for i in 1 to n do { write(y) } ;
                 done ; close(y)
let rec f_i x y n = if b then read(x) ; f_i(x,y,n+1)
                         else close(x) ; f_o(y,n)
let s_o = open_out "socket_out"
in let s_i = open_in "socket_in"
   in f_i(s_i,s_o,0)
```

An recursion scheme (order 4, 15 rules) is obtained by "slicing" the source program[1] (about 110 LOC of ML), and then CPS-transforming it.
Correctness property: If the "in" socket stops transmitting data then the "out" socket is eventually closed i.e. $AG\,(close_{in} \implies AF\,close_{out})$.
The APT has 2 states.
Execution time: 35 ms. `https://mjolnir.comlab.ox.ac.uk/thors`

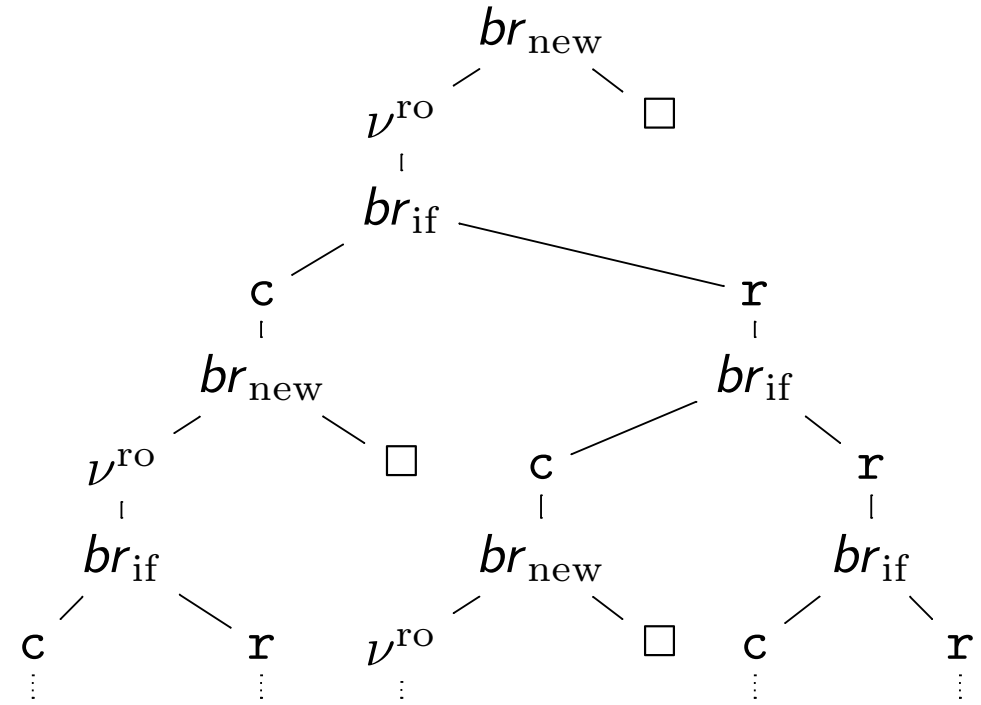
[1] `intercept.ml` at ML systems programming repository `http://abaababa.ouvaton.org/caml`.

Example 2. Liveness with fairness assumption

```
let rec g x = if b then close(x) ;
                        let r' = open_in gensym() in g(r')
                    else read(x) ; g(x)
in let r = open_in gensym()
   in g(r)
```

Say
an access sequence is unfair if, from some point onwards, it only takes the right branch of br_{if} (intuitively because it corresponds to reading an infinite "readonly" resource).
Set φ to be the CTL formula

$$AG\,(\mathtt{r} \Rightarrow A\,((\mathtt{r} \vee br_{\text{if}})\,U\,\mathtt{c})).$$



Restricted to fair paths, the tree satisfies φ.

Example 3: Fibonacci numbers.

