Model Checking Higher-Order Computation: II

Luke Ong

Computing Laboratory, University of Oxford

Marktoberdorf Summer School, 4-15 August 2009

Outline of Part 2

1. A Typing System Characterising MSO Theories of Recursion Schemes
   - Preliminaries: Mu-Calculus, APT and Parity Games
   - An Intersection Type System
   - Two Relatively Cheap Fragments

2. Application: A New Approach to Verifying Functional Programs
   - Verification by Reduction to Model Checking Recursion Schemes
   - Resource Usage Problem: A Case Study
   - Experimentation: Preliminary Results and Demo:

Lecture slides and references will be viewable on my homepage
users.comlab.ox.ac.uk/luke.ong

Outline of Part 2

1. A Typing System Characterising MSO Theories of Recursion Schemes
   - Preliminaries: Mu-Calculus, APT and Parity Games
   - An Intersection Type System
   - Two Relatively Cheap Fragments

2. Application: A New Approach to Verifying Functional Programs
   - Verification by Reduction to Model Checking Recursion Schemes
   - Resource Usage Problem: A Case Study
   - Experimentation: Preliminary Results and Demo:

Lecture slides and references will be viewable on my homepage
`users.comlab.ox.ac.uk/luke.ong`

Outline

1. A Typing System Characterising MSO Theories of Recursion Schemes
   - Preliminaries: Mu-Calculus, APT and Parity Games
   - An Intersection Type System
   - Two Relatively Cheap Fragments
2. Application: A New Approach to Verifying Functional Programs
   - Verification by Reduction to Model Checking Recursion Schemes
   - Resource Usage Problem: A Case Study
   - Experimentation: Preliminary Results and Demo:

Theorem (Equivalence, Emerson+Jutla 91)

Let $\mathcal{A}, \varphi, T$ range over APT, modal mu-formulas, ranked trees resp.

1. *For each $\mathcal{A}$, there exists φ such that $\mathcal{A}$ accepts T iff T satisfies φ.*
2. *For each φ, there exists $\mathcal{A}$ such that $\mathcal{A}$ accepts T iff T satisfies φ.*

Positive Boolean formulas over X: $\mathrm{B}^+(X) \ni \theta \ ::= \ \mathsf{t} \mid \mathsf{f} \mid x \mid \theta \wedge \theta \mid \theta \vee \theta$
$Y \subseteq X$ *satisfies* θ just if assigning true to elements in Y and false to others makes θ true.

An APT over Σ-labelled trees is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$ where

- Σ is a ranked alphabet; m is the largest arity of terminals
- $q_I \in Q$ is the initial state
- $\delta : Q \times \Sigma \longrightarrow \mathrm{B}^+(\{1, \ldots, m\} \times Q)$ is the transition function
- $\Omega : Q \longrightarrow \{0, \cdots, M-1\}$ is the priority function.

Theorem (Equivalence, Emerson+Jutla 91)

Let $\mathcal{A}, \varphi, T$ range over APT, modal mu-formulas, ranked trees resp.

1. *For each $\mathcal{A}$, there exists φ such that $\mathcal{A}$ accepts T iff T satisfies φ.*
2. *For each φ, there exists $\mathcal{A}$ such that $\mathcal{A}$ accepts T iff T satisfies φ.*

Positive Boolean formulas over X: $\mathrm{B}^{+}(X) \ni \theta \ ::= \ \mathsf{t} \mid \mathsf{f} \mid x \mid \theta \wedge \theta \mid \theta \vee \theta$
$Y \subseteq X$ *satisfies* θ just if assigning true to elements in Y and false to others makes θ true.

An APT over Σ-labelled trees is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$ where

- Σ is a ranked alphabet; m is the largest arity of terminals
- $q_I \in Q$ is the initial state
- $\delta : Q \times \Sigma \longrightarrow \mathrm{B}^{+}(\{1, \ldots, m\} \times Q)$ is the transition function
- $\Omega : Q \longrightarrow \{0, \cdots, M-1\}$ is the priority function.

Run-tree of an APT

A run-tree of an APT is just a set of maximal state-annotated paths in the tree that respect the transition relation.

A tree is accepted by an APT just if there is a run-tree such that every infinite path π in it satisfies the parity condition.

Let $\pi = \pi_1 \, \pi_2 \, \cdots$ be an infinite path in r; for each $i \geq 0$, let the state label of the node $\pi_1 \cdots \pi_i$ be q_{n_i} where q_{n_0}, the state label of ϵ, is q_I. We say that

π satisfies the *parity* condition

The largest priority that occurs infinitely often in $\Omega(q_{n_0}) \, \Omega(q_{n_1}) \, \Omega(q_{n_2}) \cdots$ is even.

