Model-based approach to verification of higher-order programs

Igor Walukiewicz
Bordeaux University
joint work with Sylvain Salvati

Verification in three steps

1. Program → λ-term
$P \rightarrow M$

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \vDash \varphi$

Verification in three steps

1. Program → λ-term
$P \to M$

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \vDash \varphi$

A general technique for analysis of higher-order programs

Control flow is represented faithfully, but the data part is abstracted

Verification in three steps

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

Why higher order?

Programs can manipulate data as well as other programs.
Powerful abstraction mechanism.
Analysis is challenging and algorithmically difficult.

1. Program → λ-term
P → M

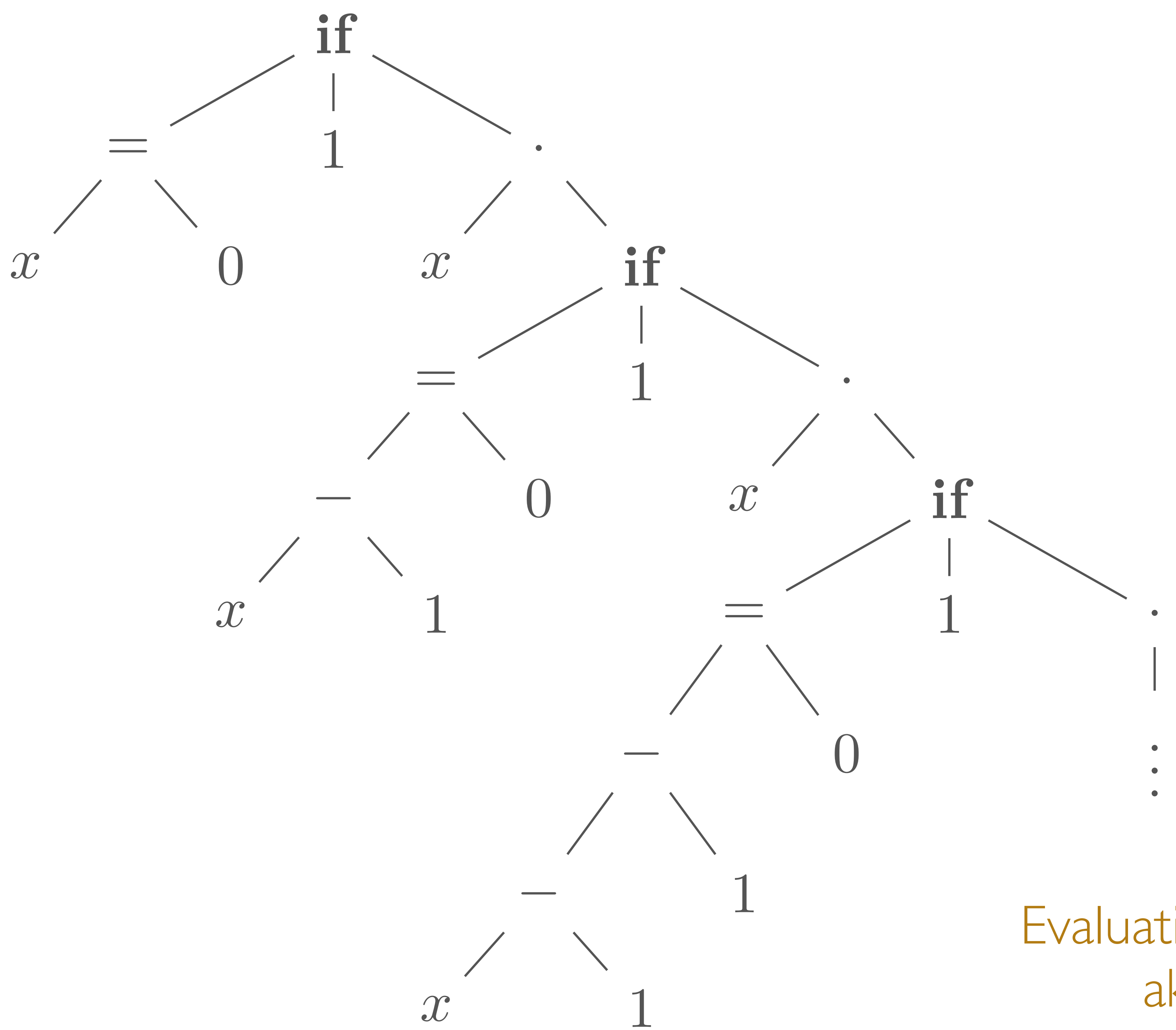
2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

First example: factorial

$$Fct(x) \;\equiv\; \textbf{if}\ x = 0\ \textbf{then}\ 1\ \textbf{else}\ Fct(x-1)\cdot x\ .$$

$$Fct(x) \;\equiv\; \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } Fct(x-1)\cdot x \;.$$



