Recursion Schemes, Games, and Model Checking of Higher-Order Computation

Luke Ong

Oxford University Computing Laboratory

Ecole de Printemps d'Informatique Théorique, 23-27 May 2011

Model checking and computer-aided verification

Beginning in the 80s, computer-aided verification (notably model checking) of finite-state systems (e.g. hardware and communication protocols) has been a great success story in computer science.

Clarke, Emerson and Sifakis won the 2007 ACM Turing Award

"for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

Focus of past decade: transfer of these techniques to software verification.

What is (software) model checking?

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

The model checking approach:

1. Find an abstract 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.

Huge strides made in **verification of 1st-order imperative programs**.

Many tools: SLAM, Blast, Terminator, SatAbs, etc.

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

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

Verification of higher-order programs

Examples: OCaml, F#, Haskell, Lisp/Scheme, JavaScript, and Erlang; even C++.
By comparison with 1st-order imperative program, the model checking of higher-order programs is in its infancy. Some theoretical advances in recent years; very little tool development.

Model-checking higher-order programs is hard

1. Infinite-state and extremely complex: Even without recursion, higher-order programs over a finite base type are infinite-state.

   Many other sources of infinity: data structures and manipulation, control structures (with recursion), asynchronous communication, real-time and embedded systems, systems with parameters etc.

2. Models of higher-order features as studied in semantics – are typically too "abstract" to support any algorithmic analysis.

   A notable exception is game semantics.

Verifying higher-order programs: a worthwhile challenge

1. **Widely used in diverse domains**. Succinct, less error-prone, easy to write and hence good for prototyping; performance (of e.g. F#) approaching *C++*.

Traditional applications: theorem proving and reasoning assistance, computational linguistics, programming language processing.

More recently: databases, networking, internet search (Google's MapReduce), trading and investment banking.
See Wadler's page "Functional Programming in the Real World"[1]

2. **Many hard theoretical problems**: E.g. termination analysis, higher-order matching, and (contextual) reachability analysis.

Our goal: To use semantic methods, in conjunction with algorithmic ideas and techniques from Verification, to formally analyze programming situations in which higher-order features are important.

[1] `http://homepages.inf.ed.ac.uk/wadler/realworld/`

Verification of Functional Programs

Model checking techniques which have worked so well for first-order imperative programs like C are much less useful for higher-order functional programs.

Verifying functional programs: two standard approaches

1. Type-based program analysis
   - sound, scalable but often imprecise
2. Theorem proving and dependent types
   - accurate, typically requires human intervention; does not scale well

Aims of the lecture course

1. We introduce a systematic approach to the algorithmics of infinite structures generated by families of higher-order generators.
2. We present an approach to verifying higher-order functional programs by reduction to the model checking of recursion schemes.

Outline I

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

Order-2 pushdown automata

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

Operations on 2-stacks: s_i ranges over 1-stacks. Top of stack is at the right.

$$
\begin{array}{rcll}
push_2 & : & [s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i}] & \mapsto \quad [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1}\, [a_1 \cdots a_n]] & \mapsto \quad [s_1 \cdots s_{i-1}] \\
push_1\, a & : & [s_1 \cdots s_{i-1}\, [a_1 \cdots a_n]] & \mapsto \quad [s_1 \cdots s_{i-1}\, [a_1 \cdots a_n\, a]] \\
pop_1 & : & [s_1 \cdots s_{i-1}\, [a_1 \cdots a_n\, a_{n+1}]] & \mapsto \quad [s_1 \cdots s_{i-1}\, [a_1 \cdots a_n]]
\end{array}
$$

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

HOPDA as recognizers of word languages

HOPDA can be used as recognizing/generating device for

1. finite-word languages (Maslov 74) (and ω-word languages)

$$\langle \Sigma, Q, q_0, \Gamma, \Delta \subseteq (\Sigma \cup \{\epsilon\}) \times Q \times \Gamma \times Op_n \times Q, F \rangle$$

2. possibly-infinite (ranked) trees (KNU01), and tree languages
3. possibly infinite graphs (Muller+Schupp 86, Courcelle 95, Cachat 03)

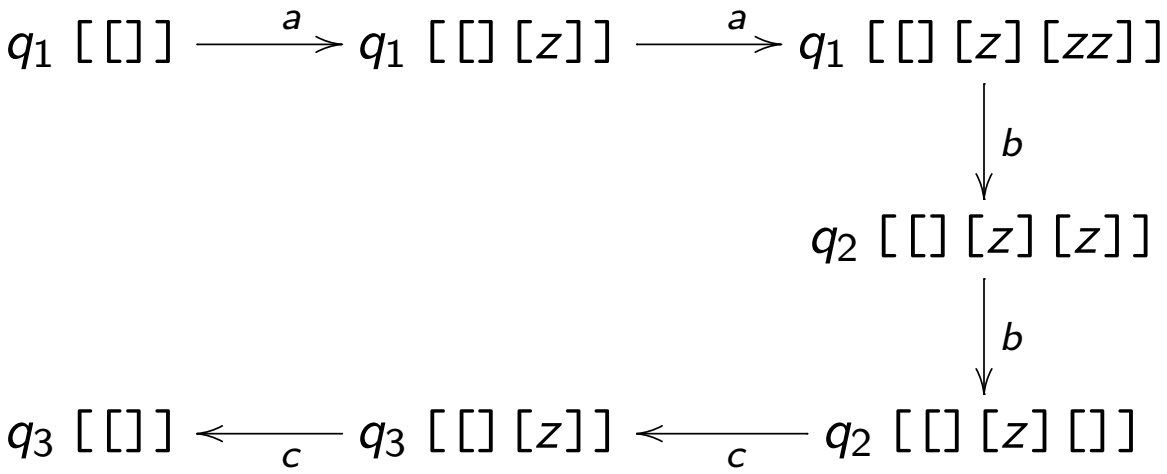
Some basic facts (Maslov 74, 76):