Example

Let $\Sigma = \{ a : 2, b : 1, c : 0 \}$.

Let $\mathcal{A}$ be the APT $(\Sigma, \{ q_0, q_1 \}, \delta, q_0, \Omega)$, where (let $q \in \{ q_0, q_1 \}$)

$$
\begin{array}{lcl}
\delta & : & \begin{cases} (q, a) \mapsto (1, q_1) \wedge (2, q) \\ (q, b) \mapsto (1, q) \\ (q, c) \mapsto \mathrm{true} \end{cases} \\
\Omega & : & \begin{cases} q_0 \mapsto 2 \\ q_1 \mapsto 1 \end{cases}
\end{array}
$$

$\mathcal{A}$ accepts a Σ-tree t just if for every path of t, if the path ever takes the left branch of a node labeled by a, then the path contains c.

For a tree rejected by $\mathcal{A}$, consider the full binary tree with nodes labelled by a.

Parity game

A parity game is a tuple $(V_R, V_V, v_0, E, \Omega)$ such that

- $E \subseteq V \times V$ is the edge relation of a directed graph whose node-set $V := V_R + V_V$; $v_0 \in V$ is the start node
- $\Omega : V \longrightarrow \{ 0, \cdots, M-1 \}$ assigns a priority to each node.

Playing a parity game

A play consists in the players, R (Refuter) and V (Verifier), taking turns to move a token along the edges of the graph. At a given stage of the play, suppose the token is on an R-node v (respectively V-node), then R (respectively V) chooses an edge (v, v') and moves the token onto v'. At the start of a play, the token is placed on v_0.

Thus a play is a finite or infinite path $\pi = v_0 \, v_{n_1} \, v_{n_2} \cdots$ in the graph that starts from v_0.

A parity game is a tuple $(V_R, V_V, v_0, E, \Omega)$ such that

- $E \subseteq V \times V$ is the edge relation of a directed graph whose node-set $V := V_R + V_V$; $v_0 \in V$ is the start node
- $\Omega : V \longrightarrow \{\, 0, \cdots, M-1 \,\}$ assigns a priority to each node.

Playing a parity game

A play consists in the players, R (Refuter) and V (Verifier), taking turns to move a token along the edges of the graph. At a given stage of the play, suppose the token is on an R-node v (respectively V-node), then R (respectively V) chooses an edge (v, v') and moves the token onto v'. At the start of a play, the token is placed on v_0.

Thus a play is a finite or infinite path $\pi = v_0\, v_{n_1}\, v_{n_2} \cdots$ in the graph that starts from v_0.

Who wins a maximal play?

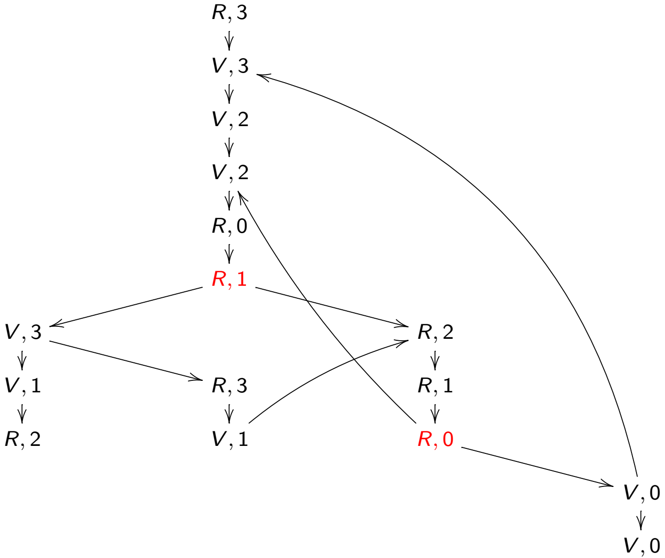
Let π be a maximal play.

- If π is finite, and ends in a V-node (respectively R-node), then R (respectively V) wins.
- If π is infinite, V wins iff π satisfies the parity condition.

