A semantic study of higher-order model-checking

Charles Grellois (PhD work, joint with Paul-André Melliès)

PPS & LIAFA — Université Paris 7
University of Dundee

Dundee University — Aug 21, 2015

Model-checking higher-order programs

A well-known approach in verification: model-checking.

- Construct a model $\mathcal{M}$ of a program
- Specify a property φ in an appropriate logic
- Make them interact: the result is whether

$$\mathcal{M} \quad \vDash \quad \varphi$$

When the model is a word, a tree. . . of actions: translate φ to an equivalent automaton:

$$\varphi \quad \mapsto \quad \mathcal{A}_{\varphi}$$

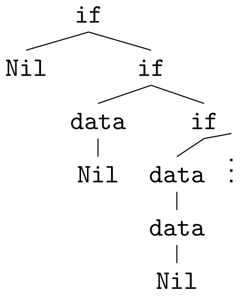
Model-checking higher-order programs

For higher-order programs with recursion (Haskell, OCaml, Javascript, Python. . .), $\mathcal{M}$ is a higher-order tree.

Example:

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen } x & = & \texttt{if } \textit{end} \texttt{ then } x \texttt{ else Listen (data } x\texttt{)} \end{array}$$

modelled as



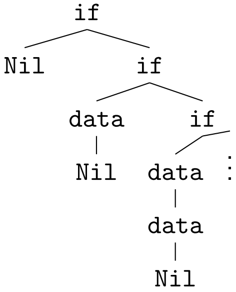
Model-checking higher-order programs

For higher-order programs with recursion (Haskell, OCaml, Javascript, Python...), $\mathcal{M}$ is a higher-order tree.

Example:

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen } x & = & \texttt{if } \mathit{end} \texttt{ then } x \texttt{ else Listen (data } x\texttt{)} \end{array}$$

modelled as



How to represent this tree finitely?

Model-checking higher-order programs

For higher-order programs with recursion (Haskell, OCaml, Javascript, Python...), $\mathcal{M}$ is a higher-order tree

over which we run

an alternating parity tree automaton (APT) $\mathcal{A}_\varphi$

corresponding to a

monadic second-order logic (MSO) formula φ.

(safety, liveness properties, etc)

Model-checking higher-order programs

For higher-order programs with recursion (Haskell, OCaml, Javascript, Python...), $\mathcal{M}$ is a higher-order tree

over which we run

an alternating parity tree automaton (APT) $\mathcal{A}_\varphi$

corresponding to a

monadic second-order logic (MSO) formula φ.

(safety, liveness properties, etc)

Can we decide whether a higher-order tree satisfies a MSO formula?

Automata theory and typing

A very naive model-checking problem

Let's simplify our model-checking problem:

- Actions of the program are modelled by a finite word
- The property to check corresponds to a finite automaton

A very naive model-checking problem

A word of actions :

$$open \ \cdot \ (read \cdot write)^2 \cdot close$$

A property to check: is every *read* immediately followed by a *write* ?

Corresponds to an automaton with two states: $Q \ = \ \{q_0, \, q_{read}\}$.

q_0 is both initial and final.

A type-theoretic intuition

The transition function may be seen as a typing of the letters of the word, seen as function symbols.

For example,

$$\delta(q_0, \, read) \quad = \quad q_{read}$$

corresponds to the typing

$$read \;\; : \;\; q_{read} \rightarrow q_0$$

The type of a word is a state from which it is accepted.

A type-theoretic intuition: a run of the automaton

$$\frac{}{\vdash open \; \cdot \; (read \cdot write)^2 \cdot close \; : \; q_0}$$

A type-theoretic intuition: a run of the automaton

$$\dfrac{\vdash open \ : \ q_0 \rightarrow q_0 \qquad \dfrac{}{\vdash \ (read \cdot write)^2 \cdot close \ : \ q_0}}{\vdash open \ \cdot \ (read \cdot write)^2 \cdot close \ : \ q_0}$$

A type-theoretic intuition: a run of the automaton

$$\dfrac{\vdash \mathit{read} \ : \ q_{read} \to q_0 \qquad \dfrac{}{\vdash \ \mathit{write} \cdot \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_{read}}}{\dfrac{\vdash \ (\mathit{read} \cdot \mathit{write})^2 \cdot \mathit{close} \ : \ q_0}{\vdots}}$$

A type-theoretic intuition: a run of the automaton

$$\dfrac{\vdash \mathit{read} \;:\; q_{read} \to q_0 \qquad \dfrac{\color{red}{\vdash \mathit{write} \;:\; q_0 \to q_{read}} \qquad \dfrac{}{\color{red}{\vdash \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \;:\; q_0}}}{\vdash \mathit{write} \cdot \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \;:\; q_{read}}}{\dfrac{\vdash (\mathit{read} \cdot \mathit{write})^2 \cdot \mathit{close} \;:\; q_0}{\vdots}}$$

and so on.

