Higher-Order Model Checking

III: Reducing Model Checking to Type Inference
IV: Applications: Verifying Higher-order Functional Programs

Luke Ong

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

Estonia Winter School in Computer Science, 3-8 Mar 2013

Some Background

Rabin (1969) answered Büchi's question, and developed a theory of automata on infinite trees.

Theorem (Rabin 1969)

A tree language over Σ is MSO-definable iff it is recognisable by a parity (Muller) tree automaton.

Over trees, MSO logic and modal mu-calculus are equi-expressive.

Equi-expressivity (Emerson + Jutla 1991)

For defining tree languages, the following are equi-expressive (in appropriate sense):

1. alternating parity tree automata
2. parity games
3. modal mu-calculus

A type system characterising MSO / modal mu-calculus theories

Theorem (Characterisation. Kobayashi + O. LiCS 2009)

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

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

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

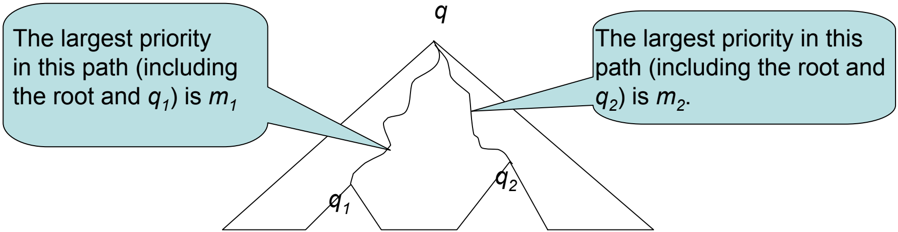
Intersection types embedded with states and priorities

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

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

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



Typing judgement $\Gamma \vdash t : \theta$

Typing judgements are of the shape

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

where the environment Γ is a finite set of variable bindings of the form $x : (\theta, m)$, with θ ranging over types, and m over priorities.

Idea: $\Gamma \vdash s : \theta$

If $x : (q, m) \in \Gamma$, then the largest priority seen in the path (of the value tree) from the current tree node to the node where x is used is exactly m.

Validity of the judgements are defined by induction over four rules.

Rules of the Type System $\mathcal{K}_{\mathcal{A}}$ where APT $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$

$$\frac{}{x : (\theta, \Omega(\theta)) \vdash x : \theta} \qquad \text{(T-VAR)}$$

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

where $m_{ij} = max(\Omega(q_{ij}), \Omega(q))$

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

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

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

Type-Checking Recursion Scheme G w.r.t. $\mathcal{K}_{\mathcal{A}}$

Definition

G is typable just if Verifier has a winning strategy in a **parity game**, parameterised by the APT $\mathcal{A} = \langle\, Q, \delta, q_I, \Omega \,\rangle$, defined (informally) as follows:

Finite bipartite game graph: two kinds of nodes "$F : (\theta, m)$" and "Γ". Verifier tries to prove that G is typable; Refuter tries to disprove it.

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

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

How to decide "Given $\mathcal{A}$ and G, does APT $\mathcal{A}$ accept $[\![G]\!]$?"

Fix $\mathcal{A} = \langle Q, \delta, q_I, \Omega \rangle$ and G. The type inference algorithm has two phases:

Step 1: Construct the parity game associated with the type system $\mathcal{K}_{\mathcal{A}}$.

Finite, bipartite game graph: Verifier nodes are bindings $F : (\theta, m)$; Refuter nodes are environments Γ.

- For each Γ, and each binding "$F : (\theta, m)$" in Γ, there is an edge $\Gamma \longrightarrow F : (\theta, m)$.
- For each "$F : (\theta, m)$", and each Γ such that $\Gamma \vdash rhs(F) : \theta$ is provable, there is an edge $F : (\theta, m) \longrightarrow \Gamma$.

Step 2: Decide whether there is a winning strategy for Verifier for the parity game.

Theorem (Characterisation. Kobayashi + O. LiCS 2009)

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

Remark on proof.