Definitions

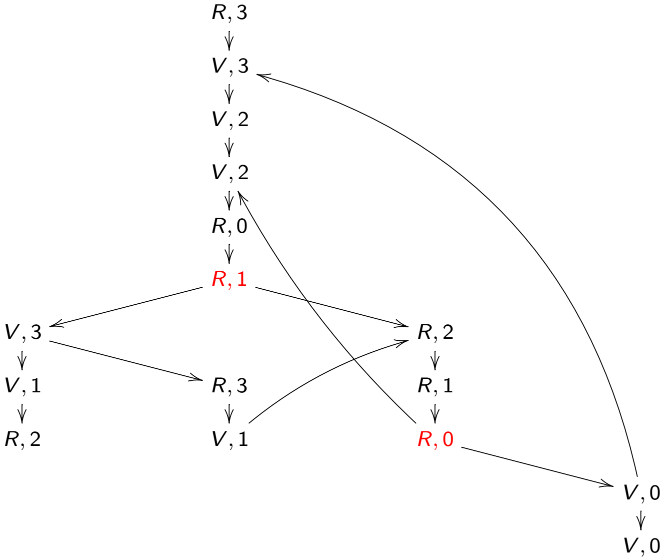
A V-strategy $\mathcal{W}$ is a map from plays ending in a V-node to a node extending the play.
$\mathcal{W}$ is winning if V wins every (maximal) play π that conforms with the strategy.

An Example: Who has a winning strategy?



R always chooses the right child - winning strategy.

An Example: Who has a winning strategy?



R always chooses the right child - winning strategy.

"Parity Game $\equiv$ Modal Mu-Calculus"

A Game Reading of the Fundamental Semantic Theorem

Theorem (Emerson+Street 89)

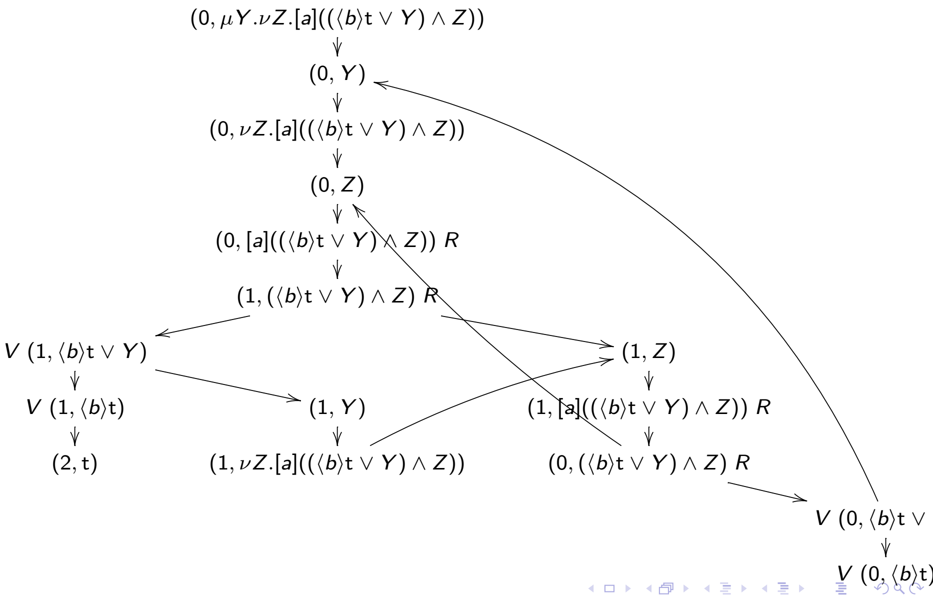
Given a labelled transition system $\mathcal{L}$, a start state s_0, and a modal mu-formula φ, there is a parity game $G(\mathcal{L}, s_0, \varphi)$ such that $\mathcal{L}, s_0 \models \varphi$ iff Verifier has a winning strategy for $G(\mathcal{L}, s_0, \varphi)$.

Example $G(\mathcal{L}, 0, \mu Y.\nu Z.[a]((\langle b \rangle \mathrm{t} \vee Y) \wedge Z))$ where

$$\mathcal{L} \; = \; \begin{tikzcd} 0 \arrow[r, bend left, "a"] & 1 \arrow[l, bend left, "a"] \arrow[r, "b"] & 2 \end{tikzcd}$$

(For the game graph, see next slide.)

Game graph of $\mathcal{G}(T, \mu Y.\nu Z.[a]((\langle b \rangle \mathrm{t} \vee Y) \wedge Z)$



A typing system characterising MSO / modal mu-calculus theories

Theorem (Characterisation. Kobayashi + O. LiCS 2009)

Given a property φ (APT / mu-calculus) there is a typing system $\mathcal{K}_\varphi$ such that for every recursion scheme G, the tree $[\![G]\!]$ satisfies φ iff G is $\mathcal{K}_\varphi$-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

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

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

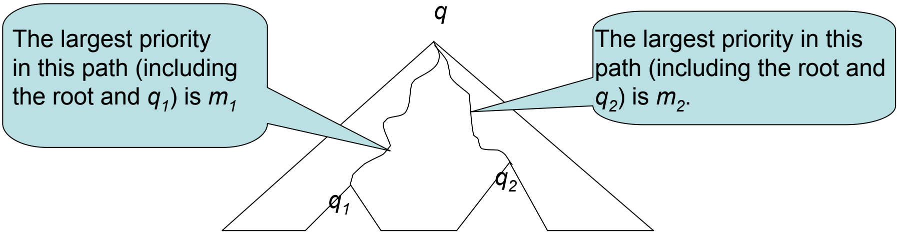
Intersection types: Long history. First used to construct filter models for untyped λ-calculus (Dezani, Barendregt, et al. early 80s).