Typing naturally extends to programs computing words.

We will try to do the same for recursion schemes – which compute trees.

Automata and recognition

Recall that, given a language $L \subseteq A^*$,

there exists a finite automaton $\mathcal{A}$ recognizing L

if and only if

there exists a finite monoid M, a subset $K \subseteq M$
and a homomorphism $\phi \,:\, A^* \to M$ such that $L = \phi^{-1}(K)$.

Roughly speaking: there exists a finite algebraic structure in which the language is interpreted.

A very naive model-checking problem

The model-checking problem can be solved by:

- computing the interpretation of a word (its denotation)
- and check whether it belongs to M

Reminiscent of interpretations in logical models $\longrightarrow$ model-check terms.

Link with typing: typings compute the denotations.

A very naive model-checking problem

A more elaborate problem: what about ultimately periodic words and Büchi automata ?

Extend the monoid's behaviour with recursion (for periodicity) modelling the Büchi condition.

Or, on typings, define an acceptance condition on infinite derivations.

Higher-order recursion schemes

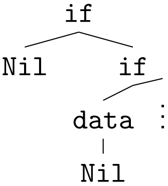
Higher-order recursion schemes

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen } x & = & \texttt{if } \textit{end} \texttt{ then } x \texttt{ else Listen (data } x\texttt{)} \end{array}$$

is abstracted as

$$\mathcal{G} = \begin{cases} \texttt{S} & = \quad \texttt{L Nil} \\ \texttt{L } x & = \quad \texttt{if } x \,(\texttt{L (data } x\,)\,) \end{cases}$$

which produces (how ?) the higher-order tree of actions



Higher-order recursion schemes

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$

Sort of deterministic higher-order grammar providing a finite representation of higher-order trees.

Rewrite rules have (higher-order) parameters.

"Everything" is simply-typed.

Rewriting produces a tree $\langle \mathcal{G} \rangle$.

Higher-order recursion schemes

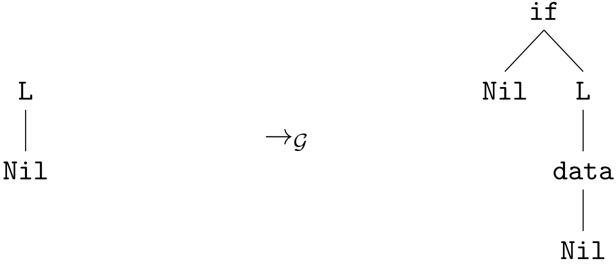
$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$

Rewriting starts from the start symbol S:

$$\mathtt{S} \qquad \rightarrow_{\mathcal{G}} \qquad \begin{array}{c} \mathtt{L} \\ | \\ \mathtt{Nil} \end{array}$$

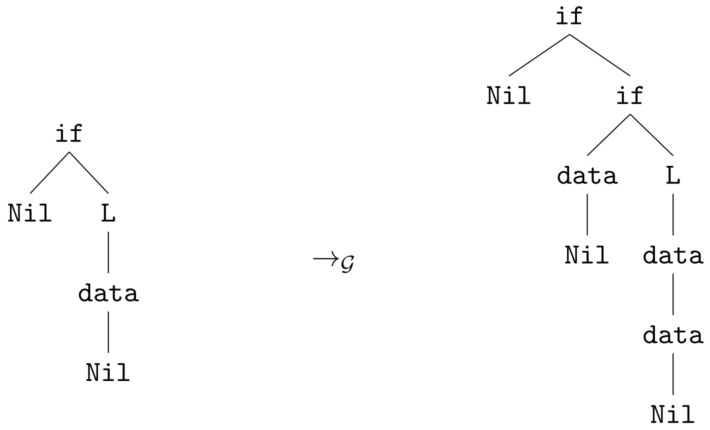
Higher-order recursion schemes

$$\mathcal{G} = \begin{cases} \mathtt{S} & = \quad \mathtt{L\ Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