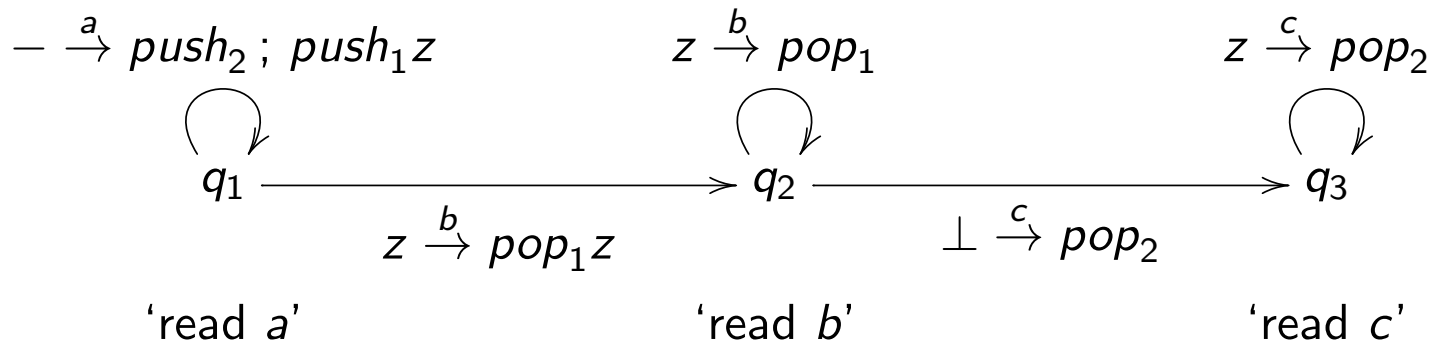
1. HOPDA define an infinite hierarchy of word languages.
2. Low orders are well-known: orders 0, 1 and 2 are the regular, context free, and indexed languages (Aho 68). Higher-order languages are poorly understood.
3. For each $n \geq 0$, the order-n languages form an abstract family of languages (closed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

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

L is not context free. Use the "*uvwxy* Lemma".

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





Pumping Lemma for Context-Free Languages

Theorem (*uvwxy*)

Let L be an infinite CFL. Every word in L longer then p can be written as a concatenation of subwords, $u\,v\,w\,x\,y$, such that $|v\,w\,x| \leq p$, $|v\,x| \geq 1$, and for every $i \geq 0$, $u\,v^i\,w\,x^i\,y$ is in L.

A reminder: simple types

Types $A \quad ::= \quad \mathsf{o} \quad | \quad (A \to B)$

Every type can be written uniquely as

$$A_1 \to (A_2 \cdots \to (A_n \to \mathsf{o}) \cdots), \quad n \geq 0$$

often abbreviated to $A_1 \to A_2 \cdots \to A_n \to \mathsf{o}$.

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

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

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

Notation. $e : A$ means "expression e has type A".

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

An order-n recursion scheme = closed ground-type term definable in order-n fragment of simply-typed λ-calculus with recursion and uninterpreted order-1 constant symbols.

Example: An order-1 recursion scheme. Fix ranked alphabet $\Sigma = \{ f : 2, g : 1, a : 0 \}$.

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

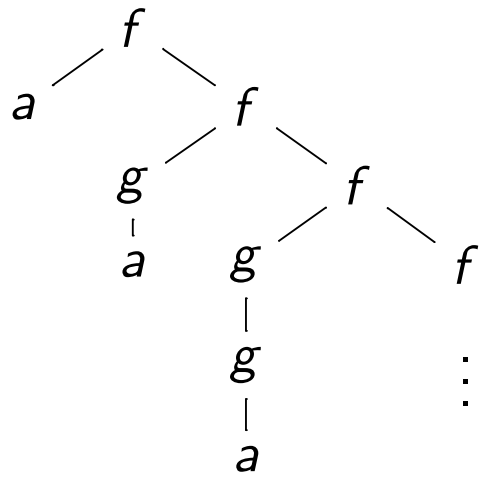
Unfolding from the start symbol S:

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

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

Representing the term-tree $[\![G]\!]$ as a Σ-labelled tree

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



We view the infinite term $[\![G]\!]$ as a Σ-labelled tree, formally, a map $T \longrightarrow \Sigma$, where T is a prefix-closed subset of Dir^*, with Dir a set of edge labels.

Formally term-trees such as $[\![G]\!]$ are ranked and ordered.

Definition: Order-n (deterministic) recursion scheme $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$

Fix a set of typed variables (written as φ, x, y etc).

- $\mathcal{N}$: Typed non-terminals of order at most n (written as upper-case letters), including a distinguished start symbol $S : o$.
- Σ: Ranked alphabet of terminals: $f \in \Sigma$ has arity $\mathrm{ar}(f) \geq 0$
- $\mathcal{R}$: An equation for each non-terminal $D : A_1 \to \cdots \to A_m \to o$ of shape

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

where the term $e : o$ is constructed from

  - terminals f, g, a, etc. from Σ
  - variables $\varphi_1 : A_1, \cdots, \varphi_m : A_m$ from Var,
  - non-terminals D, F, G, etc. from $\mathcal{N}$.

using the application rule: If $s : A \to B$ and $t : A$ then $(s\,t) : B$.

The tree generated by a recursion scheme: value tree

Given a term t, define a (finite) tree $t^\perp$ by

$$t^\perp := \begin{cases} f & \text{if } t \text{ is a terminal } f \\ t_1^\perp\, t_2^\perp & \text{if } t = t_1\, t_2 \text{ and } t_1^\perp \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

We extend the flat partial order on Σ (i.e. $\perp \leq a$ for all $a \in \Sigma$) to trees by:

$$s \leq t \;:=\; \forall \alpha \in \mathit{dom}(s)\,.\, \alpha \in \mathit{dom}(t) \wedge s(\alpha) \leq t(\alpha)$$

E.g. $\perp \leq f\perp\perp \leq f\perp b \leq fab$.