Fix an APT $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$.

Idea: Refine intersection types with APT states q and priorities m_i of APT.

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

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



Intersection types embedded with states and priorities of an APT

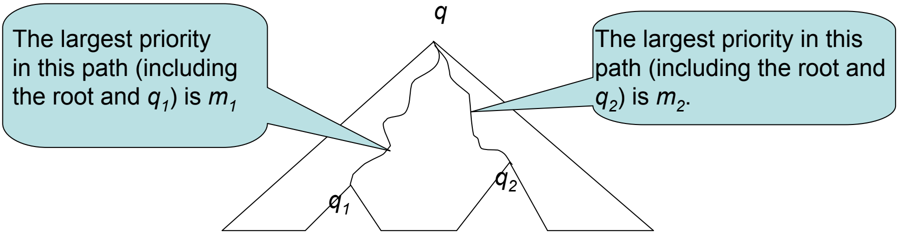
Intersection types: Long history. First used to construct filter models for untyped λ-calculus (Dezani, Barendregt, et al. early 80s).

Fix an APT $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$.

Idea: Refine intersection types with APT states q and priorities m_i of APT.

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

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



Typing judgement

$$\Gamma \vdash t : \theta$$

where the environment Γ is a finite set of bindings $x : (\theta, m)^b$ with $b \in \{\, \mathrm{t}, \mathrm{f} \,\}$.

- $x : (\theta, m)^{\mathrm{t}} \in \Gamma$ means x can be used only before visiting a state with priority larger than m.
- $x : (\theta, m)^{\mathrm{f}} \in \Gamma$ means it is additionally required that x can be used after visiting a state with priority m.

E.g. Suppose $\Omega(q) = 0$. Then $\{\, x : (q, 1)^{\mathrm{t}} \,\} \vdash x : q$ is valid.

Type-checking infinite trees with parity condition

Typing rules are simple: only four rules - one per term-constructor.

Definition of typability. We say that G is typable just if Verifier has a winning strategy in a **parity game** determined by the APT (Q, δ, q_I, Ω).

Intuition of the parity game: A way to construct an infinite tree of type derivations, suitable for parity condition reasoning.
Underlying graph is bipartite; two kinds of vertices "$F : (\theta, m)$" and "Γ".
Verifier tries to prove that scheme is typable; Refuter tries to disprove it.

Start vertex: $S : (q_I, \Omega(q_I))$.

Verifier: Given $F : (\theta, m)$, choose Γ such that $\Gamma \vdash rhs(F) : \theta$ is valid.

Refuter: Given Γ, choose $F : (\theta, m) \in \Gamma$ (and ask Verifier to prove why F has type θ).

Proof "Standard" methods (e.g. type soundness via type preservation) apply, except reasoning about priorities, which is novel and of independent interest.

Type-checking infinite trees with parity condition

Typing rules are simple: only four rules - one per term-constructor.

Definition of typability. We say that G is typable just if Verifier has a winning strategy in a **parity game** determined by the APT (Q, δ, q_I, Ω).

Intuition of the parity game: A way to construct an infinite tree of type derivations, suitable for parity condition reasoning.
Underlying graph is bipartite; two kinds of vertices "$F : (\theta, m)$" and "Γ".
Verifier tries to prove that scheme is typable; Refuter tries to disprove it.

Start vertex: $S : (q_I, \Omega(q_I))$.

Verifier: Given $F : (\theta, m)$, choose Γ such that $\Gamma \vdash \mathit{rhs}(F) : \theta$ is valid.

Refuter: Given Γ, choose $F : (\theta, m) \in \Gamma$ (and ask Verifier to prove why F has type θ).

Proof "Standard" methods (e.g. type soundness via type preservation) apply, except reasoning about priorities, which is novel and of independent interest.

Type-checking infinite trees with parity condition

Typing rules are simple: only four rules - one per term-constructor.

Definition of typability. We say that G is typable just if Verifier has a winning strategy in a **parity game** determined by the APT (Q, δ, q_I, Ω).

Intuition of the parity game: A way to construct an infinite tree of type derivations, suitable for parity condition reasoning.
Underlying graph is bipartite; two kinds of vertices "$F : (\theta, m)$" and "Γ".
Verifier tries to prove that scheme is typable; Refuter tries to disprove it.

Start vertex: $S : (q_I, \Omega(q_I))$.