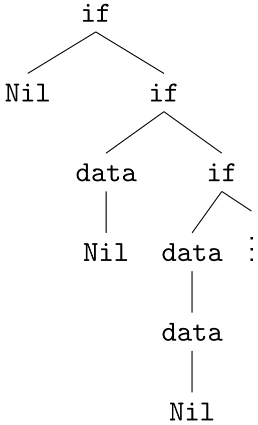
Higher-order recursion schemes

$$\mathcal{G} \;=\; \begin{cases} \mathtt{S} & = \quad \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



Higher-order recursion schemes

$$\mathcal{G} = \begin{cases} \texttt{S} & = \quad \texttt{L Nil} \\ \texttt{L}\ x & = \quad \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ )\ ) \end{cases}$$



$\langle \mathcal{G} \rangle$ is an infinite non-regular tree.

It is our model $\mathcal{M}$.

Higher-order recursion schemes

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\,(\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$

HORS can alternatively be seen as simply-typed λ-terms with

free variables of order at most 1 (= tree constructors)

and

simply-typed recursion operators $Y_\sigma \;:\; (\sigma \rightarrow \sigma) \rightarrow \sigma$.

Here : $\mathcal{G} \quad \leftrightsquigarrow \quad (Y_{o \rightarrow o}\,(\lambda \mathtt{L}.\lambda x.\mathtt{if}\ x\ \ (\mathtt{L}\,(\mathtt{data}\ x))))\ \mathtt{Nil}$

Higher-order recursion schemes

In general, many reductions could be used to compute (subsets of) $\langle \mathcal{G} \rangle$.
For (infinitary) λ-calculus, we can focus on the parallel head reduction, defined coinductively:

$$\frac{}{(\,\lambda x.\, s\,)\ t \quad \rightarrow_w \quad s[x \leftarrow t]} \qquad\qquad \frac{s \quad \rightarrow_w \quad s'}{s\ t \quad \rightarrow_w \quad s'\ t}$$

Higher-order recursion schemes

In general, many reductions could be used to compute (subsets of) $\langle \mathcal{G} \rangle$.
For (infinitary) λ-calculus, we can focus on the parallel head reduction, defined coinductively:

$$\frac{}{(\lambda x.\, s)\ t \quad \rightarrow_w \quad s[x \leftarrow t]} \qquad \frac{s \quad \rightarrow_w \quad s'}{s\ t \quad \rightarrow_w \quad s'\ t}$$

$$\frac{s \quad \rightarrow_w \quad s'}{s \quad \rightarrow_h \quad s'} \qquad \frac{s \quad \rightarrow_h \quad s'}{\lambda x.\, s \quad \rightarrow_h \quad \lambda x.\, s'}$$

Higher-order recursion schemes

In general, many reductions could be used to compute (subsets of) $\langle \mathcal{G} \rangle$.
For (infinitary) λ-calculus, we can focus on the parallel head reduction, defined coinductively:

$$\frac{}{(\,\lambda x.\, s\,)\ t \quad \to_w \quad s[x \leftarrow t]} \qquad\qquad \frac{s \quad \to_w \quad s'}{s\ t \quad \to_w \quad s'\ t}$$

$$\frac{s \quad \to_w \quad s'}{s \quad \to_h \quad s'} \qquad\qquad \frac{s \quad \to_h \quad s'}{\lambda x.\, s \quad \to_h \quad \lambda x.\, s'}$$

$$\frac{t \quad \to_h^{*} \quad \lambda x_1 \cdots \lambda x_m .\, a\ t_1\ \cdots\ t_n \qquad t_i \quad \to_h^{\infty} \quad t_i' \quad (\forall i) \qquad a \not\equiv \perp}{t \quad \to_h^{\infty} \quad \lambda x_1 \cdots \lambda x_m .\, a\ t_1'\ \cdots\ t_n'}$$

$$\frac{t \text{ has no hnf}}{t \quad \to_h^{\infty} \quad \perp}$$

Higher-order recursion schemes

We can adapt this to HORS:

$$\frac{}{(\,\lambda x.\, s\,)\ t \quad \rightarrow_{\mathcal{G}w} \quad s[x \leftarrow t]} \qquad \frac{s \quad \rightarrow_{\mathcal{G}w} \quad s'}{s\ t \quad \rightarrow_{\mathcal{G}w} \quad s'\ t}$$