For a directed set T of trees, we write $\bigsqcup T$ for the lub of T w.r.t. $\leq$.

Let G be a recursion scheme. We define the tree generated by G by

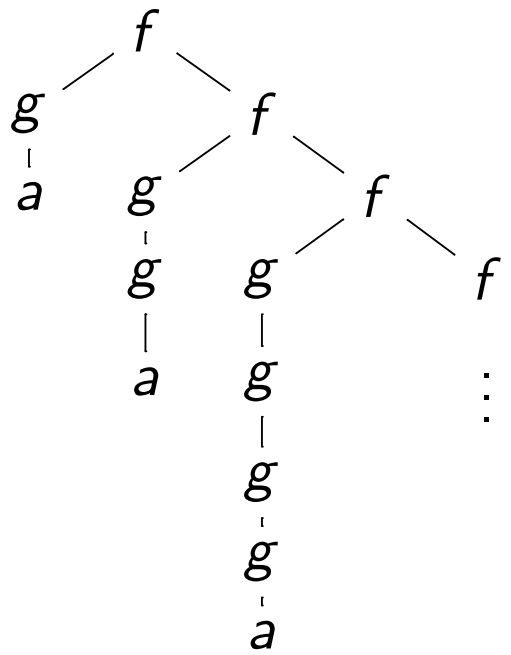
$$[\![\, G \,]\!] \;:=\; \bigsqcup \{\, t^\perp \mid S \to^* t \,\}$$

An order-2 example

$\Sigma = \{ f : 2, g : 1, a : 0 \}$.
$S : o, \quad B : (o \to o) \to (o \to o) \to o \to o, \quad F : (o \to o) \to o$

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

The generated tree, $[\![G_2]\!] : \{ 1, 2 \}^* \longrightarrow \Sigma$, is:



An Order-3 Example: Fibonacci Numbers

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

Non-terminals: Write Ch as a shorthand for $(o \to o) \to o \to o$

$$\begin{array}{lcl} S & : & o \\ Z & : & Ch \\ U & : & Ch \\ F & : & Ch \to Ch \to o \\ P & : & Ch \to Ch \to (o \to o) \to o \to o \end{array}$$

$$\texttt{fib} \left\{ \begin{array}{rcl} S & = & F\ Z\ U \\ Z\ \varphi\ x & = & x \\ U\ \varphi\ x & = & \varphi\ x \\ F\ n_1\ n_2 & = & c\ (n_1\ b\ a)\ (F\ n_2\ (P\ n_1\ n_2)) \\ P\ n_1\ n_2\ \varphi\ x & = & n_1\ \varphi\ (n_2\ \varphi\ x) \end{array} \right.$$

Using recursion schemes as generators of word languages

Idea: A word is just a linear tree.

Represent a finite word "$a\,b\,c$" (say) as the applicative term $a\,(b\,(c\,e))$, viewing a, b and c as symbols of arity 1, where e is the arity-0 end-of-word marker.

Fix an input alphabet Σ. We can use a (non-deterministic) recursion scheme to generate finite-word languages, with ranked alphabet $\overline{\Sigma} := \{\, a : 1 \mid a \in \Sigma \,\} \cup \{\, e : 0 \,\}$.

Example. $\{\, a^n\, b^n \mid n \geq 0 \,\}$ is generated by order-1 recursion scheme:

$$\begin{cases} S & \to & F\,e \\ F\,x & \to & a\,(F\,(b\,x)) \quad \mid \quad x \end{cases}$$

Exercises

1. Find an order-2 (word-language) recursion scheme that generates $L = \{ a^i b^i c^i \mid i \geq 0 \}$.
2. Prove that context-free languages are equivalent to languages generated by order-1 (word-language) recursion schemes.

Answer to 1.

$$\begin{cases} S & \to & F\, I\, e \\ F\, \varphi\, x & \to & \varphi\, x \;\mid\; F\, (H\, \varphi)\, (c\, x) \\ H\, \varphi\, y & \to & a\, (\varphi\, (b\, y)) \\ I\, x & \to & x \end{cases}$$

Relating the two generator-families: word-language case

Theorem (Equi-expressivity)

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

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

generate the same class of word languages.

What is safety? (See later.)

Maslov Hierarchy: Many Open Problems

1. Pumping Lemma, Myhill-Nerode, and Parikh Theorems.
   Weak "pumping lemmas" for levels 1 and 2 (Hayashi 73, Gilman 96).
   Pace (Blumensath 08) for Maslov Hierarchy – but runs (*not* plays) are pumpable, conditions given as lengths of runs and configuration size.
2. Logical characterisations.
   E.g. MSOL for regular languages (Büchi 60). Characterisation of CFL using quantification over matchings (LST 94).
3. Complexity-theoretic characterisations.
   Pace (Engelfriet 83, 91): characterisations of languages accepted by alternating / two-way / multi-head / space-auxiliary order-n PDA as time-complexity classes (but no result for Maslov Hierarchy itself)
4. Relationship with Chomsky Hierachy. E.g. is level 3 context-sensitive?

Why study the two families of generators?

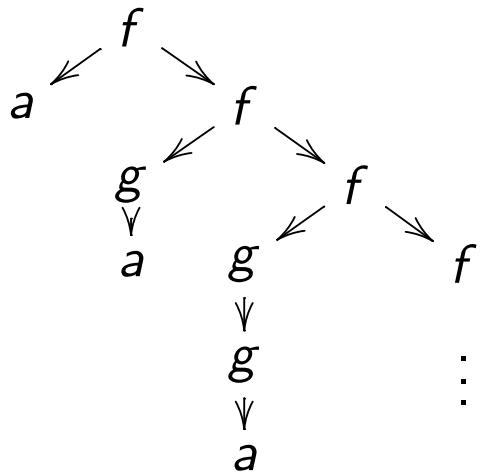
They are relevant to both semantics and verification:

1. Recursion schemes are an old and influential formalism for the semantical analysis of imperative and functional programs (Nivat 75, Damm 82).
   They are a compelling model of computation for higher-order functional programs.
2. Pushdown automata characterize the control flow of 1st-order (recursive) procedural programs.
   Pushdown checkers (e.g. MOPED) are essential back-end engines of state-of-the-art software model checkers (e.g. SLAM, Terminator).
3. Higher-order (collapsible) pushdown automata are highly accurate models of computation of higher-order procedural programs.