Verifier: Given $F : (\theta, m)$, choose Γ such that $\Gamma \vdash \mathit{rhs}(F) : \theta$ is valid.

Refuter: Given Γ, choose $F : (\theta, m) \in \Gamma$ (and ask Verifier to prove why F has type θ).

Proof "Standard" methods (e.g. type soundness via type preservation) apply, except reasoning about priorities, which is novel and of independent interest.

$$\frac{(\theta, m)^b \uparrow \Omega(\theta) = (\theta, m)^{\mathtt{t}}}{x : (\theta, m)^b \vdash x : \theta} \qquad \text{(T-Var)}$$

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

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

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

Trivial APT are APT with a single priority of 0. [Aehlig, LMCS 2007]
Trivial acceptance condition: A tree is accepted just if there is a run-tree (i.e. state-annotation of nodes respecting the transition relation).
Equi-expressive with the "safety fragment" of mu-calculus:

$$\varphi, \psi \ ::= \ P_f \mid Z \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle i \rangle \varphi \mid \nu Z.\varphi.$$

But surprisingly

Theorem (Kobayashi + O., ICALP 2009)

The Trivial APT Acceptance Problem for order-n recursion schemes is still n-EXPTIME complete.

[n-EXPTIME hardness by reduction from word acceptance problem of order-n alternating PDA which is n-EXPTIME complete [Engelfriet 91].]

Safety Fragment of Mu-Calculus / Trivial APT

Trivial APT are APT with a single priority of 0. [Aehlig, LMCS 2007]
Trivial acceptance condition: A tree is accepted just if there is a run-tree (i.e. state-annotation of nodes respecting the transition relation).
Equi-expressive with the "safety fragment" of mu-calculus:

$$\varphi, \psi \ ::= \ P_f \mid Z \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle i \rangle \varphi \mid \nu Z.\varphi.$$

But surprisingly

Theorem (Kobayashi + O., ICALP 2009)

The Trivial APT Acceptance Problem for order-n recursion schemes is still n-EXPTIME complete.

[n-EXPTIME hardness by reduction from word acceptance problem of order-n alternating PDA which is n-EXPTIME complete [Engelfriet 91].]

Trivial APT are APT with a single priority of 0. [Aehlig, LMCS 2007]
Trivial acceptance condition: A tree is accepted just if there is a run-tree (i.e. state-annotation of nodes respecting the transition relation).
Equi-expressive with the "safety fragment" of mu-calculus:

$$\varphi, \psi \ ::= \ P_f \mid Z \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle i \rangle \varphi \mid \nu Z . \varphi.$$

But surprisingly

Theorem (Kobayashi + O., ICALP 2009)

The Trivial APT Acceptance Problem for order-n recursion schemes is still n-EXPTIME complete.

[n-EXPTIME hardness by reduction from word acceptance problem of order-n alternating PDA which is n-EXPTIME complete [Engelfriet 91].]

Disjunctive APT are APT whose transition function maps each state-symbol pair to a purely disjunctive positive boolean formula.

Disjunctive APT capture path / linear-time properties; equi-expressive with "disjunctive fragment" of mu-calculus:

$$\varphi, \psi \ ::= \ P_f \wedge \varphi \mid Z \mid \varphi \vee \psi \mid \langle i \rangle \varphi \mid \nu Z.\varphi \mid \mu Z.\varphi$$

Theorem (Kobayashi + O., ICALP 2009)

The Disjunctive APT Acceptance Problem for order-n recursion schemes is $(n-1)$-EXPTIME complete.

$(n-1)$-EXPTIME decidable: For order-1 APT-types $\bigwedge S_1 \to \cdots \to \bigwedge S_k \to q$, we may assume at most one S_i's is nonempty (and is singleton). Hence only $k \times |Q|^2 \times m$ many such types (N.B. exponential for general APT).

$(n-1)$-EXPTIME hardness: by reduction from emptiness problem of order-n deterministic PDA [Engelfriet 91].

Disjunctive APT are APT whose transition function maps each state-symbol pair to a purely disjunctive positive boolean formula.

Disjunctive APT capture path / linear-time properties; equi-expressive with "disjunctive fragment" of mu-calculus:

$$\varphi, \psi \ ::= \ P_f \wedge \varphi \mid Z \mid \varphi \vee \psi \mid \langle i \rangle \varphi \mid \nu Z.\varphi \mid \mu Z.\varphi$$

Theorem (Kobayashi + O., ICALP 2009)

The Disjunctive APT Acceptance Problem for order-n recursion schemes is $(n-1)$-EXPTIME complete.