$$\frac{}{F \quad \rightarrow_{\mathcal{G}w} \quad \mathcal{R}(F)}$$

$$\frac{t \quad \rightarrow^{*}_{\mathcal{G}w} \quad a\ t_1\ \cdots\ t_n \qquad t_i \quad \rightarrow^{\infty}_{\mathcal{G}} \quad t'_i \quad (\forall i)}{t \quad \rightarrow^{\infty}_{\mathcal{G}} \quad a\ t'_1\ \cdots\ t'_n}$$

and this reduction computes $\langle \mathcal{G} \rangle$ whenever it exists (a decidable question).

This presentation allows coinductive reasoning on rewriting.

Alternating tree automata

Alternating parity tree automata

For a MSO formula φ,

$$\langle \mathcal{G} \rangle \quad \vDash \quad \varphi$$

iff an equivalent APT $\mathcal{A}_\varphi$ has a run over $\langle \mathcal{G} \rangle$.

APT $\quad = \quad$ alternating tree automata (ATA) + parity condition.

Alternating tree automata

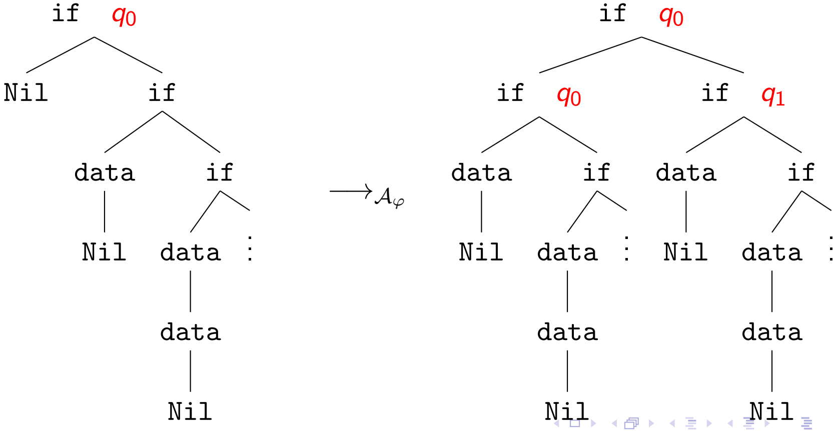
ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

Typically: $\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$.

Alternating tree automata

ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

Typically: $\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$.



Alternating tree automata

ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

Typically: $\delta(q_0, \mathtt{if}) \ = \ (2, q_0) \wedge (2, q_1)$.

This infinite process produces a run-tree of $\mathcal{A}_\varphi$ over $\langle \mathcal{G} \rangle$.

It is an infinite, unranked tree.

ATA vs. HORS

$$\frac{}{(\,\lambda x.\, s\,)\ t \quad \rightarrow_{\mathcal{G}w} \quad s[x \leftarrow t]} \qquad \frac{s \quad \rightarrow_{\mathcal{G}w} \quad s'}{s\ t \quad \rightarrow_{\mathcal{G}w} \quad s'\ t}$$

$$\frac{}{F \quad \rightarrow_{\mathcal{G}w} \quad \mathcal{R}(F)}$$

$$\frac{t \quad \rightarrow_{\mathcal{G}w}^{*} \quad a\ t_1\ t_n \qquad t_i\ :\ q_{ij} \quad \rightarrow_{\mathcal{G},\mathcal{A}}^{\infty} \quad t_i'\ :\ q_{ij}}{t\ :\ q \quad \rightarrow_{\mathcal{G},\mathcal{A}}^{\infty} \quad (a\ (t_{11}'\ :\ q_{11})\ \cdots\ (t_{nk_n}'\ :\ q_{nk_n}))\ :\ q}$$

where the duplication "conforms to δ" (there is non-determinism).

Starting from $S\ :\ q_0$, this computes run-trees of an ATA $\mathcal{A}$ over $\langle \mathcal{G} \rangle$.

We get closer to type theory. . .

Alternating tree automata and intersection types

A key remark (Kobayashi 2009):

$$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$$

can be seen as the intersection typing

$$\texttt{if} \;:\; \emptyset \rightarrow (q_0 \wedge q_1) \rightarrow q_0$$

refining the simple typing

$$\texttt{if} \;:\; o \rightarrow o \rightarrow o$$

(this talk is NOT about filter models!)