Evaluation tree
aka
Böhm tree

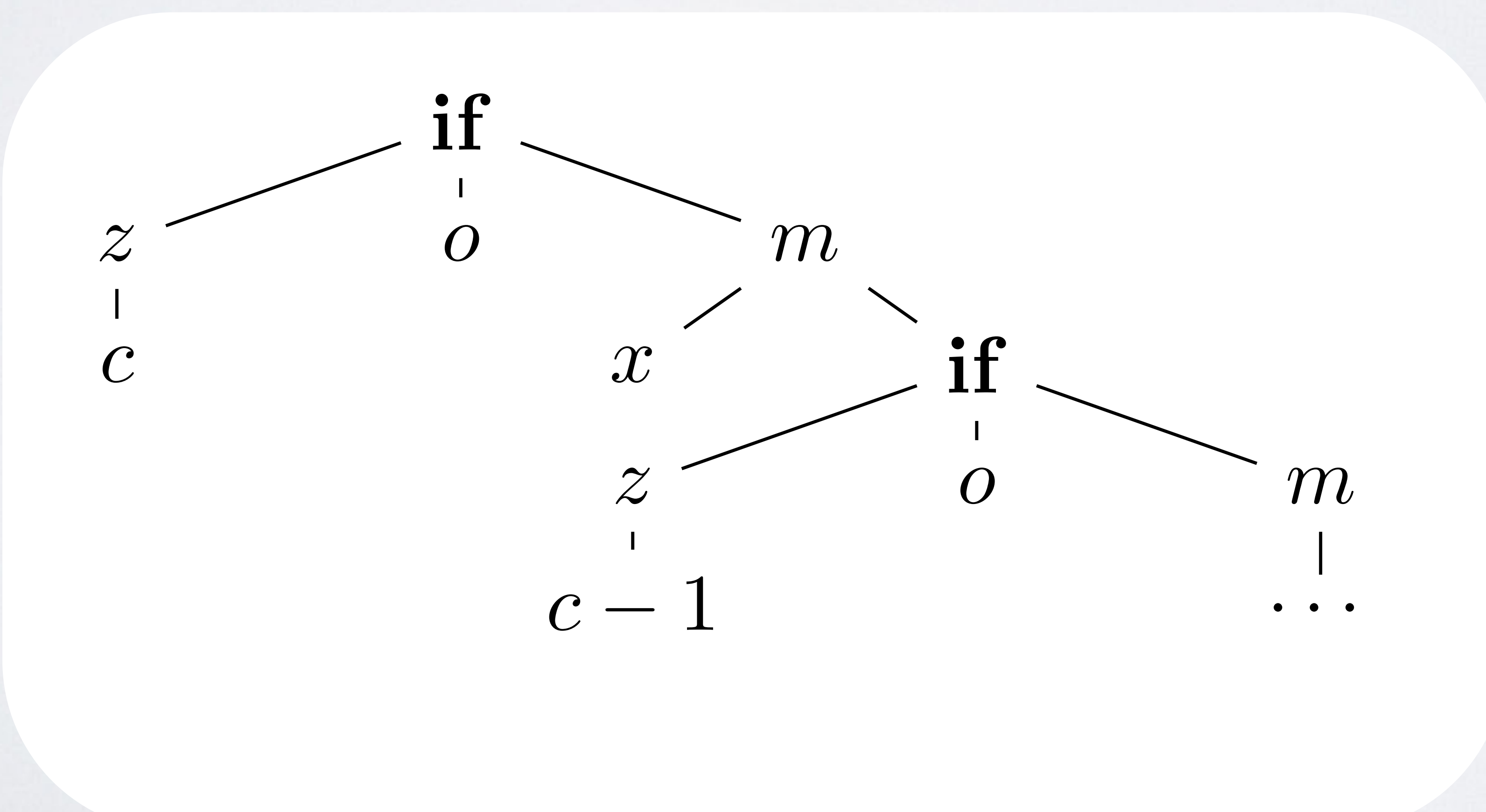
1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

$$Fct(x) \ \equiv \ \textbf{if} \ x = 0 \ \textbf{then} \ 1 \ \textbf{else} \ Fct(x-1)\cdot x \ .$$