A challenge problem in higher-order verification

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

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



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

Let $\mathbf{RecSchTree}_n$ be the class of Σ-labelled trees generated by order-n recursion schemes.

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?

Why study MSO logic?

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

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

What is MSO logic?

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

Fix a vocabulary:

- Parent-child relationship between nodes: $\mathbf{d}_i(x, y) \equiv$ "y is i-child of x"
- Node labelling: $\mathbf{p}_f(x) \equiv$ "x has label f" where f is a Σ-symbol
- Set-membership: $x \in X$

First-order variables: x, y, z, etc. (ranging over nodes)
Second-order variables: X, Y, Z, etc. (ranging over sets of nodes)

MSO formulas are generated from three kinds of atomic formulas:

$$\mathbf{d}_i(x, y), \quad \mathbf{p}_f(x), \quad x \in X$$

and closed under boolean connectives, first-order quantification $(\forall x.-, \exists x.-)$ and second-order quantifications: $(\forall X.-, \exists X.-)$.

A Σ-labelled tree $t : dom(t) \longrightarrow \Sigma$ is represented as a structure

$$\langle\, dom(t), \quad \langle\, \mathbf{d}_i : 1 \leq i \leq m \,\rangle, \quad \langle\, \mathbf{p}_f : f \in \Sigma \,\rangle \,\rangle$$

Examples of MSO-definable properties

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. "Is-an-ancestor-of" or prefix ordering $x \leq y$ (and hence $x = y$):

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

Reachability property: "X is a path"

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

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

E.g. MSO can expresss "$\exists$ infinitely many f-labelled nodes"

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

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

Lemma

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.

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

Hence "there are finitely many nodes labelled by f" is expressible in MSO by $\exists X \,.\, \mathrm{Finite}(X) \;\wedge\; \forall x \,.\, \mathbf{p}_f(x) \rightarrow x \in X$.
But "MSO cannot count": E.g. "X has twice as many elements as Y".

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

- 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.
- Subsuming all the above:
  Caucal (MFCS 2002). **CaucalTree**$_n\Sigma$ and **CaucalGraph**$_n\Sigma$.

Theorem (KNU-Caucal 2002)

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

What is the safety constraint on recursion schemes?

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

Definition (Damm TCS 82, KNU FoSSaCS'02)

An order-2 equation is unsafe if the RHS has a subterm P s.t.

1. P is order 1
2. P occurs in an operand position (i.e. as 2nd argument of application)
3. P contains an order-0 parameter.

Consequence: An order-i subterm of a safe term can only have free variables of order at least i.

Example (unsafe eqn): $F : (o \to o) \to o \to o \to o,\ f : o^2 \to o,\ x, y : o.$

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

What is the point of safety?

Safety does have an important algorithmic advantage!

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

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

Theorem

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

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

Infinite structures generated by recursion schemes: key questions

1. **MSO decidability**: Is safety a genuine constraint for decidability?
   I.e. do trees generated by (arbitrary) recursion schemes have decidable MSO theories?
2. **Machine characterisation**: Find a hierarchy of automata that characterise the expressive power of recursion schemes.
   I.e. how should the power of higher-order pushdown automata be augmented to achieve equi-expressivity with (arbitrary) recursion schemes?
3. **Expressivity**: Is safety a genuine constraint for expressivity?
   I.e. are there inherently unsafe word languages / trees / graphs?

Infinite structures generated by recursion schemes: key questions

4 **Graph families**:

1. **Definition**: What is a good definition of "graphs generated by recursion schemes"?
2. **Model-checking properties**: What are the decidable (modal-) logical theories of the graph families?

Q1. Do trees in RecSchTree$_n\Sigma$ have decidable MSO theories?

Some progress:

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

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

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

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

What about higher orders?

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

Q1. Do trees in RecSchTree$_n\Sigma$ have decidable MSO theories? Yes

Theorem (O. LICS 2006)

For $n \geq 0$, the modal mu-calculus model-checking problem for **RecSchTree**$_n\Sigma$ *(i.e. trees generated by order-n recursion schemes) is n-EXPTIME complete. Thus these trees have decidable MSO theories.*

Proof Idea. Two key ingredients:

Generated tree $[\![G]\!]$ satisfies mu-calculus formula φ

$\iff$ { Emerson + Jutla 1991}

APT $\mathcal{B}_\varphi$ has accepting run-tree over generated tree $[\![G]\!]$

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

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

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

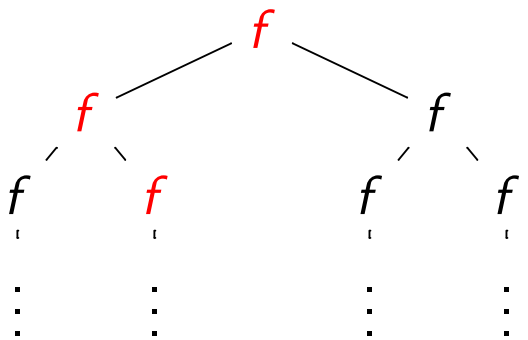
APT $\mathcal{C}_\varphi$ has an accepting run-tree over computation tree $\lambda(G)$

which is decidable because $\lambda(G)$ is regular.

