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 $\varphi$ in an appropriate logic
- Make them interact: the result is whether

$$\mathcal{M} \models \varphi$$

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

$$\varphi \rightarrow A_\varphi$$
Model-checking higher-order programs

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

Example:

\[
\begin{align*}
\text{Main} & = \text{Listen Nil} \\
\text{Listen } x & = \text{if } \text{end then } x \text{ else Listen (data } x)\end{align*}
\]

modelled as

```
if
  Nil
  if
    data
      if
        Nil
        data
          data
            data
                data
                    data
                        nil
```
Model-checking higher-order programs

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

Example:

\[
\begin{align*}
\text{Main} & = \text{Listen Nil} \\
\text{Listen } x & = \text{if } \text{end} \text{ then } x \text{ else Listen (data } x) \\
\end{align*}
\]

modelled as

\[
\text{if} \quad \text{if} \\
\text{Nil} \quad \text{data} \quad \text{if} \\
\text{data} \quad \text{Nil} \quad \text{data} \\
\quad \quad \quad \text{Nil}
\]

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

\[ \text{an alternating parity tree automaton (APT) } A_\varphi \]

corresponding to a

\[ \text{monadic second-order logic (MSO) formula } \varphi. \]

(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) $A_\varphi$ corresponding to a monadic second-order logic (MSO) formula $\varphi$.

(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 \textit{read} immediately followed by a \textit{write}?

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

\( q_0 \) 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,

\[ \delta(q_0, \text{read}) = q_{\text{read}} \]

corresponds to the typing

\[ \text{read} : q_{\text{read}} \rightarrow q_0 \]

The type of a word is a state from which it is accepted.
A type-theoretic intuition: a run of the automaton

\[ \vdash \text{open} \cdot (\text{read} \cdot \text{write})^2 \cdot \text{close} : q_0 \]
A type-theoretic intuition: a run of the automaton

\[
\begin{align*}
\vdash \text{open} : q_0 \rightarrow q_0 & \quad \vdash (\text{read} \cdot \text{write})^2 \cdot \text{close} : q_0 \\
\hline
\vdash \text{open} \cdot (\text{read} \cdot \text{write})^2 \cdot \text{close} : q_0
\end{align*}
\]
A type-theoretic intuition: a run of the automaton

\[ \vdash \text{read} : q_{\text{read}} \rightarrow q_0 \quad \vdash \text{write} \cdot \text{read} \cdot \text{write} \cdot \text{close} : q_{\text{read}} \]

\[ \vdash (\text{read} \cdot \text{write})^2 \cdot \text{close} : q_0 \]

\[ \vdots \]
A type-theoretic intuition: a run of the automaton

\[
\begin{align*}
\vdash \text{read} : q_0 &\rightarrow q_{\text{read}} \\
\vdash \text{write} : q_0 &\rightarrow q_{\text{read}} \\
\vdash \text{read} \cdot \text{write} \cdot \text{close} : q_0 \\
\vdash \text{write} \cdot \text{read} \cdot \text{write} \cdot \text{close} : q_{\text{read}} \\
\vdash (\text{read} \cdot \text{write})^2 \cdot \text{close} : q_0 \\
\vdash \ldots
\end{align*}
\]

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 $A$ recognizing $L$

if and only if

there exists a finite monoid $M$, a subset $K \subseteq M$
and a homomorphism $\phi : A^* \rightarrow M$ 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 $\rightarrow$ 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

\[
\text{Main} \quad = \quad \text{Listen \ Nil}
\]
\[
\text{Listen \ } x \quad = \quad \text{if \ end \ then \ } x \ \text{else \ Listen \ (data \ } x \text{)}
\]

is abstracted as

\[
G = \begin{cases} 
S &= \text{L \ Nil} \\
\text{L} \ \text{x} &= \text{if} \ \text{x} \ (\text{L \ (data \ } x \text{)})
\end{cases}
\]

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

![Diagram](image-url)
Higher-order recursion schemes

\[ G = \begin{cases} 
  S & = & L \text{ Nil} \\
  L \times & = & \text{if } x (L \text{ (data } x \text{ )}) 
\end{cases} \]

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 \( \langle G \rangle \).
Higher-order recursion schemes

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

Rewriting starts from the start symbol $S$:

$$S \rightarrow_G L \ Nil$$
Higher-order recursion schemes

\[ G = \begin{cases} 
  S & = & L \text{ Nil} \\
  L \ x & = & \text{if } x (L \ (\text{data } x)) 
\end{cases} \]
Higher-order recursion schemes

\[ G = \begin{cases} 
S &= L \text{ Nil} \\
L \times &= \text{if } x (L \text{ (data } x \text{ )}) 
\end{cases} \]

\[ \text{if} \\
\text{Nil} \quad \text{if} \\
\text{data} \quad \text{L} \\
\text{data} \quad \text{data} \\
\text{Nil} \quad \text{Nil} \]
Higher-order recursion schemes

\[ \mathcal{G} = \begin{cases} 
    S & = \text{L Nil} \\
    \text{L } x & = \text{if } x (\text{L (data } x )) 
\end{cases} \]