Alternating tree automata and intersection types

In a derivation typing $\texttt{if} \;\; T_1 \;\; T_2$:

$$
\dfrac{\dfrac{\dfrac{\delta}{\emptyset \vdash \texttt{if} : \emptyset \to (q_0 \wedge q_1) \to q_0}\quad \color{red}{\emptyset}}{\emptyset \vdash \texttt{if}\; T_1 : (q_0 \wedge q_1) \to q_0}\,\text{App} \qquad \dfrac{\vdots}{\color{red}{\Gamma_{21} \vdash T_2 : q_0}} \qquad \dfrac{\vdots}{\color{red}{\Gamma_{22} \vdash T_2 : q_1}}}{\Gamma_{21}, \Gamma_{22} \vdash \texttt{if}\; T_1 \; T_2 : q_0}\,\text{App}
$$

Intersection types naturally lift to higher-order – and thus to $\mathcal{G}$, which finitely represents $\langle \mathcal{G} \rangle$.

Theorem (Kobayashi)

$\emptyset \vdash S : q_0$ *iff* *the ATA* $\mathcal{A}_\varphi$ *has a run-tree over* $\langle \mathcal{G} \rangle$.

A type-system for verification: without colours

Axiom $$\frac{}{x : \bigwedge_{\{i\}} \ \theta_i \ :: \ \kappa \ \vdash \ x : \theta_i \ :: \ \kappa}$$

δ $$\frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\} \quad \text{satisfies} \quad \delta_A(q, a)}{\emptyset \vdash a \ : \ \bigwedge_{j=1}^{k_1} \ q_{1j} \ \rightarrow \ \ldots \ \rightarrow \ \bigwedge_{j=1}^{k_n} \ q_{nj} \rightarrow q \ :: \ o \rightarrow \cdots \rightarrow o}$$

App $$\frac{\Delta \vdash t : (\ \theta_1 \ \wedge \cdots \wedge \ \theta_k) \rightarrow \theta \ :: \ \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u \ : \ \theta_i \ :: \ \kappa}{\Delta \ + \ \Delta_1 \ + \ \ldots \ + \ \Delta_k \ \vdash \ \ t\, u \ : \ \theta :: \ \kappa'}$$

λ $$\frac{\Delta\, ,\, x \ : \ \bigwedge_{i \in I} \ \ \theta_i \ :: \ \kappa \ \vdash \ \ t \ : \ \theta \ :: \ \kappa'}{\Delta \ \vdash \ \lambda\, x \, . \, t \ : \ \left(\bigwedge_{i \in I} \ \ \theta_i\right) \rightarrow \theta :: \ \kappa \rightarrow \kappa'}$$

fix $$\frac{\Gamma \vdash \mathcal{R}(F) \ : \ \theta \ :: \ \kappa}{F \ : \ \ \theta \ :: \ \kappa \ \vdash \ F \ : \theta \ :: \ \kappa}$$

An alternate proof

Non-idempotent types + extension of $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$ to typing trees:

$$\dfrac{\dfrac{\begin{array}{c}\pi \\ \vdots \\ \Gamma,\, x \,:\, \bigwedge_i \tau_i \;\vdash\; s \,:\, \sigma\end{array}}{\Gamma \;\vdash\; \lambda x.\, s \,:\, \bigwedge_i \tau_i \rightarrow \sigma} \qquad \begin{array}{c}\pi_i \\ \vdots \\ \Gamma_i \;\vdash\; t \,:\, \tau_i\end{array}}{\Gamma + \sum_i \Gamma_i \;\vdash\; (\lambda x.\, s)\; t \,:\, \sigma}$$

reduces to

$$\begin{array}{c}\pi[x \leftarrow (\pi_i)_i] \\ \vdots \\ \Gamma + \sum_i \Gamma_i \;\vdash\; s[x \leftarrow t] \,:\, \sigma\end{array}$$

Note that quantitativity (= non-idempotence) makes this process linear.

An alternate proof

If we consider the infinitary λ-term $t(\mathcal{G})$ obtained by unfolding $\mathcal{G}$, we get

Theorem

$\emptyset \vdash t(\mathcal{G}) : q_0$ *iff the ATA* $\mathcal{A}_\phi$ *has a run-tree over* $\langle \mathcal{G} \rangle$.

by proving coinductively an infinitary subject reduction property, using the previous reduction for typing trees.