$(n-1)$-EXPTIME decidable: For order-1 APT-types $\bigwedge S_1 \to \cdots \to \bigwedge S_k \to q$, we may assume at most one S_i's is nonempty (and is singleton). Hence only $k \times |Q|^2 \times m$ many such types (N.B. exponential for general APT).

$(n-1)$-EXPTIME hardness: by reduction from emptiness problem of order-n deterministic PDA [Engelfriet 91].

Disjunctive Fragment of Mu-Calculus / Disjunctive APT

Disjunctive APT are APT whose transition function maps each state-symbol pair to a purely disjunctive positive boolean formula.

Disjunctive APT capture path / linear-time properties; equi-expressive with "disjunctive fragment" of mu-calculus:

$$\varphi, \psi \ ::= \ P_f \wedge \varphi \mid Z \mid \varphi \vee \psi \mid \langle i \rangle \varphi \mid \nu Z.\varphi \mid \mu Z.\varphi$$

Theorem (Kobayashi + O., ICALP 2009)

The Disjunctive APT Acceptance Problem for order-n recursion schemes is $(n-1)$-EXPTIME complete.

$(n-1)$-EXPTIME decidable: For order-1 APT-types $\bigwedge S_1 \to \cdots \to \bigwedge S_k \to q$, we may assume at most one S_i's is nonempty (and is singleton). Hence only $k \times |Q|^2 \times m$ many such types (N.B. exponential for general APT).

$(n-1)$-EXPTIME hardness: by reduction from emptiness problem of order-n deterministic PDA [Engelfriet 91].

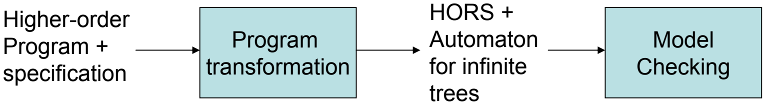
Why study trivial and disjunctive APT?

Corollary

The following problems are $(n-1)$-EXPTIME complete: assume G is an order-n recursion scheme

1. *Reachability: "Does $[\![\, G \,]\!]$ have a node labelled by a given symbol?"*
2. *LTL Model-Checking: "Does every path in $[\![\, G \,]\!]$ satisfy a given φ?"*
3. *Resource Usage Problem*

Outline



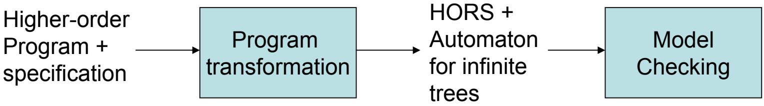
Verification Problem: "Does P satisfy φ?"

- The functional program P is transformed to a recursion scheme $\widetilde{P}$ that generates a tree representing all possible event sequences in P.
- $[\![\, \widetilde{P} \,]\!]$ is then model checked against (transformed) property $\widetilde{\varphi}$, so that $P \vDash \varphi$ iff $[\![\, \widetilde{P} \,]\!] \vDash \widetilde{\varphi}$.

This method is fully automatic, sound and complete.

Program Classes	Models of Computation
imperative programs + iteration	finite-state automata
imperative programs + recursion	PDA / boolean programs
order-n functional programs	order-n recursion schemes

Verification by reduction to model checking recursion schemes



Verification Problem: "Does P satisfy φ?"

- The functional program P is transformed to a recursion scheme $\widetilde{P}$ that generates a tree representing all possible event sequences in P.
- $[\![\, \widetilde{P} \,]\!]$ is then model checked against (transformed) property $\widetilde{\varphi}$, so that $P \vDash \varphi$ iff $[\![\, \widetilde{P} \,]\!] \vDash \widetilde{\varphi}$.

This method is fully automatic, sound and complete.

Program Classes	Models of Computation
imperative programs + iteration	finite-state automata
imperative programs + recursion	PDA / boolean programs
order-n functional programs	order-n recursion schemes

Scenario. Higher-order (recursive) functional programs generated from booleans with dynamic resource creation and access primitives.

Question. Does program P access each resource ρ according to the given resource specification ρ^L, where L is a regular language over the alphabet of resource access primitives.

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

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

Does the program access the resource `foo` in accord with L?

Resource Usage Problem (Igarashi-Kobayashi, POPL 2002)

Scenario. Higher-order (recursive) functional programs generated from booleans with dynamic resource creation and access primitives.

Question. Does program P access each resource ρ according to the given resource specification ρ^L, where L is a regular language over the alphabet of resource access primitives.

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

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

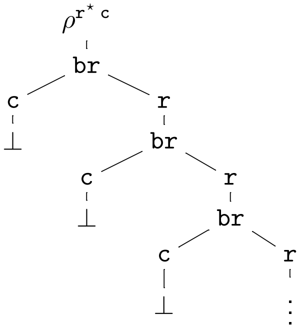
Does the program access the resource `foo` in accord with L?