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

Using a DWT we can check that they obey the ordering

$$(\mathit{even}\ \mathit{odd}\ \mathit{odd})^{\omega}.$$

Experimental data for AWT model checking

Example	*O*	*R*	*Q*	*Time*	*Nodes*	*Game*	*Result*	*Property*
D1	4	7	2	1	19	16	Y	Det. Weak
D2	4	7	3	1	26	17	Y	Conj. Weak
D2-ex	4	7	3	1	26	-	Y	Alt. Trivial
intercept	4	15	2	35	200	31	Y	Conj. Weak
imperative	3	6	3	129	200	17	Y	Det. Weak
boolean2	2	15	1	1	13	-	Y	Det. Trivial
order5-2	5	9	4	19	200	37	N	Det. Co-trivial
lock1	4	12	3	2	32	32	Y	Det. Co-trivial
order5-v-dwt	5	11	4	163	400	53	Y	Det. Weak
lock2	4	11	4	109	800	-	Y	Det. Trivial
example2-1	1	2	2	190	200	-	Y	Det. Trivial

Time in ms; O (resp. R) = order (resp. # rules) of recursion scheme; Q = # states of automaton; *Game* = # nodes in game graph;

Outline

Verifying Haskell: pattern-matching algebraic data types

Pattern-matching recursion schemes (PMRS) (O. + Ramsay POPL'11)
- Virtually all interesting properties of PMRS are undecidable.

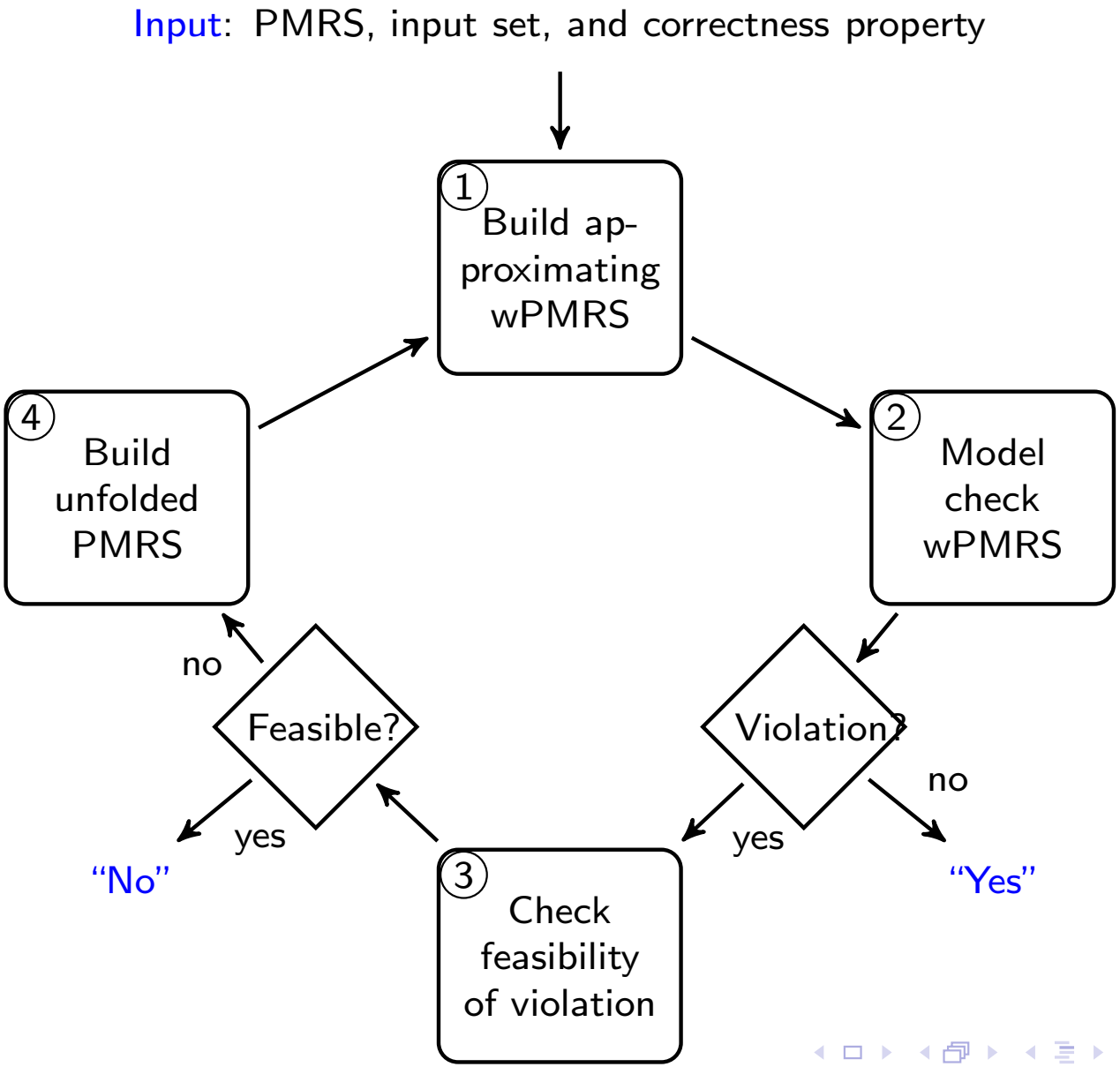
PMRS Verification Problem

Given a correctness property φ, a functional program P (*qua* PMRS) and an input (regular) set I, does every term which is reachable from I under rewriting by P satisfy φ?