We can then "fold back" the result to HORS.

Models of linear logic and HOMC

Intersection types and linear logic

$$A \rightarrow B \quad = \quad !\, A \multimap B$$

A program of type $A \rightarrow B$

duplicates or drops elements of A

and then

uses linearly ($=$ once) each copy

Just as intersection types.

Intersection types and linear logic

$$A \rightarrow B \;=\; !\,A \multimap B$$

Set $[\![o]\!] = Q$. Two interpretations of the exponential modality:

Qualitative models
(Scott semantics)

$$!\,A \;=\; \mathcal{P}_{fin}(A)$$

$$[\![o \rightarrow o]\!] \;=\; \mathcal{P}_{fin}(Q) \times Q$$

$$\{q_0,\, q_0,\, q_1\} \;=\; \{q_0,\, q_1\}$$

Order closure

Quantitative models
(Relational semantics)

$$!\,A \;=\; \mathcal{M}_{fin}(A)$$

$$[\![o \rightarrow o]\!] \;=\; \mathcal{M}_{fin}(Q) \times Q$$

$$[q_0,\, q_0,\, q_1] \;\neq\; [q_0,\, q_1]$$

Unbounded multiplicities

Intersection types and linear logic

Models of linear logic and intersection types (refining simple types):

$$
\begin{tikzcd}
Rel & & Rel_! \arrow[ll, tail] \arrow[d, dotted] \arrow[rr, leftrightarrow, "Bucciareli-Ehrhard", "de\ \ Carvalho"'] & & \text{Non-idempotent types} \arrow[d, "Ehrhard,\ G-M"] \\
Scott \arrow[from=u, "Ehrhard"'] & & Scott_! \arrow[ll, tail] \arrow[rr, leftrightarrow, "Terui"'] & & \text{Idempotent types}
\end{tikzcd}
$$

Fundamental idea: derivations of the intersection type systems compute denotations in the associated model.

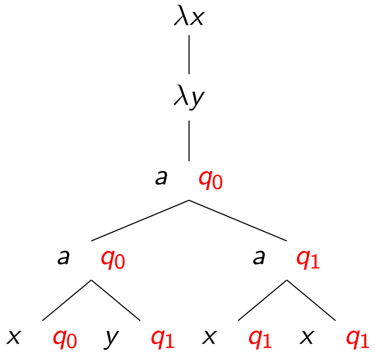
Intersection types and linear logic

Models of linear logic and intersection types (refining simple types):

$$\begin{tikzcd}
Rel \arrow[d, "Ehrhard"'] & Rel_! \arrow[l, tail] \arrow[d, dotted] \arrow[r, leftrightarrow, "Bucciareli-Ehrhard", "de\ \ Carvalho"'] & \text{Non-idempotent types} \arrow[d, "{Ehrhard,\ G-M}"] \\
Scott & Scott_! \arrow[l, tail] \arrow[r, leftrightarrow, "Terui"'] & \text{Idempotent types}
\end{tikzcd}$$

$$\begin{tikzcd}
{[q_0,\ q_0,\ q_1] \multimap q_0} \arrow[r, mapsto] \arrow[d, mapsto] & q_0 \wedge q_0 \wedge q_1 \to q_0 \arrow[d, mapsto] \\
\{q_0,\ q_1\} \multimap q_0 \arrow[r, mapsto] & q_0 \wedge q_1 \to q_0
\end{tikzcd}$$

An example of interpretation



will be interpreted in the model as

$$([q_0, q_1, q_1], [q_1], q_0)$$

Quantitative semantics and tree automata

If we add an inductive fixpoint operator to the relational semantics, we get:

Theorem (Grellois-Melliès)

In the relational semantics,

$$q_0 \in [\![\mathcal{G}]\!] \quad \textit{iff} \quad \text{the ATA } \mathcal{A}_\phi \text{ has a finite run-tree over } \langle \mathcal{G} \rangle.$$

Also true with qualitative semantics.

An infinitary model of linear logic

Restrictions to finiteness: lack of a countable multiplicity ω.

Indeed, we consider tree constructors as free variables.

In *Rel*, we introduce a new exponential $A \mapsto \lightning\, A$ s.t.

$$[\![\lightning\, A]\!] \quad = \quad \mathcal{M}_{count}([\![A]\!])$$

(finite-or-countable multisets)

An infinitary model of linear logic

This defines an infinitary model of linear logic, which corresponds to

non-idempotent intersection types with countable multiplicities

and derivations of countable depth.