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

1. Transform source program to rec. scheme

$$\begin{cases} S & \rightarrow & \rho^{\mathtt{r}^* \mathtt{c}}\,(G\,d\,\perp) \\ G\,x\,k & \rightarrow & \mathtt{br}\,(c\,k)\,(\mathtt{r}\,(G\,x\,k)) \end{cases}$$

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



2. Reduce resource usage problem to model checking the scheme against a transformed property given by a trivial automaton.

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

Resource Usage Verification Problem

Instance: A functional program P using resources ($\lambda^{\rightarrow}$ + recursion + booleans + resource creation / access primitives), and specification φ (regular expression).
Question: Does P use resources in accord with φ?

Theorem (Kobayashi + O., ICALP 2009)

For an order-n source program, the Resource Usage Problem is $(n-1)$-EXPTIME complete.

Many verification problems reducible to Resource Usage Problem

- **Program Reachability**: "Given a program (closed term of ground type), does its computation reach a special construct `fail`?"
- Assertion-based verification problems; safety properties
- **Flow Analysis**: "Given a program and its subterms s and t, does the value of s flow to the value of t?"

An interesting exception!

What is reachability in higher-order functional programs?

Contextual Reachability

"Given a term P and its (coloured) subterm N^α, is there a program context $C[\,]$ such that evaluating $C[P]$ cause control to flow to N^α?"

Many versions of the problem. Connexions with Stirling's dependency tree automata.

(See O. + Tzevelekos, "Functional Reachability", In *Proc. LiCS*, 2009).

Many verification problems reducible to Resource Usage Problem

- **Program Reachability**: "Given a program (closed term of ground type), does its computation reach a special construct `fail`?"
- Assertion-based verification problems; safety properties
- **Flow Analysis**: "Given a program and its subterms s and t, does the value of s flow to the value of t?"

An interesting exception!

What is reachability in higher-order functional programs?

Contextual Reachability

"Given a term P and its (coloured) subterm N^α, is there a program context $C[\,]$ such that evaluating $C[P]$ cause control to flow to N^α?"

Many versions of the problem. Connexions with Stirling's dependency tree automata.

(See O. + Tzevelekos, "Functional Reachability", In *Proc. LiCS*, 2009).

Classes of comparatively tractable model checking problems

Two useful fragments of the modal mu-calculus / APT:

(1) Trivial APT ("Safety Fragment"): APT with a singleton priority of 0.

(2) Disjunctive APT: APT whose transition function maps each state to a positive boolean formula that is purely disjunctive.

Theorem (Kobayashi + O., ICALP 2009)

1. *The Trivial APT Acceptance Problem for order-n recursion schemes is still n-EXPTIME complete.*
2. *The Disjunctive APT Acceptance Problem for order-n recursion schemes is $(n-1)$-EXPTIME complete.*

Useful Corollary: The following problems (for order-n schemes) are $(n-1)$-EXPTIME complete:

1. Resource Usage Problem
2. Reachability: "Does $[\![G]\!]$ have a node labelled by a given symbol?"

Two useful fragments of the modal mu-calculus / APT:

(1) Trivial APT ("Safety Fragment"): APT with a singleton priority of 0.

(2) Disjunctive APT: APT whose transition function maps each state to a positive boolean formula that is purely disjunctive.

Theorem (Kobayashi + O., ICALP 2009)

1. *The Trivial APT Acceptance Problem for order-n recursion schemes is still n-EXPTIME complete.*
2. *The Disjunctive APT Acceptance Problem for order-n recursion schemes is $(n-1)$-EXPTIME complete.*

Useful Corollary: The following problems (for order-n schemes) are $(n-1)$-EXPTIME complete:

1. Resource Usage Problem
2. Reachability: "Does $[\![G]\!]$ have a node labelled by a given symbol?"

Classes of comparatively tractable model checking problems

Two useful fragments of the modal mu-calculus / APT:

(1) Trivial APT ("Safety Fragment"): APT with a singleton priority of 0.

(2) Disjunctive APT: APT whose transition function maps each state to a positive boolean formula that is purely disjunctive.