$$Y Fct.\ \lambda x.\ \textbf{if-then-else}(z(x), o, m(Fct(x-1), x))$$

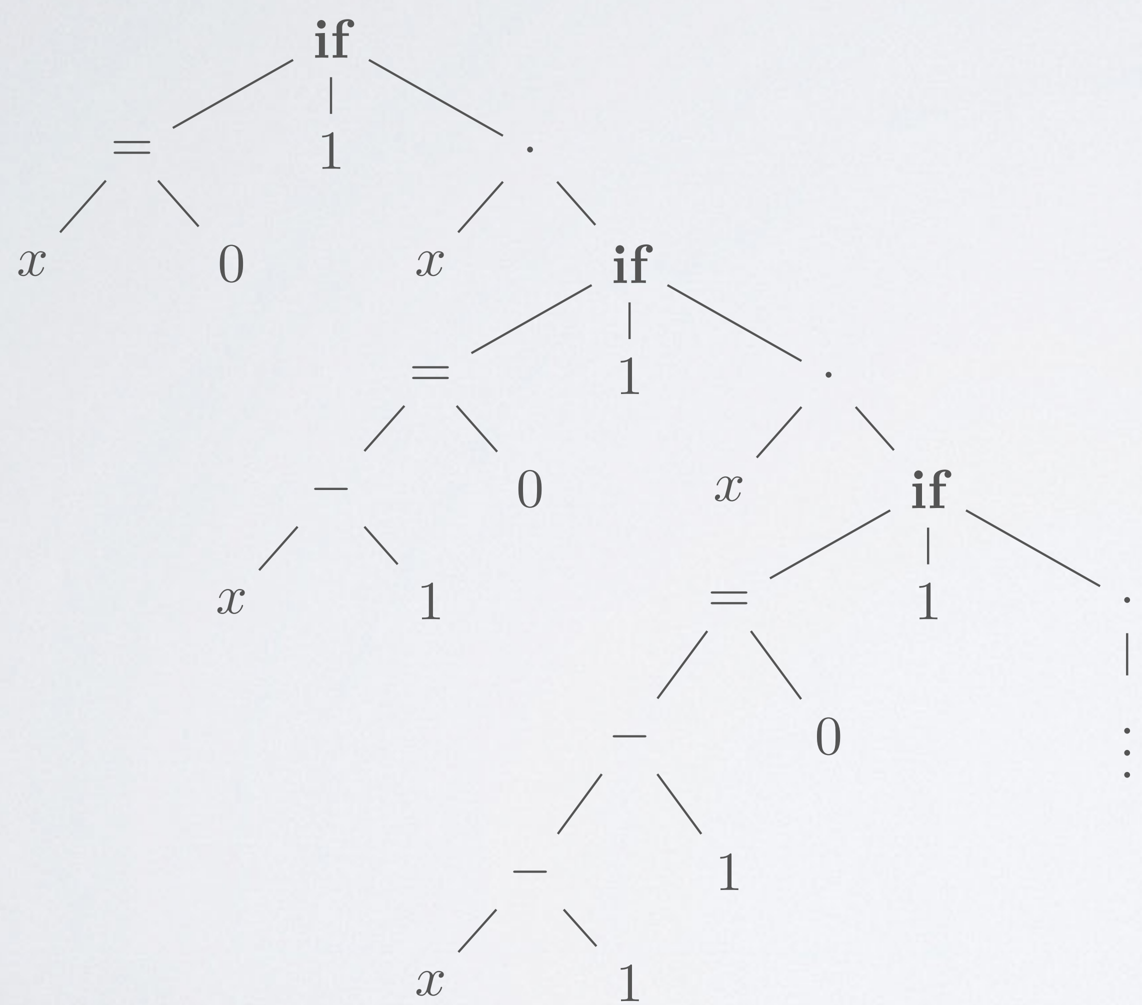


1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

$$Y Fct.\ \lambda x.\ \textbf{if-then-else}(z(x), o, m(Fct(x-1), x))$$



Property:
Every path with a left turn is finite

Second example: javascript

[Grabowski, Hofmann, Li]

The problem: malicious code execution via specially crafted input string

```
let makecode(x)="<script> alert(" + x + "); </script>"
in
    y=first(untrusted_stream);
    output(makecode(y));
```

How: input string escapes the alert function

```
makecode("); form.submit(http://...); ")=
    alert(); form.submit(http://...);
```

Protection: always validate input strings to be executed

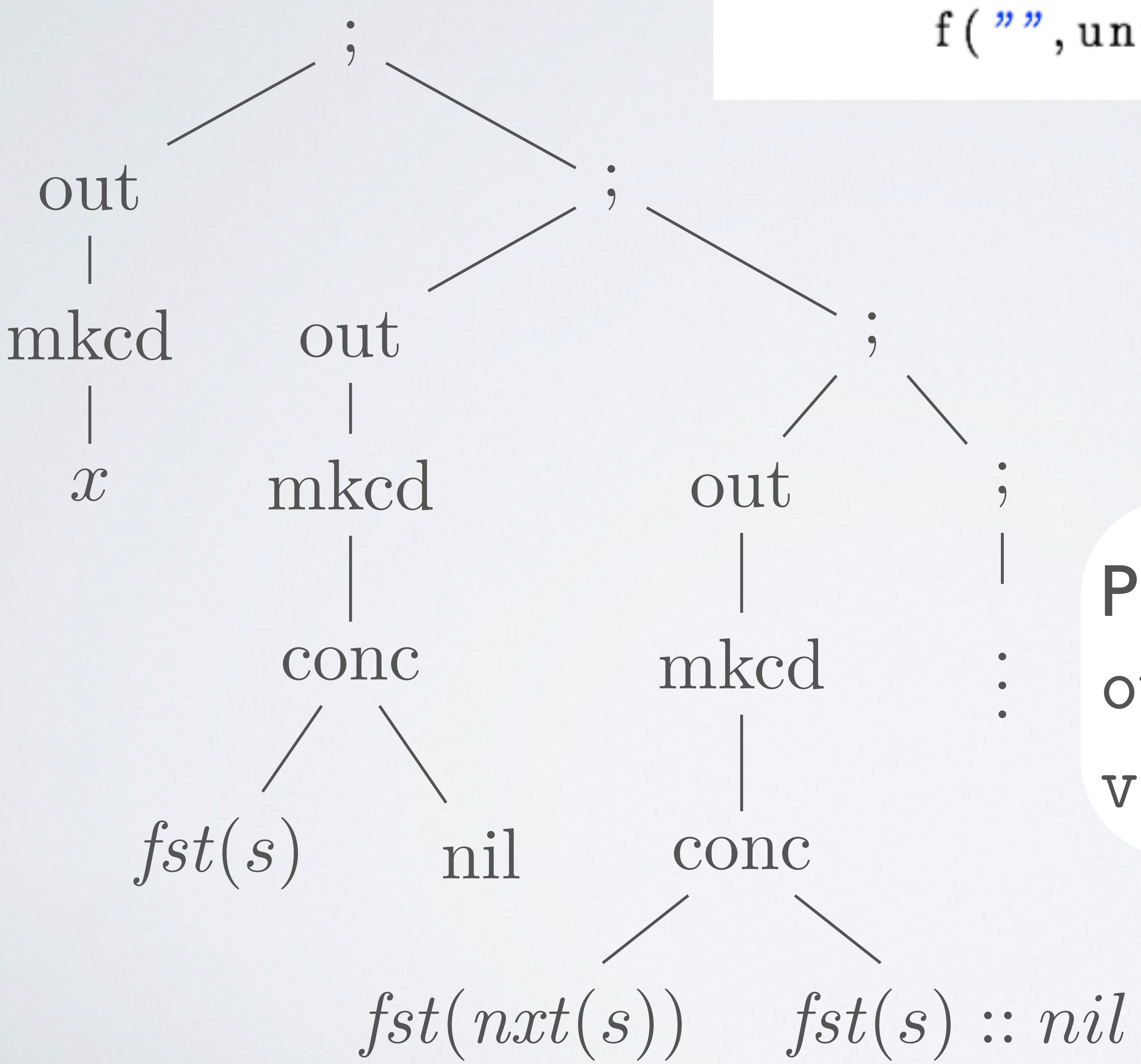
```
let makecode(x)="<script> alert(" + x + "); </script>"
in
    y=first(untrusted_stream);
    output(makecode(validate(y)));
```

Second example: javascript (cont)

The same but in some more elaborate setting:

```
let  makecode(x)=..  in
letrec  f(x,s)=
                    let  y=first(s)  in
                         output(makecode(x));
                         f(conc(y,x),next(s));
in
          f("",untrusted_stream);
```

```
let makecode(x)=.. in
letrec f(x,s)=
                let y=first(s) in
                    output(makecode(x));
                    f(conc(y,x),next(s));
in
        f("",untrusted_stream);
```



Property: always between `out`, and `s` there should be `validate`

Examples of properties

- reachability
  fail constant is reachable
- resource usage
  every open file is eventually closed
- method invocation patterns
  m.init should appear before m.usage
- fairness properties
  if access is asked infinitely often then it is granted infinitely often

Third example: CFA

[Tobia, Tsukada, Kobayashi]

The problem: determine what functions are called at a given location.

$$(\lambda^1 x.\ x@^2())@^3(\lambda^4 z.()).$$

What functions can be called at the context 2?