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

It is our model \( \mathcal{M} \).
Higher-order recursion schemes

\[ G = \begin{cases} 
S &= L \text{ Nil} \\
L \ x &= \text{if } x (L (\text{data } x)) 
\end{cases} \]

HORS can alternatively be seen as simply-typed \( \lambda \)-terms with

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

and

simply-typed recursion operators \( Y_\sigma : (\sigma \to \sigma) \to \sigma \).

Here: \( G \leftrightarrow (Y_{o\to o} (\lambda L.\lambda x.\text{if } x (L (\text{data } x)))) \text{ Nil} \)
Higher-order recursion schemes

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

\[
(\lambda x. s) t \rightarrow_w s[x \leftarrow t]
\]

\[
\frac{s \rightarrow_w s'}{s t \rightarrow_w s' t}
\]
Higher-order recursion schemes

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

\[
\begin{align*}
(\lambda x. s) t & \rightarrow_w s[x \leftarrow t] \\
\lambda x. s & \rightarrow_h \lambda x. s'
\end{align*}
\]

\[
\begin{align*}
s & \rightarrow_w s' \\
\lambda x. s & \rightarrow_h \lambda x. s'
\end{align*}
\]

\[
\begin{align*}
s & \rightarrow_w s' \\
s t & \rightarrow_w s' t
\end{align*}
\]

\[
\begin{align*}
s t & \rightarrow_w s' t
\end{align*}
\]
Higher-order recursion schemes

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

\[
\begin{align*}
( \lambda x. s ) t & \rightarrow_w s[x \leftarrow t] \\
( \lambda x. s ) t & \rightarrow_h s'
\end{align*}
\]

\[
\begin{align*}
\frac{s \rightarrow_w s'}{s} & \rightarrow_h s' \\
\frac{s t \rightarrow_w s' t}{s t} & \rightarrow_h s' t
\end{align*}
\]

\[
\begin{align*}
\frac{s \rightarrow_h s'}{\lambda x. s} & \rightarrow_h \lambda x. s'
\end{align*}
\]

\[
\begin{align*}
t & \rightarrow^* \lambda x_1 \cdots \lambda x_m. a t_1 \cdots t_n \\
t_i & \rightarrow^\infty t'_i \quad (\forall i) \\
t & \rightarrow^\infty \lambda x_1 \cdots \lambda x_m. a t'_1 \cdots t'_n
\end{align*}
\]

\[
\begin{align*}
t & \text{has no hnf} \\
t & \rightarrow^\infty \bot
\end{align*}
\]
Higher-order recursion schemes

We can adapt this to HORS:

\[
\begin{array}{c}
(\lambda x. s) \; t \; \rightarrow_{gw} \; s[x \leftarrow t] \\
\hline
s \; t \; \rightarrow_{gw} \; s' \; t
\end{array}
\]

\[
F \; \rightarrow_{gw} \; R(F)
\]

\[
\begin{array}{c}
t \; \rightarrow_{gw}^* \; a \; t_1 \cdots t_n \; \hline \\
\hline
\begin{array}{c}
t_i \; \rightarrow_{gw}^\infty \; t'_i \; (\forall i) \\
\hline
\end{array}
\end{array}
\]

\[
t \; \rightarrow_{gw}^\infty \; a \; t'_1 \cdots t'_n
\]

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

This presentation allows coinductive reasoning on rewriting.
Alternating tree automata
Alternating parity tree automata

For a MSO formula \( \varphi \),

\[ \langle G \rangle \models \varphi \]

iff an equivalent APT \( A_\varphi \) has a run over \( \langle G \rangle \).

\[ \text{APT} = \text{alternating tree automata (ATA) + parity condition.} \]
Alternating tree automata

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

Typically: \( \delta(q_0, \text{if}) = (2, q_0) \land (2, q_1) \).
Alternating tree automata

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

Typically: $\delta(q_0, \text{if}) = (2, q_0) \land (2, q_1)$.
Alternating tree automata

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

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

This infinite process produces a run-tree of $A_\varphi$ over $\langle G \rangle$.