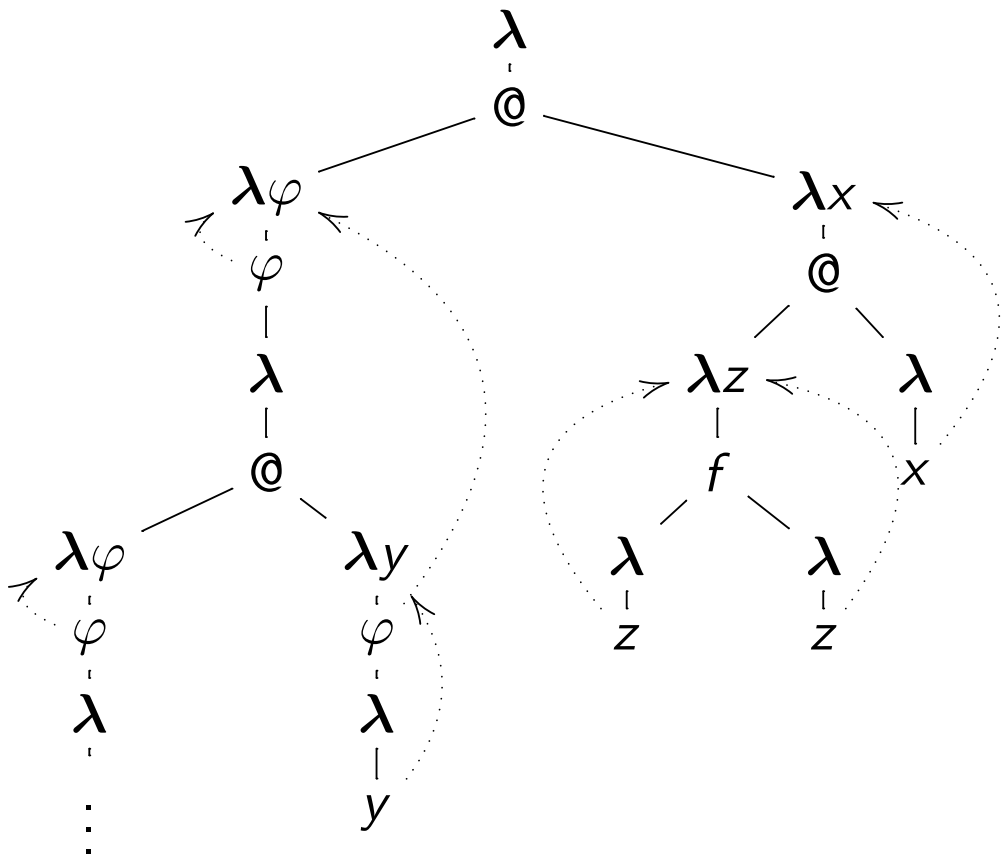
Transference principle, based on a theory of traversals

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

$[\![\, G \,]\!]$



$\lambda(G)$



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

Theorem (Path-traversal correspondence)

Let G be an order-n recursion scheme.

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

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

Proof is by game semantics.

Explanation (for game semanticists):

- Term-tree $[\![G]\!]$ is (a representation of) the game semantics of G.
- Paths in $[\![G]\!]$ correspond to plays in the strategy-denotation.
- Traversals t_p over computation tree $\lambda(G)$ are just (representations of) the uncoverings of the plays (= path) p in the game semantics of G.

Q2: Machine characterization: collapsible pushdown automata

Order-2 collapsible pushdown automata [HOMS, LiCS 08a] are essentially the same as 2PDA with links [AdMO 05] and panic automata [KNUW 05].

Idea: Each stack symbol in 2-stack "remembers" the stack content at the point it was first created (i.e. $push_1$ed onto the stack), by way of a pointer to some 1-stack underneath it (if there is one such).

Two new stack operations: $a \in \Gamma$ (stack alphabet)

- $push_1\ a$: pushes a onto the top of the top 1-stack, together with a pointer to the 1-stack immediately below the top 1-stack.
- *collapse* (= panic) collapses the 2-stack down to the prefix pointed to by the top_1-element of the 2-stack.

Note that the pointer-relation is preserved by $push_2$.

Collapsible pushdown automata: extending to all finite orders

In **order-n CPDA**, there are $n-1$ versions of $push_1$, namely, $push_1^j\ a$, with $1 \leq j \leq n-1$:

> $push_1^j\ a$: *pushes a onto the top of the top 1-stack, together with a pointer to the j-stack immediately below the top j-stack.*

Example: Urzyczyn's Language U over alphabet $\{(,),*\}$

Definition (Aehlig, de Miranda + O. FoSSaCS 05) A *U*-**word** has 3 segments:

$$\underbrace{(\cdots(\cdots(}_{A}\ \underbrace{(\cdots)\cdots(\cdots)}_{B}\ \underbrace{*\cdots*}_{C}$$

- Segment A is a prefix of a well-bracketed word that ends in $($, and the opening $($ is not matched in the entire word.
- Segment B is a well-bracketed word.
- Segment C has length equal to the number of $($ in segment A.