Setting: a small simply typed call-by-value language :

$$\begin{aligned} t\ (\text{terms}) &::= () \mid x \mid \mathtt{fun}^\ell(f,x,t) \mid t_1 @^\ell t_2 \mid \mathtt{if*}\ t_1\ t_2 \\ v\ (\text{value}) &::= () \mid \mathtt{fun}^\ell(f,x,t) \\ T\ (\text{types}) &::= \mathtt{Unit} \mid T_1 \to T_2. \end{aligned}$$

Objective: determine if ℓ_2 is called at ℓ_1

$$t \longrightarrow^* E[\mathtt{fun}^{\ell_2}(f,x,t')@^{\ell_1} v]$$

CPS translation to call sequence problem

$$
\begin{aligned}
[\![()]\!] &= \lambda k.\ k\ () \\
[\![x]\!] &= \lambda k.\ k\ x \\
[\![\mathtt{fun}^{\ell}(f, x, t)]\!] &= \lambda k.\ k\ (\mathtt{fun}^{\ell}(f, x, [\![t]\!])) \\
[\![t_1 @^{\ell} t_2]\!] &= \lambda k.\ [\![t_1]\!]\ (\lambda f.\ [\![t_2]\!]\ (\lambda^{\ell} z.\ (f\ z)\ k)) \qquad f, z \text{ fresh} \\
[\![\mathtt{if}{*} t_1 t_2]\!] &= \lambda k.\ \mathtt{if}{*}\ ([\![t_1]\!]\ k)\ ([\![t_2]\!]\ k)
\end{aligned}
$$

CSA problem: determine if ℓ_2 is called just after ℓ_1

$$
t \longrightarrow^{*} E_1[\mathtt{fun}^{\ell_1}(f_1, x_1, t_1)\ v_1] \rightarrow E_2[\mathtt{fun}^{\ell_2}(f_2, x_2, t_2)\ v_2]
$$

Fact: CFA is reduced to CSA.

Translation to the model-checking problem

CFA problem: determine if ℓ_2 is called at ℓ_1

$$t \longrightarrow^* E[\mathbf{fun}^{\ell_2}(f, x, t')@^{\ell_1} v]$$

CSA problem: determine if ℓ_2 is called just after ℓ_1

$$t \longrightarrow^* E_1[\mathbf{fun}^{\ell_1}(f_1, x_1, t_1)\ v_1] \rightarrow E_2[\mathbf{fun}^{\ell_2}(f_2, x_2, t_2)\ v_2]$$

Model-checking problem: determine if in the evaluation tree of t there is a path where ℓ_2 appears just after ℓ_1

$$\langle \mathbf{fun}^{\ell}(f, x, \lambda k.\ t) \rangle = \mathbf{fun}(f, x, \lambda k. \ell \langle t \rangle)$$

Translation to the model-checking problem

CFA problem: determine if ℓ_2 is called at ℓ_1

$$t \longrightarrow^* E[\mathbf{fun}^{\ell_2}(f, x, t')@^{\ell_1} v]$$

CSA problem: determine if ℓ_2 is called just after ℓ_1

$$t \longrightarrow^* E_1[\mathbf{fun}^{\ell_1}(f_1, x_1, t_1)\ v_1] \to E_2[\mathbf{fun}^{\ell_2}(f_2, x_2, t_2)\ v_2]$$

Model-checking problem: determine if in the evaluation tree of t there is a path where ℓ_2 appears just after ℓ_1

$$\langle \mathbf{fun}^{\ell}(f, x, \lambda k.\ t) \rangle = \mathbf{fun}(f, x, \lambda k. \ell\langle t \rangle)$$

- The analysis is exact (modulo data abstraction)
- It has (unavoidable) big complexity.
- It is selective.
- Any other kind of program analysis can be encoded that way

1. Program → λ-term
$P \to M$

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \models \varphi$

- We consider programs with: semicolon, let, and evaluation by value.
- We use λY-calculus: simply typed λ-calculus with fix point operator as our target language
- To translate programs to λY-calculus we can use some sort of CPS translation.

1. Program → λ-term

$P \rightarrow M$

(P and M have similar Böhm trees)

2. Property → MSOL-formula

'no fail' → φ

3. Verification

$BT(M) \models \varphi$

Why Böhm trees (evaluation trees) are interesting:

- Giving a denotational semantics for the full language is difficult.
- Standard denotational semantics talks about reachability/safety properties.
- A Böhm tree gives full interpretation of the control-flow,
  but does not interpret commands operating on data.

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \vDash \varphi$

Why MSOL:

- Standard logic for tree properties (regular tree properties).
- Can express many interesting properties.
- The MSOL theory of a Böhm tree of a λY-term is decidable (Ong's Theorem).