It admits a coinductive fixpoint, which we use to interpret Y.

Theorem (Grellois-Melliès)

In the infinitary relational semantics,

$$q_0 \in [\![\mathcal{G}]\!] \quad \textit{iff} \quad \text{the ATA } \mathcal{A}_\phi \text{ has a run-tree over } \langle \mathcal{G} \rangle.$$

Parity conditions

Alternating parity tree automata

MSO allows to discriminate inductive from coinductive behaviour.

This allows to express properties as

"a given operation is executed infinitely often in some execution"

or

"after a `read` operation, a `write` eventually occurs".

Alternating parity tree automata

Each state of an APT is attributed a color

$$\Omega(q) \in Col \subseteq \mathbb{N}$$

An infinite branch of a run-tree is winning iff the maximal color among the ones occuring infinitely often along it is even.

A run-tree is winning iff all its infinite branches are.

For a MSO formula φ:

$$\mathcal{A}_\varphi \text{ has a winning run-tree over } \langle \mathcal{G} \rangle \text{ iff } \langle \mathcal{G} \rangle \vDash \phi$$

Alternating parity tree automata

We add coloring informations to intersection types:

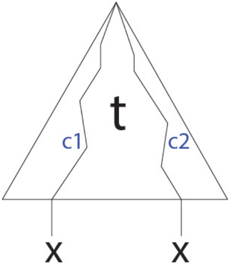
$$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$$

now corresponds to

$$\texttt{if} \;:\; \emptyset \rightarrow \left(\Box_{\Omega(q_0)}\, q_0 \wedge \Box_{\Omega(q_1)}\, q_1\right) \rightarrow q_0$$

Application computes the "local" maximum of colors, and the fixpoint deals with the acceptance condition.

Parity conditions



In this setting, t has some type $\square_{c_1}\ \sigma_1 \wedge \square_{c_2}\ \sigma_2 \rightarrow \tau$.

The color labelling each occurence is the maximal color leading to it in the normal form of t.

Proof: by studying the reduction of colored proof-trees.

A type-system for verification (Grellois-Melliès 2014)

Axiom $$\frac{}{x \,:\, \bigwedge_{\{i\}} \Box_{\epsilon}\ \theta_i \ :: \ \kappa \ \vdash \ x \,:\, \theta_i \ :: \ \kappa}$$

δ $$\frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\,\} \quad \text{satisfies} \quad \delta_A(q, a)}{\emptyset \vdash a \,:\, \bigwedge_{j=1}^{k_1} \Box_{\Omega(q_{1j})}\ q_{1j} \ \rightarrow \ \ldots \ \rightarrow \ \bigwedge_{j=1}^{k_n} \Box_{\Omega(q_{nj})}\ q_{nj} \rightarrow q \ :: \ o \rightarrow \cdots \rightarrow o \rightarrow o}$$

App $$\frac{\Delta \vdash t : (\Box_{m_1}\ \theta_1 \ \wedge \cdots \wedge \Box_{m_k}\ \theta_k) \rightarrow \theta \ :: \ \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u \,:\, \theta_i \ :: \ \kappa}{\Delta \,+\, \Box_{m_1}\Delta_1 \,+\, \ldots \,+\, \Box_{m_k}\Delta_k \ \vdash \ t\, u \,:\, \theta :: \ \kappa'}$$

fix $$\frac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \ :: \ \kappa}{F \,:\, \Box_{\epsilon}\ \theta \ :: \ \kappa \ \vdash \ F \,:\, \theta \ :: \ \kappa}$$

λ $$\frac{\Delta\,,\, x \,:\, \bigwedge_{i \in I} \Box_{m_i}\ \theta_i \ :: \ \kappa \ \vdash \ t \,:\, \theta \ :: \ \kappa'}{\Delta \ \vdash \ \lambda x \,.\, t \,:\, \left(\bigwedge_{i \in I} \Box_{m_i}\ \theta_i\right) \rightarrow \theta :: \ \kappa \rightarrow \kappa'}$$

A type-system for verification (Grellois-Melliès 2014)

This type system can have infinite-depth derivations, over which we recast the parity condition.

Each infinite branch of a run-tree over $\langle \mathcal{G} \rangle$ corresponds to an infinite branch of the associated typing tree π, and is computed by an infinite reduction.