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

Four different proofs of the decidability result

1. Game semantics and traversals (O. LiCS 2006)
   variable profiles
2. Collapsible pushdown automata (HMOS LiCS 2008)
   equi-expressivity theorem + rank aware automata
3. Type theory (KO LiCS 2009)
   intersection types
4. Krivine machine (Salvati + Walukiewicz ICALP 2011)
   residuals

A common thread

1. Decision problem equivalent to solving an infinite parity game.
2. Simulate the infinite game by a finite parity game.
3. The "control states" of the finite game are variable profiles / intersection types / residuals, which are strikingly similar.

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].)

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].

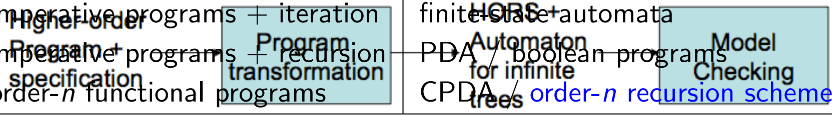
Why study trivial and disjunctive APT?

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*

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



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. linear-time or branching-time temporal formula) or an automaton (e.g. alternating parity automaton).

Example. A simple resource specification: φ = "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?

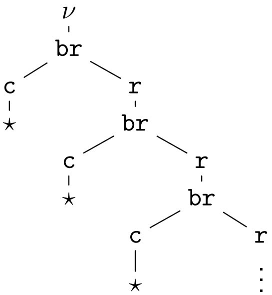
Are questions of this kind decidable?

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

1. Transform source program (by CPS and lambda-lifting) to rec. scheme

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

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

For an order-n source program, the Resource Usage Verification Problem is n-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).

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). So can deal with CTL properties.
- Builds on and extends Kobayashi's TRECS ("hybrid algorithm").
- Uses partial evaluation and symmetry reduction to drastically reduce search space.

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

This program[1] 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 g y n = for i in 1 to n
                              do write(y) ; done ; close(y)
let rec f x y n = if b then read(x) ; f(x,y,n+1)
                         else close(x) ; g(y,n)
let t = open_out "socket2" in
let s = open_in "socket1" in f(s,t,0)
```

An order-4 recursion scheme is obtained after "slicing" the source program and CPS transform; # rules = 15, # APT states = 2.

Correctness property: If the "in" socket stops transmitting data then the "out" socket is eventually closed i.e. $AG\,(close_{in} \Rightarrow AF\; close_{out})$.

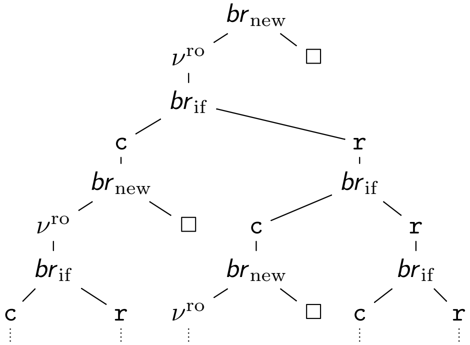
[1] obtained by "slicing" `intercept.ml` (about 110 LOC) at `http://abaababa.ouvaton.org/caml`.

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.

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

Using a DWT we can check that they obey the ordering

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

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;

Verifying (nearly) all of Haskell: pattern-matching alg. data types

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

Verification Problem

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

Our algorithm constructs an order-n weak pattern-matching recursion scheme which over-approximates the set of terms reachable from the input set—giving the most accurate reachability / flow analysis of its kind.

Further, the (trivial automaton) model checking problem for wPMRS is decidable.

Finally, there is a simple notion of automatic abstraction-refinement giving rise to a semi-completeness property.

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.
- Broadbent, Carayol, O. + Serre. Recursion schemes and logical refection. In *Proc. LiCS 2010*.
- S. Ramsay + O. Verification of higher-order functional programs with pattern matching ADT. In *Proc. POPL 2011*.

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.

Further directions:

1. Is safety a genuine constraint on expressiveness? Equivalently, are order-n CPDA more expressive than order-n PDA for generating trees?
2. Major case study: Develop a fully-fledged model checker for Haskell / OCaml.