1. Program → λ-term
P → M

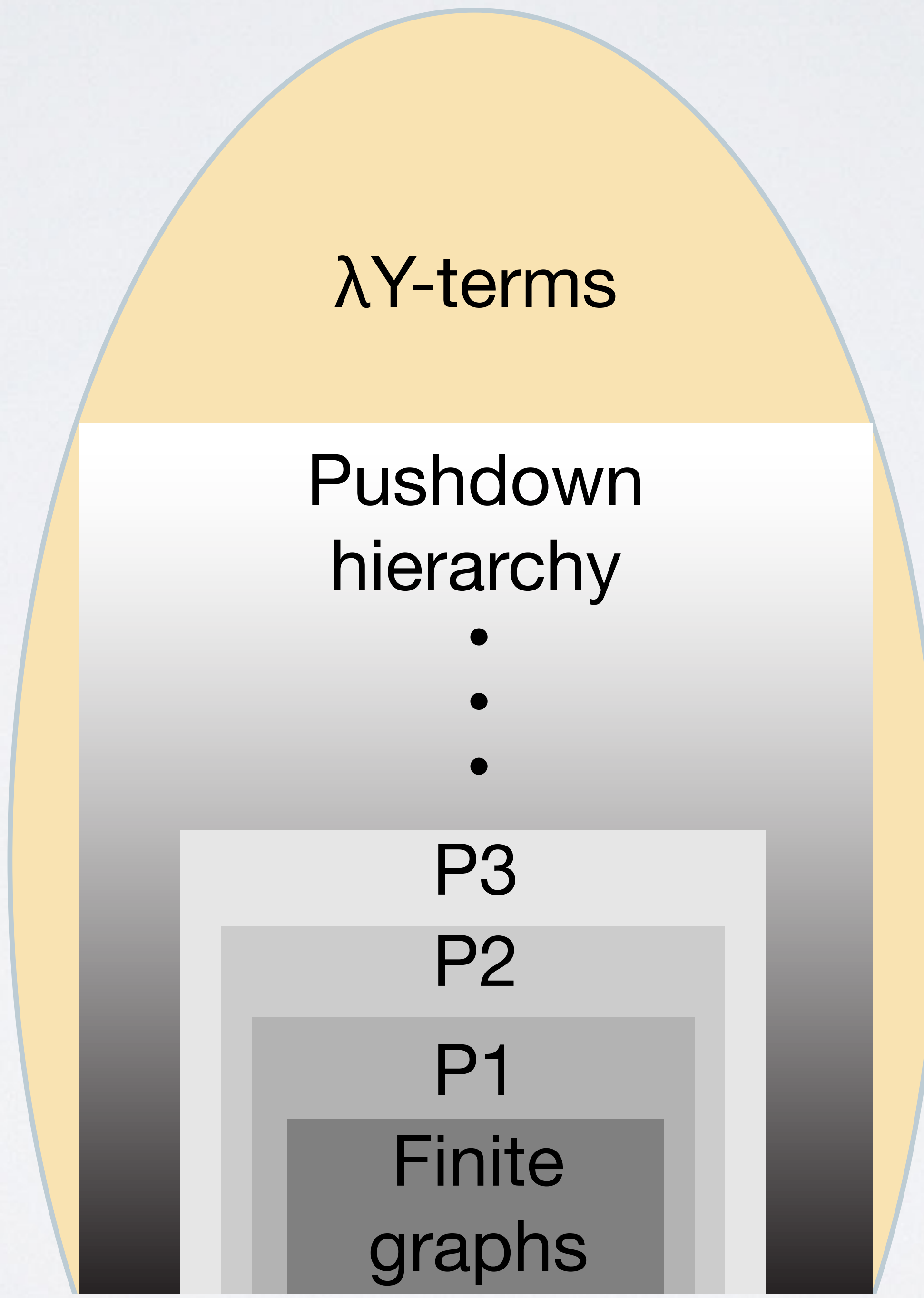
2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \models \varphi$

Is this really new?

Why infinite trees are more challenging than finite ones?

What are trees generated by λY-terms?



Digression: game determinacy

An infinite game: We are given a set $Win \subseteq \{0,1\}^\omega$;
In each turn Eve chooses a finite sequence of bits, then Adam chooses one too.
After infinitely many turns an infinite word $w_0 w_1 w_2 \ldots$ is formed.
Eve wins if $w_0 w_1 w_2 \cdots \in Win$.

Question: Is this game determined for every $Win \subseteq \{0,1\}^\omega$?

Eve has a winning strategy when the following is « true »:

$$\exists x_0 \forall x_1 \ldots . \ (x_0 x_1 \cdots \in Win)$$

Similarly for Adam:

$$\forall x_0 \exists x_1 \ldots . \ (x_0 x_1 \cdots \notin Win)$$

Determinacy is:

$$\neg\big[\exists x_0 \forall x_1 \ldots \ (x_0 x_1 \cdots \in Win)\big] \ \equiv \ \forall x_0 \exists x_1 \ldots \ (x_0 x_1 \cdots \notin Win)$$

Digression: game determinacy

An infinite game: We are given a set $Win \subseteq \{0,1\}^\omega$;
In each turn Eve chooses a finite sequence of bits, then Adam chooses one too.
After infinitely many turns an infinite word $w_0 w_1 w_2 \ldots$ is formed.
Eve wins if $w_0 w_1 w_2 \cdots \in Win$.

Question: Is this game determined for every $Win \subseteq \{0,1\}^\omega$?

Def: An *infinite XOR* is a function $f : \{0,1\}^\omega \to \{0,1\}$ such that:
if $w, w' \in \{0,1\}^\omega$ differ on only one position then $f(w) \neq f(w')$.

Prop: Infinite XOR exists.

Prop: Let f be an infinite XOR.
No player has a winning strategy in the game with
$Win_f = \{w : f(w) = 1\}$.

Digression: game determinacy

An infinite game: We are given a set $Win \subseteq \{0,1\}^\omega$;
In each turn Eve chooses a finite sequence of bits, then Adam chooses one too.
After infinitely many turns an infinite word $w_0 w_1 w_2 \dots$ is formed.
Eve wins if $w_0 w_1 w_2 \dots \in Win$.

Question: Is this game determined for every $Win \subseteq \{0,1\}^\omega$?