Infinite branches of π have infinitely many occurences F_i of non-terminals. The head normalization of F_i produces $C_i[F_{i+1}]$ in finitely many steps.

The maximal color on the path from the root of C_i to F_{i+1} is the same as the one from F_i to F_{i+1} in π. It is ϵ iff C_i is empty.

Non-terminals allow to conveniently factor the colors along a branch.

Theorem (G.-Melliès 2014, see also Kobayashi-Ong 2009)

$S : q_0 \vdash S : q_0$ admits a winning typing derivation iff $\mathcal{A}$ accepts $\langle \mathcal{G} \rangle$.

The coloring comonad

Our work shows that coloring is a modality. It defines a comonad in the semantics:

$$\square\, A \quad = \quad Col \times A$$

which can be composed with $\lightning$, so that

$$\texttt{if} \;\; : \;\; \emptyset \rightarrow \left(\square_{\Omega(q_0)}\, q_0 \wedge \square_{\Omega(q_1)}\, q_1 \right) \rightarrow q_0$$

corresponds to

$$[\;] \multimap [(\Omega(q_0),\, q_0), (\Omega(q_1),\, q_1)] \multimap q_0 \quad \in \quad [\![\texttt{if}]\!]$$

in the semantics.

An inductive-coinductive fixpoint operator

We define an inductive-coinductive fixpoint operator on denotations, which composes inductively or coinductively elements of the semantics, according to the current color.

Theorem (G.-Melliès 2015)

$\mathcal{A}$ accepts $\langle \mathcal{G} \rangle$ iff $q_0 \in [\![\mathcal{G}]\!]$.

But this model is infinitary, how to get decidability?

The final picture

$$\begin{tikzcd}
Rel_{\lightning} + \square + Y \arrow[r, leftrightarrow] \arrow[d, dotted] & \text{Non-idempotent types} + \square + Y \arrow[d, dotted] \\
Scott_{\lightning} + \square + Y \arrow[r, leftrightarrow] & \text{Idempotent types} + \square + Y
\end{tikzcd}$$

Ehrhard 2012: "collapsing" *Rel* by forgetting multiplicities gives *Scott*.

We can enrich Scott with a coloring modality and a fixpoint operator, and get the same result of HOMC.

We obtain decidability, by finiteness of the semantics.

The selection problem

Even better: the selection problem is decidable.

If $\mathcal{A}_\phi$ accepts $\langle \mathcal{G} \rangle$, we can compute effectively a new scheme $\mathcal{G}'$ such that $\langle \mathcal{G}' \rangle$ is a winning run-tree of $\mathcal{A}_\phi$ over $\langle \mathcal{G} \rangle$.

In other words: there is a higher-order winning run-tree.

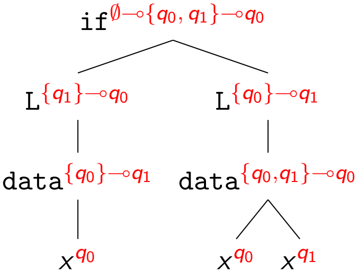
(the key: annotate the rules with their denotation/their types).

The selection problem

$$\begin{cases} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L} & = & \lambda x.\,\texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x)) \end{cases}$$

becomes e.g.

$$\begin{cases} \texttt{S}^{q_0} & = & \texttt{L}^{\{q_0,\,q_1\}\multimap q_0}\ \texttt{Nil}^{q_0}\ \texttt{Nil}^{q_1} \\ \texttt{L}^{\{q_0,\,q_1\}\multimap q_0} & = & \lambda x^{\{q_0,\,q_1\}}. \\ \texttt{L}^{\{q_0\}\multimap q_1} & = & \cdots \\ \texttt{L}^{\{q_1\}\multimap q_0} & = & \cdots \end{cases}$$



Conclusion

- Sort of static analysis of infinitary properties.
- We lift to higher-order the behavior of APT.
- Coloring is a modality, stable by reduction in some sense, and can therefore be added to models and type systems.
- In finitary semantics, we obtain decidability of HOMC and of the selection problem.

Thank you for your attention!

Conclusion

- Sort of static analysis of infinitary properties.
- We lift to higher-order the behavior of APT.
- Coloring is a modality, stable by reduction in some sense, and can therefore be added to models and type systems.
- In finitary semantics, we obtain decidability of HOMC and of the selection problem.

Thank you for your attention!