It is an infinite, unranked tree.
ATA vs. HORS

\[(\lambda x. s) \ t \ \rightarrow_{Gw} \ s[x \leftarrow t]\]

\[s \rightarrow_{Gw} \ s' \ t\]

\[F \rightarrow_{Gw} \ \mathcal{R}(F)\]

\[t \rightarrow_{Gw}^* \ a \ t_1 \ t_n \quad t_i : q_{ij} \rightarrow_{Gw}^\infty \ A \quad t_i' : q_{ij}\]

\[t : q \rightarrow_{Gw}^\infty \ A \quad (a \ (t_{11} : q_{11}) \cdots (t_{nk_n} : q_{nk_n})) : q\]

where the duplication “conforms to \(\delta\)” (there is non-determinism).

Starting from \(S : q_0\), this computes run-trees of an ATA \(A\) over \(\langle G \rangle\).

We get closer to type theory...
Alternating tree automata and intersection types

A key remark (Kobayashi 2009):

\[ \delta(q_0, \text{if}) = (2, q_0) \land (2, q_1) \]

can be seen as the intersection typing

\[ \text{if} : \emptyset \rightarrow (q_0 \land q_1) \rightarrow q_0 \]

refining the simple typing

\[ \text{if} : o \rightarrow o \rightarrow o \]

(this talk is NOT about filter models!)
Alternating tree automata and intersection types

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

\[
\begin{align*}
\delta & \quad \frac{\emptyset \vdash \text{if} : \emptyset \rightarrow (q_0 \land q_1) \rightarrow q_0}{\emptyset} \\
\text{App} & \quad \frac{\emptyset \vdash \text{if } T_1 : (q_0 \land q_1) \rightarrow q_0}{\emptyset} \\
\text{App} & \quad \frac{\Gamma_{21} \vdash T_2 : q_0}{\Gamma_{21}, \Gamma_{22} \vdash \text{if } T_1 \ T_2 : q_0} \\
\Gamma_{22} & \vdash T_2 : q_1
\end{align*}
\]

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 $A_\varphi$ has a run-tree over $\langle \mathcal{G} \rangle$. 
A type-system for verification: without colours

Axiom

\[ x : \bigwedge_{i} \theta_i :: \kappa \vdash x : \theta_i :: \kappa \]

\[
\delta
\]

\[ \{ (i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i \} \text{ satisfies } \delta_A(q, a) \]

\[ \emptyset \vdash a : \bigwedge_{j=1}^{k_1} q_{1j} \rightarrow \cdots \rightarrow \bigwedge_{j=1}^{k_n} q_{nj} \rightarrow q :: o \rightarrow \cdots \rightarrow o \]

App

\[ \Delta \vdash t : (\theta_1 \land \cdots \land \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \]

\[ \Delta_i \vdash u : \theta_i :: \kappa \]

\[ \Delta + \Delta_1 + \cdots + \Delta_k \vdash t \ u : \theta :: \kappa' \]

\lambda

\[ \Delta, x : \bigwedge_{i \in I} \theta_i :: \kappa \vdash t : \theta :: \kappa' \]

\[ \Delta \vdash \lambda x. t : (\bigwedge_{i \in I} \theta_i) \rightarrow \theta :: \kappa \rightarrow \kappa' \]

fix

\[ \Gamma \vdash R(F) : \theta :: \kappa \]

\[ \Gamma \vdash F : \theta :: \kappa \vdash F : \theta :: \kappa \]
An alternate proof

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

\[
\begin{array}{c}
\pi \\
\vdots \\
\Gamma, x : \bigwedge_i \tau_i \vdash s : \sigma \\
\end{array}
\quad 
\begin{array}{c}
\vdots \\
\Gamma \vdash \lambda x. s : \bigwedge_i \tau_i \rightarrow \sigma \\
\Gamma_i \vdash t : \tau_i \\
\end{array}
\quad 
\begin{array}{c}
\Gamma + \sum_i \Gamma_i \vdash (\lambda x. s) t : \sigma \\
\end{array}
\]

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 $\lambda$-term $t(G)$ obtained by unfolding $G$, we get

**Theorem**

$\emptyset \vdash t(G) : q_0$ iff the ATA $A_\phi$ has a run-tree over $\langle G \rangle$.

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)

\[
! A = \mathcal{P}_{\text{fin}}(A) \\
[o \rightarrow o] = \mathcal{P}_{\text{fin}}(Q) \times Q \\
\{q_0, q_0, q_1\} = \{q_0, q_1\}
\]

**Quantitative models** (Relational semantics)

\[
! A = \mathcal{M}_{\text{fin}}(A) \\
[o \rightarrow o] = \mathcal{M}_{\text{fin}}(Q) \times Q \\
[q_0, q_0, q_1] \neq [q_0, q_1]
\]

Order closure

Unbounded multiplicities

Charles Grellois (PPS - LIAFA - Dundee)  
Semantics and model-checking  
Aug 21, 2015  
30 / 48
Intersection types and linear logic