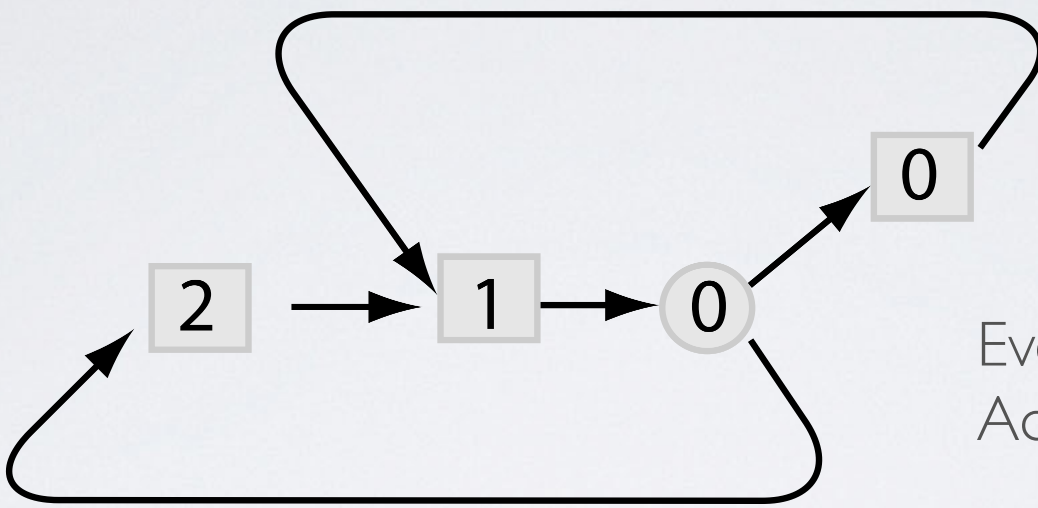
Thm [Martin]: If Win is Borel then the game is determined.

Some history:

- 1953 D.Gale & M.Steward Σ^0_1 determinacy.
- 1955 M.Wolfe Σ^0_2 determinacy.
- 1964 M.Davis Σ^0_3 determinacy.
- 1972 J.B.Paris Σ^0_4 determinacy.
- 1975 D.A.Martin Borel determinacy

Rem: With MSOL we are in $\Sigma^0_3 \cap \Pi^0_3$.

Parity games



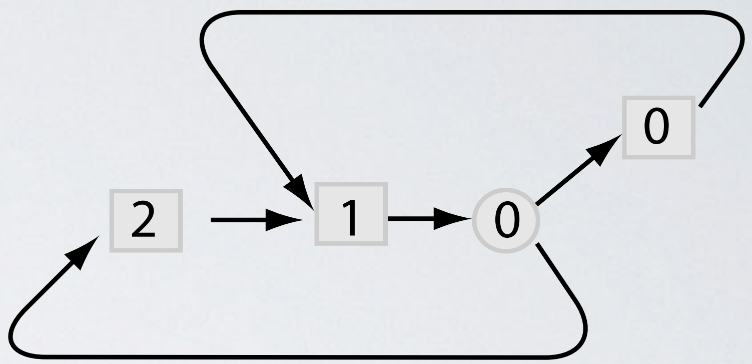
Eve makes the choice in round nodes,
Adam in square nodes.

Parity condition:

the biggest rank seen infinitely often is even.

The set of winning positions for Eve can be expressed with a fix point formula.

Reachability of W: $$\mu X.\ W \vee \begin{bmatrix} P_{\mathrm{Eve}} \Rightarrow \langle\rangle X \\ P_{\mathrm{Adam}} \Rightarrow []X \end{bmatrix}$$



Safety (avoiding L): $$\nu X.\ \neg L \wedge \begin{bmatrix} P_{\mathrm{Eve}} \Rightarrow \langle\rangle X \\ P_{\mathrm{Adam}} \Rightarrow []X \end{bmatrix}$$

Parity: $$\mu X_n.\nu X_{n-1}\ldots\mu X_1.\nu X_0.\ \begin{bmatrix} P_{\mathrm{Eve}} \Rightarrow \bigwedge_{i=0}^{n}(R_i \Rightarrow \langle\rangle X_i) \\ P_{\mathrm{Adam}} \Rightarrow \bigwedge_{i=0}^{n}(R_i \Rightarrow []X_i) \end{bmatrix}$$

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \models \varphi$

Why infinite trees are more challenging than finite ones?

Because MSOL can express parity games,
and winning in parity games involves nested least and greatest fixpoints.

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

Verification by evaluation:

For a given φ construct an interpretation of λY-terms D, and a set $F \subseteq D$ s.t. for every λY-term M:

$$BT(M) \vDash \varphi \quad \text{iff} \quad [\![M]\!]^D \in F.$$

- For finite words ➟ semigroups (algebraic theory of regular lang.)
- For infinite words ➟ Wilke algebras
- For finite trees ➟ pre-clones, forest algebras
- For infinite trees ➟ [Bojanczyk, Idziaszek], [Blumensath]

Semantics: GFP-models

Types: 0, $\alpha \to \beta$

Typed tems: c^α, x^α, $(M^{\alpha\to\beta} N^\alpha)^\beta$, $(\lambda x^\alpha . M^\beta)^{\alpha\to\beta}$, $(Y x^\alpha . M^\alpha)^\alpha$

Semantics, GFP-model

$$\mathcal{D}^{\mathcal{A}} = \langle \{D_\alpha\}_{\alpha\in\mathcal{T}}, [\![b]\!], \ldots\rangle \text{ where}$$

$$D_{\alpha\to\beta} = \mathrm{mon}[D_\alpha \mapsto D_\beta]$$

$$[\![Y f^{\alpha\to\alpha}.\ M^\alpha]\!]_v = \mathsf{GFP}(\lambda F.[\![M]\!]_{v[F/f]})$$

A model can *recognise* a set of terms:
a set $F \subseteq \mathcal{D}_0$ defines a set of closed terms $\{M : [\![M]\!]^{\mathcal{D}} \in F\}$.

What can finite GFP-models recognise?

TAC-automata

Tree automata with trivial acceptance conditions:

$$\mathcal{A} = \langle Q, \Sigma, \{\delta_b \subseteq Q \times Q^{ar(b)}_{b\in\Sigma}\}_{b\in\Sigma}\rangle$$

every run is accepting.