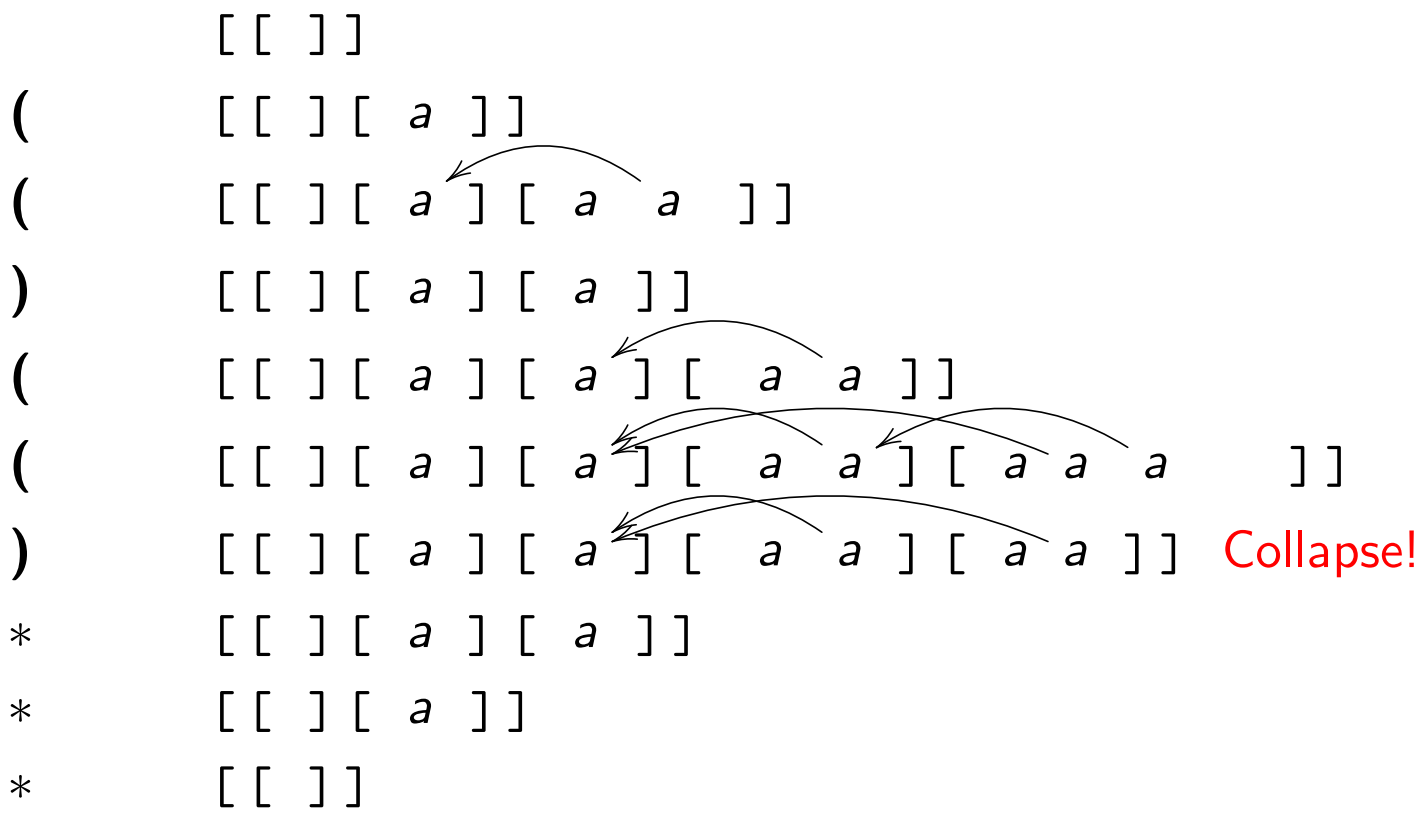
Examples

1. $(\,(\,)\,(\,(\,)\,(\,(\,)\,)\,*\ *\ *$ is a U-word
2. For each $n \geq 0$, we have $(\,(^n\,)^n\,(\ *^n\ **$ is a U-word.
   Hence by "*uvwxy* Lemma", U is not context-free.

Recognising U by a (det.) 2CPDA. E.g. (( ) (( ) $*$ $*$ $*$ $\in U$

(Ignoring control states for simplicity)

Upon reading	Do
(	$push_2$; $push_1 a$
)	pop_1
first $*$	*collapse*
subsequent $*$	pop_2



What does the depth of the top 1-stack mean?

E.g. Urzyczyn's Language U (cont'd)

Observation

1. U is recognisable by a deterministic order-2 CPDA.
2. Equivalently (thanks to [AdMO 05]) U is recognisable by a non-deterministic order-2 PDA — because of the need to guess the transition from segment A to segment B.

Theorem (Parys 2010)

U is not recognisable by a deterministic order-2 PDA.

(Related to the Safety Conjecture - more anon.)

Exercise (moderately hard). Give an order-2 recursion scheme that generates U.

Q2: Recursion schemes are equi-expressive with CPDA

Theorem (Equi-Expressivity, Hague, Murawski, O. + Serre LICS'08)

For each $n \geq 0$, order-n recursion schemes and order-n collapsible PDA are equi-expressive for Σ-labelled trees. I.e. $\mathbf{RecSchTree}_n\Sigma = \mathbf{CPDATree}_n\Sigma$

(Proof uses theory of traversals, based on game semantics.)

Consequences:

1. **Kleene's Problem:** What computing power is required to compute order-n lambda-definable functionals?

   n-CPDA gives a precise and natural "closed form" answer (as opposed to saying that it is some machine restricted to well-typed terms of order n.)
2. A **new proof** of the MSO decidability of trees generated by order-n recursion schemes.

Q3: Does safety constrain expressivity?

Case 1: Word languages.

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

At order 2, there are no inherently unsafe word languages. I.e. for every unsafe order-2 recursion scheme, there is a safe (non-deterministic) order-2 recursion scheme that generates the same language.

Conjecture: Yes, for $n \geq 3$.

Case 2: Trees.

The Safety Conjecture (Version 1)

For each $n \geq 2$, there is a tree generated by an unsafe order-n recursion scheme but not by any safe order-n recursion scheme.

True for order 2. Paweł Parys (2010).

Several versions of the Conjecture make sense.

Conjecture: Yes.

Q3: Does safety constrain expressivity?

Case 3: Graphs. Yes.

Theorem (Hague, Murawski, O. + Serre LICS 2008a)

There is an order-2 CPDA graph that is not generated by any order-2 PDA.

(See example graph later.)