Our approach and results:

1. We construct an order-n weak pattern-matching recursion scheme (wPMRS) which over-approximates the set of terms reachable from the input set. This gives the most accurate reachability / flow analysis of its kind.
2. The (trivial automaton) model checking problem for wPMRS is decidable.
3. There is a simple notion of counterexample guided abstraction refinement giving rise to a semi-completeness property: given a no-instance of the Problem, the CEGAR loop will eventually terminate with the answer "No".

Counter-Example Guided Abstraction Refinement (CEGAR) Loop



Conclusions

- Verification of higher-order programs is challenging and worthwhile.
- Recursion schemes are a robust and highly expressive language for infinite structures. They have rich algorithmic properties.
- Recent progress in the theory has been made possible by semantic methods, enabling the extraction of new (but necessarily highly complex) algorithms.
- Verification of functional programs can be reduced to model checking recursion schemes. The approach is automatic, sound and complete.

Challenges:

1. Hybrid algorithm is hyper-exponential on size of recursion scheme. A new game-semantic algorithm is linear, but need empirical confirmation. (Both not scalable on size of tree automata.)
2. Verify thousands of LOC of Haskell (using PMRS and Cegar loop).
3. Full modal mu-calculus model checker.

Conclusions

- Verification of higher-order programs is challenging and worthwhile.
- Recursion schemes are a robust and highly expressive language for infinite structures. They have rich algorithmic properties.
- Recent progress in the theory has been made possible by semantic methods, enabling the extraction of new (but necessarily highly complex) algorithms.
- Verification of functional programs can be reduced to model checking recursion schemes. The approach is automatic, sound and complete.

Challenges:

1. Hybrid algorithm is hyper-exponential on size of recursion scheme. A new game-semantic algorithm is linear, but need empirical confirmation. (Both not scalable on size of tree automata.)
2. Verify thousands of LOC of Haskell (using PMRS and Cegar loop).
3. Full modal mu-calculus model checker.

- O. On model checking trees generated by higher-order recursion schemes. In *Proc. LiCS*, 2006.
- Hague, Murawski, O. + Serre. Recursion schemes and collapsible pushdown automata. In *Proc. LiCS*, 2008.
- Carayol, Hague, Meyer, O. + Serre. Winning regions of higher-order pushdown games. In *Proc. LiCS*, 2008.
- Broadbent + O. On global model checking trees generated by higher-order recursion schemes. In *Proc. FoSSaCS*, 2009.
- Kobayashi + O. A type theory equivalent to the model checking of higher-order recursion schemes. In *Proc. LiCS*, 2009.
- O. + Tzevelekos. Functional Reachability. In *Proc. LiCS*, 2009.
- Kobayashi + O. Complexity of model-checking recursion schemes for fragments of the modal mu-calculus. In *Proc. ICALP*, 2009.
- Broadbent, Carayol, O. + Serre. Recursion Schemes and Logical Reflection. In *Proc. LICS*, 2010.
- O + Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *Proc. POPL*, 2011.