TAC-automaton $\equiv$ ν-formulas $\equiv$ safety properties

Prop. For every TAC-automaton $\mathcal{A}$,
there is a finite GFP-model recognising $\{M : eval(M) \in L(\mathcal{A})\}$.

Prop. For every finite GFP model $\mathcal{D}$ and $F \subseteq \mathcal{D}_o$,
there is a boolean combination $\mathcal{B}$ of languages of TAC automata s.t.
$\{M : [\![M]\!]^{\mathcal{D}} \in F\} = \{M : eval(M) \in \mathcal{B}\}$.

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

Verification by evaluation:

For a given φ construct an interpretation of λY-terms D, and a set $F \subseteq D$ s.t. for every λY-term M:

$$BT(M) \vDash \varphi \quad \text{iff} \quad [\![M]\!]^D \in F.$$

Models based on GFP can only handle prefix properties.

(by duality the same holds for LFP models)

Models for MSOL

Thm [Slavati, W.]
For every MSO property one can construct a finite model recognising the property.

A model can *recognise* a set of terms:
a set $F \subseteq \mathcal{D}_0$ defines a set of closed terms $\{M : [\![M]\!]^{\mathcal{D}} \in F\}$.

The fix point in the model is interpreted as an alternation of least and greatest fix points.

$$\nu X_2.\ \gamma_2(X_2,\ \mu X_1.\gamma_1(X_1,\ \nu X_0.\gamma_0(X_0)))$$

Applications of models

1. Decidability of the model-checking problem for MSO

Given an property φ and term M:

- construct the model $\mathcal{D}^{\varphi}$, and
- calculate the semantics of M in $\mathcal{D}^{\varphi}$.

Applications of models

2. Two type systems for WMSO properties

Every element of a model can be described by a type S.

$$\Gamma \vdash M \geq S \quad \text{iff} \quad [\![M]\!]^{\mathcal{D}}_{[\![\Gamma]\!]} \geq [\![S]\!]$$

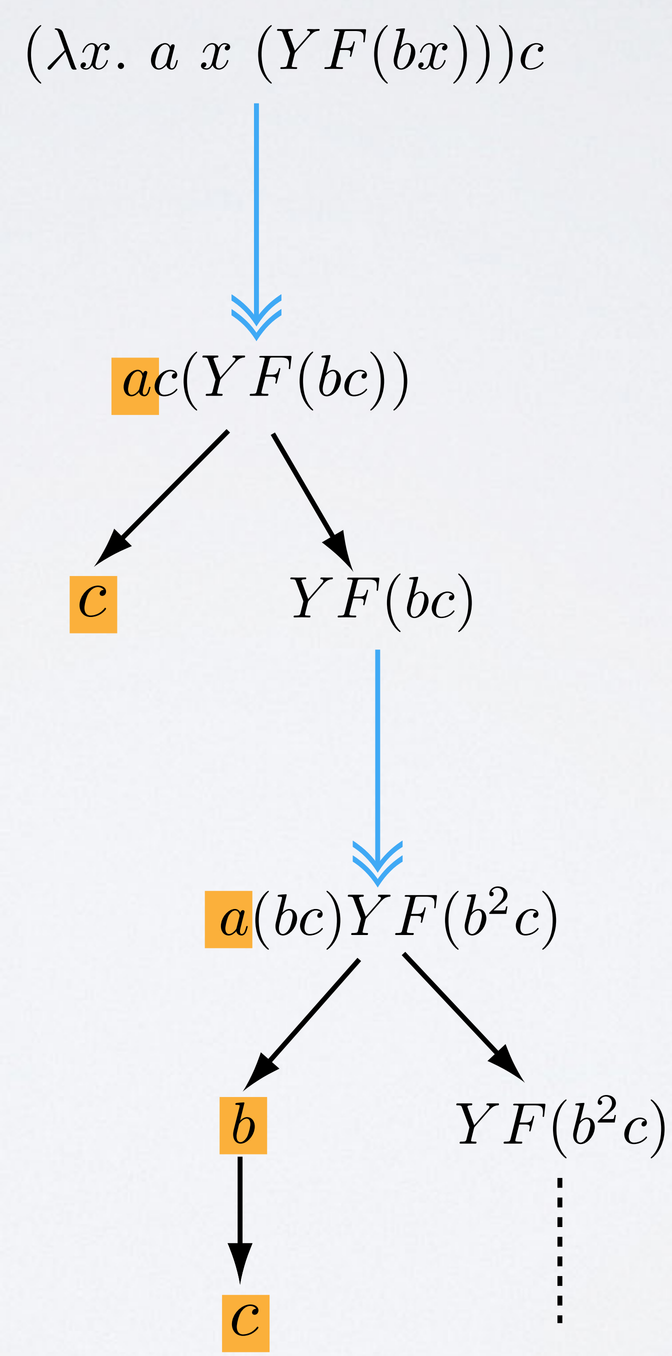
$$\Gamma \vdash M \leq S \quad \text{iff} \quad [\![M]\!]^{\mathcal{D}}_{[\![\Gamma]\!]} \leq [\![S]\!]$$

$$\frac{\Gamma \vdash M \geq S \quad \Gamma \vdash N \geq T}{\Gamma \vdash MN \geq S(T)} \qquad \frac{S \subseteq \mathit{Types}^k,\, T \subseteq \mathit{types}^k \quad \Gamma, x \geq S \vdash M \geq T}{\Gamma \vdash \lambda x.M \geq S \to T}$$

$$\frac{S, T \subseteq \mathit{Types}^{2k+1}_A, \quad \Gamma \vdash (\lambda x.M) \geq S \quad \Gamma \vdash (Yx.M) \geq T}{\Gamma \vdash Yx.M \geq S(T)}\, Y\ \mathit{odd}$$