A survey of graph families with model-checking properties

	Decidable?			
	MSO	μ	FO(R)	FO
Caucal's Graph Hierarchy	yes	yes	yes	yes
C	no	yes	?	?
Ground-term tree rewriting (Löding 02)	no	no	yes	yes
Automatic graphs (Hodgson 76, KN 94)	no	no	no	yes
Rational graphs	no	no	no	no

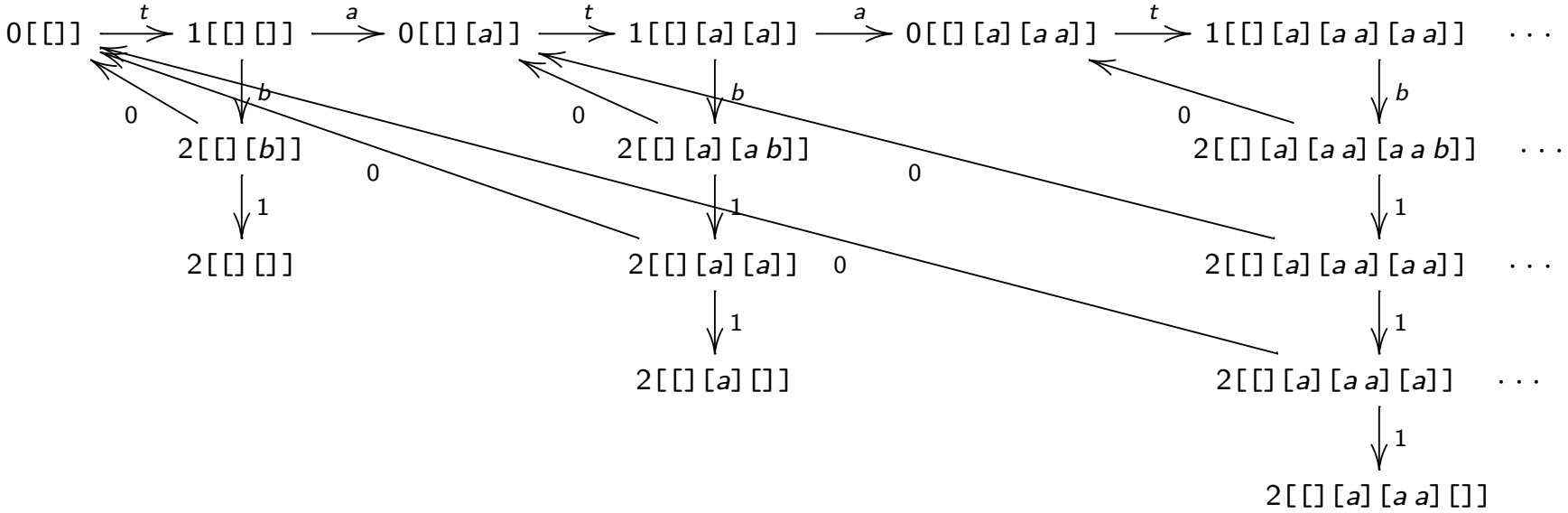
Question

Is there a generically-defined family **C** of graphs that have decidable modal-mu calculus theories but undecidable MSO theories?

Yes. See construction on next slide (HMOS, LiCS 08a).

Configuration graphs of (order-2) CPDA is not MSO-decidable

An order-2 CPDA graph: MSO-interpretable into the infinite half-grid.



To our knowledge CPDA graphs are the first "natural" generically-defined graph families that have decidable modal mu-calculus theories but undecidable MSO theories.

Q4: Model-checking properties of CPDA graphs

Theorem (Hague, Murawski, O and Serre, LiCS 2008a)

1. *For each $n \geq 0$, the decidability of modal mu-calculus model-checking problem for configuration graphs of order-n CPDA is n-EXPTIME complete.*
2. *Equivalently solvability of parity games over order-n CPDA graphs is n-EXPTIME complete.*

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 recognizable 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.

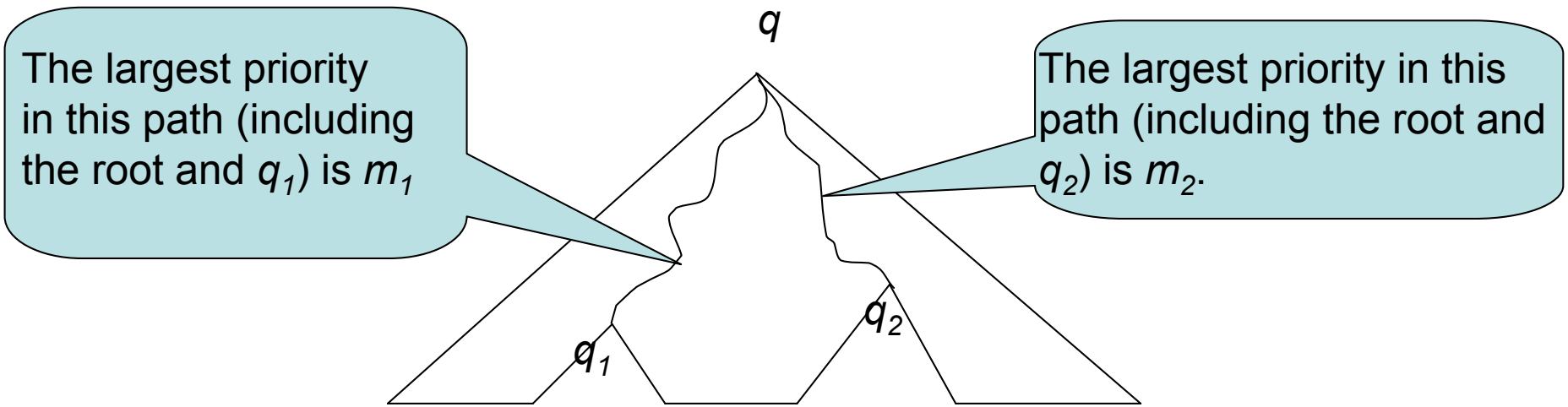
Intersection types embedded with states and priorities

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.