Theorem (Kobayashi + O., ICALP 2009)

1. *The Trivial APT Acceptance Problem for order-n recursion schemes is still n-EXPTIME complete.*
2. *The Disjunctive APT Acceptance Problem for order-n recursion schemes is $(n-1)$-EXPTIME complete.*

Useful Corollary: The following problems (for order-n schemes) are $(n-1)$-EXPTIME complete:

1. Resource Usage Problem
2. Reachability: "Does $[\![G]\!]$ have a node labelled by a given symbol?"

Preliminary experiments with TRecS (Kobayashi, PPDP 09)

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	$((o \to o) \to o) \to o$	$2^{512} \times 2 = 2^{513} \approx 10^{154}$

Example. `amscomp/compileenv.ml` (40 loc) in OCaml compiler 3.11.0

```
let read_sect() =
  let fp = open "foo" in
  {readc = fun x -> read fp;
   closec = fun x -> close fp}
let main() =
  let s = read_sect() in s.readc() ;
  s.closec()
```

Result: An order-4 recursion scheme is obtained after "slicing" the source program and CPS transform; # rules = 23, # APT states = 4. Thanks to ingenious optimisation techniques, time to infer types = ? msec.

Demo. `http://www.kb.ecei.tohoku.ac.jp/~koba/trecs/`

Preliminary experiments with TRecS (Kobayashi, PPDP 09)

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	$((o \to o) \to o) \to o$	$2^{512} \times 2 = 2^{513} \approx 10^{154}$

Example. `amscomp/compileenv.ml` (40 loc) in OCaml compiler 3.11.0

```
let read_sect() =
  let fp = open "foo" in
  {readc = fun x -> read fp;
   closec = fun x -> close fp}
let main() =
  let s = read_sect() in s.readc() ;
  s.closec()
```

Result: An order-4 recursion scheme is obtained after "slicing" the source program and CPS transform; # rules = 23, # APT states = 4. Thanks to ingenious optimisation techniques, time to infer types = ? msec.

Demo. `http://www.kb.ecei.tohoku.ac.jp/~koba/trecs/`

Preliminary experiments with TRecS (Kobayashi, PPDP 09)

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	$((o \to o) \to o) \to o$	$2^{512} \times 2 = 2^{513} \approx 10^{154}$

Example. `amscomp/compileenv.ml` (40 loc) in OCaml compiler 3.11.0

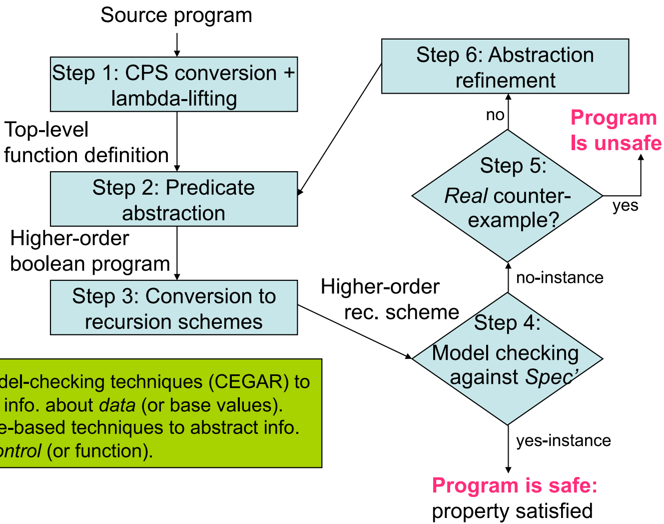
```
let read_sect() =
  let fp = open "foo" in
  {readc = fun x -> read fp;
   closec = fun x -> close fp}
let main() =
  let s = read_sect() in s.readc() ;
  s.closec()
```

Result: An order-4 recursion scheme is obtained after "slicing" the source program and CPS transform; # rules = 23, # APT states = 4. Thanks to ingenious optimisation techniques, time to infer types = ? msec.

Demo. `http://www.kb.ecei.tohoku.ac.jp/~koba/trecs/`

An abstract model checking framework (Kobayashi, POPL 2009)

Input: (i) Functional program with ground-type values (e.g. `int`), and dynamic resource creation and access. (ii) Access specification *Spec*.



- Use model-checking techniques (CEGAR) to abstract info. about *data* (or base values).
- Use type-based techniques to abstract info. about *control* (or function).

References

- O. On model checking trees generated by higher-order recursion schemes. In *Proc. LiCS*, 2006.
- O. Verification of higher-order computation: a game-semantic approach (Invited ETAPS Unifying Lecture). In *Proc. ESOP*, 2008.
- 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.

Conclusions

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

Further directions:

1. Is safety a genuine constraint on expressiveness? Equivalently, are order-n CPDA more expressive than order-n PDA?
2. Extend verification techniques to call-by-value, polymorphism, pattern matching and recursive data types.
3. Major case study: Develop a fully-fledged model checker for Haskell / OCaml.

Conclusions

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

Further directions:

1. Is safety a genuine constraint on expressiveness? Equivalently, are order-n CPDA more expressive than order-n PDA?
2. Extend verification techniques to call-by-value, polymorphism, pattern matching and recursive data types.
3. Major case study: Develop a fully-fledged model checker for Haskell / OCaml.