Applications of models

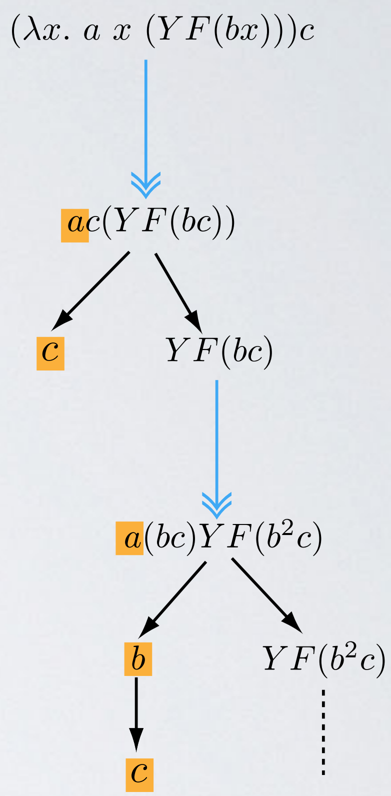
3. Program transformation



Applications of models

3. Program transformation

$$
\begin{aligned}
\alpha^{\bullet} &= \alpha \text{ when } \alpha \text{ is atomic} \\
(\alpha \to \beta)^{\bullet} &= \alpha^{\bullet} \to [\alpha] \to \beta^{\bullet}
\end{aligned}
$$



$$
\begin{aligned}
[\lambda x^{\alpha}.M, \upsilon] &= \lambda x^{\alpha^{\bullet}} \lambda y^{[\alpha]}.\ \text{case } y^{[\alpha]}\{d \to [M, \upsilon[d/x^{\alpha}]]\}_{d \in \mathcal{S}_{\alpha}} \\
[MN, \upsilon] &= [M, \upsilon]\,[N, \upsilon]\,[\![N]\!]^{\upsilon} \\
[a, \upsilon] &= \lambda x_1^0 \lambda y_1^{[0]} \lambda x_2^0 \lambda y_2^{[0]}. \\
&\qquad \text{case } y_1^{[0]}\{d_1 \to \text{case } y_2^{[0]}\{d_2 \to a^{\rho(a)d_1\,d_2} x_1 x_2\}_{d_2 \in \mathcal{S}_0}\}_{d_1 \in \mathcal{S}_0} \\
\left[Y^{(\alpha \to \alpha) \to \alpha} M, \upsilon\right] &= Y^{(\alpha^{\bullet} \to \alpha^{\bullet}) \to \alpha^{\bullet}}(\lambda x^{\alpha^{\bullet}}.\, [M, \upsilon]\, x^{\alpha^{\bullet}}\, [\![YM]\!]^{\upsilon})\ .
\end{aligned}
$$

Applications of models

4. Transfer theorem for MSO

Thm (Transfer)[Salvati & W.]
Fix a signature Σ, set of types $\mathcal{T}$, and a set of variables $\mathcal{X}$ (all finite sets).
For every MSOL formula φ there is an MSOL formula $\widehat{\varphi}$ s.t. for every term M over Σ, $\mathcal{T}$, $\mathcal{X}$:

$$M \vDash \widehat{\varphi} \quad \text{iff} \quad BT(M) \vDash \varphi$$

Consequences of the transfer theorem

$$M \vDash \widehat{\varphi} \qquad \text{iff} \qquad BT(M) \vDash \varphi$$

The set of SN terms over fixed set of variables is definable in MSOL

For a fixed $\mathcal{T}$ and $\mathcal{X}$ there is an MSOL formula defining the set of terms $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ having a normal form.

Take φ defining the set of finite trees and consider $\widehat{\varphi}$.

Consequences of the transfer theorem

$$M \vDash \widehat{\varphi} \qquad \text{iff} \qquad BT(M) \vDash \varphi$$

A « synthesis from modules » framework

Given λY-terms $M_1, \ldots, M_k$ and a formula φ.
Decide if one can construct from these terms a λY term K such that $eval(K) \vDash \varphi$.

- We can restrict to solutions K of the form
  $(\lambda x_1 \ldots x_k.\ N)M_1, \ldots, M_k$
  for some term N without constants and λ-abstractions.
- Let ψ be a formula defining terms of this form.
- There is a solution iff the formula $\psi \wedge \widehat{\varphi}$ is satisfiable.

$$M \vDash \widehat{\varphi} \qquad \text{iff} \qquad eval(M) \vDash \varphi$$

Consequences of the transfer theorem

$$M \vDash \widehat{\varphi} \qquad \text{iff} \qquad BT(M) \vDash \varphi$$

Higher-order matching with restricted no of variables

For a fixed $\mathcal{X}$. Given M and K (without fixpoints) decide if there is a substitution σ such that

$$M\sigma =_\beta K$$

Substitution Σ can use only terms from $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$.

- Let $\mathit{shape}(N)$ be MSOL formula defining the set of terms in $\mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ that can be obtained from N by substitutions.
- Let $\varphi \equiv \mathit{shape}(K)$.
- There is desired σ iff the formula $\mathit{shape}(M) \wedge \widehat{\varphi}$ is satisfiable.

If there is a solution then there is a finite one.

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \models \varphi$

Extending verification methods, from transition systems to a higher-order program calculus.

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

Verification by evaluation:

For a given φ construct an interpretation of λY-terms D, and a set $F \subseteq D$ s.t. for every λY-term M:

$$BT(M) \vDash \varphi \quad \text{iff} \quad [\![M]\!]^D \in F.$$

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \vDash \varphi$

Verification by evaluation

Extending verification methods, from transition systems to a higher-order program calculus.

Extending abstract interpretation to new kinds of models, and higher-order.

Extending typing with new kinds of types, namely behavioural types.

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
BT(M)⊨φ

Verification by evaluation:

For a given φ construct an interpretation of λY-terms D, and a set $F \subseteq D$ s.t. for every λY-term M:

$$BT(M) \vDash \varphi \quad \text{iff} \quad [\![M]\!]^D \in F.$$

- Type systems
- Program tranformation
- Transfer theorem

- Verification by evaluation
- Abstraction/refinement
- Evaluating programs directly