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

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



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

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

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

$$\frac{\Gamma_0 \vdash s : (\theta_1, m_1) \wedge \cdots \wedge (\theta_k, m_k) \to \theta \qquad \Gamma_i \vdash t : \theta_i \text{ for each } i \in \{1, \ldots, k\}}{\Gamma_0 \cup (\Gamma_1 \uparrow m_1) \cup \cdots \cup (\Gamma_k \uparrow m_i) \vdash s\ t : \theta} \quad \text{(T-App)}$$
$$\text{where } \Gamma \uparrow m = \{ F : (\theta, \max(m, m')) \mid F : (\theta, m') \in \Gamma \}$$

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

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.

Decidability

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].

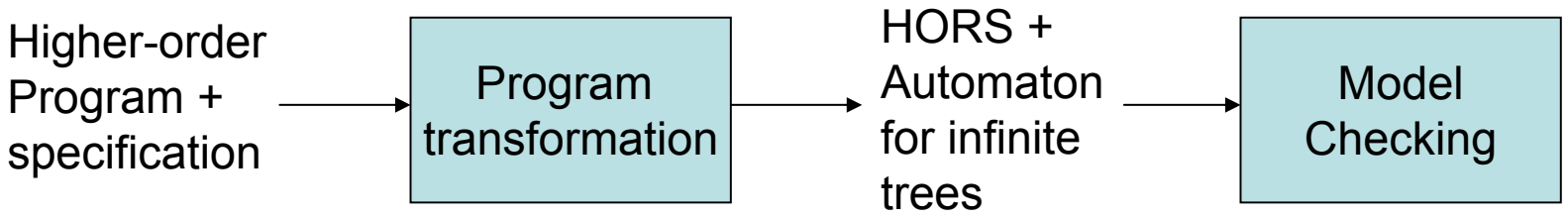
Why study trivial and disjunctive APT?

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*

Verification by Reduction to Model Checking HORS



Verification Problem: "Does P satisfy temporal specification φ?"

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

This method is fully automatic, sound and complete (for Resource Usage Verification Problem, Kobayashi POPL 2009).

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

Resource Usage Verification Problem (Igarashi + Kobayashi 2006)

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: $\varphi =$ "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?

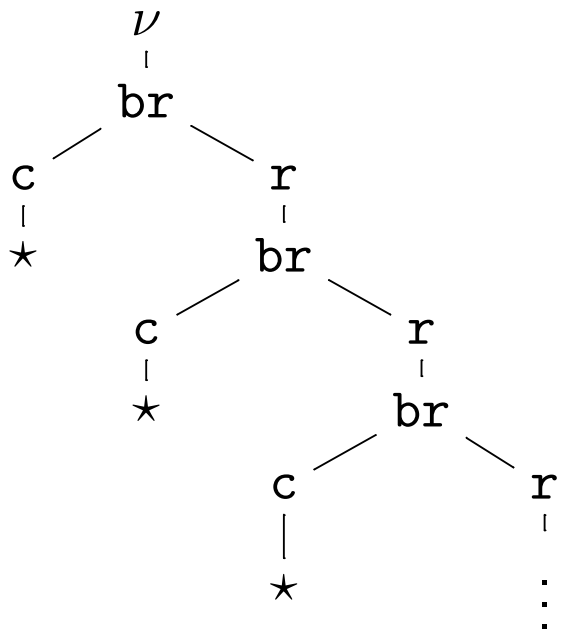
Are questions of this kind decidable?

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

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

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).

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). 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`

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

This program[2] 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. $AG\,(close_{in} \Rightarrow AF\,close_{out})$.

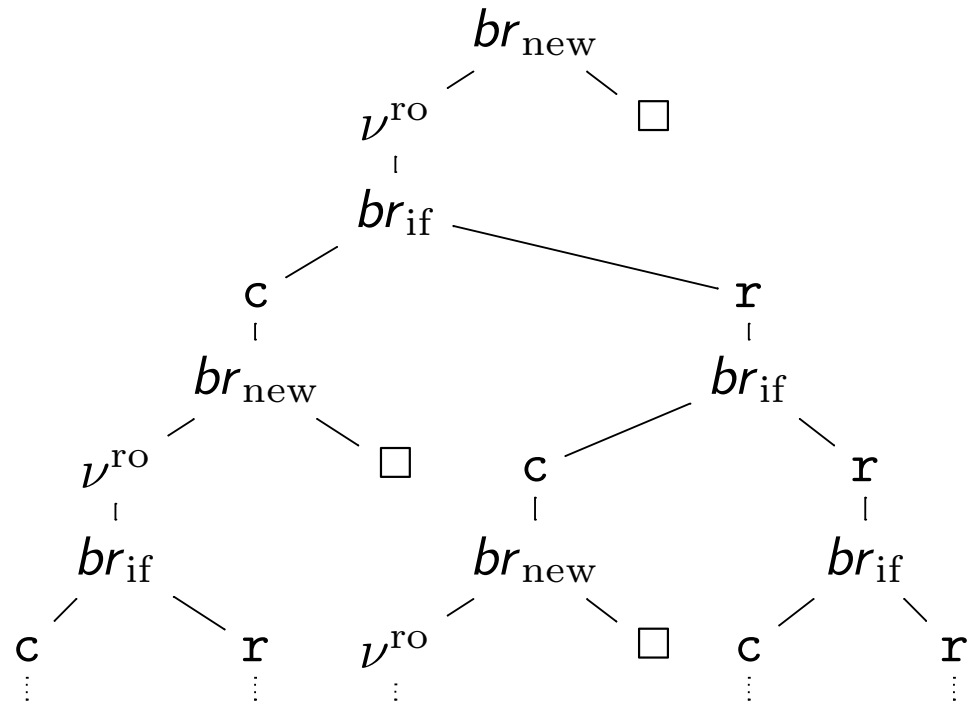
[2] obtained by "slicing" `intercept.ml` (about 110 LOC) at `http://abaababa.ouvaton.org/caml`.

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

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



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

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

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;

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.