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

\[ \text{Rel} \leftarrow \text{Rel}_! \leftarrow \text{Rel}_! \leftarrow \text{Rel} \]

\[ \text{Ehrhard} \quad \downarrow \quad \text{Ehrhard} \quad \downarrow \quad \text{Ehrhard, G-M} \]

\[ \text{Scott} \leftarrow \text{Scott}_! \leftarrow \text{Scott}_! \leftarrow \text{Scott} \]

\[ \text{Bucciarelli–Ehrhard} \quad \downarrow \quad \text{de Carvalho} \]

\[ \text{Non-idempotent types} \]

\[ \text{Idempotent types} \]

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):

\[
\begin{align*}
& \text{Rel} & \text{Rel}_! & \text{Non-idempotent types} \\
Ehrhard & \text{Bucciareli–Ehrhard} & \text{de Carvalho} & \text{Ehrhard, G–M} \\
& \text{Scott} & \text{Scott}_! & \text{Idempotent types} \\
\end{align*}
\]

\[
\begin{align*}
& [q_0, q_0, q_1] \rightarrow q_0 \rightarrow q_0 \land q_0 \land q_1 \rightarrow q_0 \\
& \{q_0, q_1\} \rightarrow q_0 \rightarrow q_0 \land q_1 \rightarrow q_0
\end{align*}
\]
An example of interpretation

\[ \lambda x \lambda y \]
\[ a \ q_0 \]
\[ a \ q_0 \quad a \ q_1 \]
\[ x \ q_0 \ y \ q_1 \quad x \ q_1 \ x \ q_1 \]

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-Mellieès)**

In the relational semantics,

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

Also true with qualitative semantics.
An infinitary model of linear logic

Restrictions to finiteness: lack of a countable multiplicity $\omega$.

Indeed, we consider tree constructors as free variables.

In $Rel$, we introduce a new exponential $A \mapsto \not\in A$ s.t.

$$\llbracket \not\in A \rrbracket = \mathcal{M}_{\text{count}}(\llbracket A \rrbracket)$$

(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-Mellïès)

In the infinitary relational semantics,

$q_0 \in \llbracket G \rrbracket$ iff the ATA $A_\phi$ has a run-tree over $\langle G \rangle$. 

Charles Grellois (PPS - LIAFA - Dundee)
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

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

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

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

For a MSO formula $\varphi$:

$$A_\varphi \text{ has a winning run-tree over } \langle G \rangle \text{ iff } \langle G \rangle \models \varphi$$
Alternating parity tree automata

We add coloring informations to intersection types:

$$\delta(q_0, \text{if}) = (2, q_0) \land (2, q_1)$$

now corresponds to

$$\text{if} : \emptyset \rightarrow (\Box_\Omega(q_0) \, q_0 \land \Box_\Omega(q_1) \, q_1) \rightarrow q_0$$

Application computes the “local” maximum of colors, and the fixpoint deals with the acceptance condition.
In this setting, $t$ has some type $\square_{c_1} \sigma_1 \land \square_{c_2} \sigma_2 \rightarrow \tau$.

The color labelling each occurrence 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)

\[ x : \bigwedge_{\{i\}} \Box e I_i : \kappa \vdash x : \theta_i : \kappa \]

\[ \{ (i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i \} \text{ satisfies } \delta_A(q, a) \]

\[ \emptyset \vdash a : \bigwedge_{j=1}^{kn} \Box \Omega(q_{nj}) q_{nj} \rightarrow \cdots \rightarrow q :: o \rightarrow \cdots \rightarrow o \rightarrow o \]

\[ \Delta \vdash t : (\Box m_1 \theta_1 \land \cdots \land \Box m_k \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa \]

\[ \Delta + \Box m_1 \Delta_1 + \cdots + \Box m_k \Delta_k \vdash tu : \theta :: \kappa' \]

\[ \Gamma \vdash \mathcal{R}(F) : \theta :: \kappa \]

\[ \text{fix} \quad F : \Box e \theta :: \kappa \vdash F : \theta :: \kappa \]

\[ \Delta, x : \bigwedge_{i \in I} \Box m_i \theta_i :: \kappa \vdash t : \theta :: \kappa' \]

\[ \Delta \vdash \lambda x . t : (\bigwedge_{i \in I} \Box m_i \theta_i) \rightarrow \theta :: \kappa \rightarrow \kappa' \]
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 $\langle G \rangle$ corresponds to an infinite branch of the associated typing tree $\pi$, and is computed by an infinite reduction.

Infinite branches of $\pi$ have infinitely many occurrences $F_i$ 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 $\pi$. It is $\epsilon$ 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 $A$ accepts $\langle G \rangle$. 
The coloring comonad

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

\[ \square A = Col \times A \]

which can be composed with \( \otimes \), so that

\[ \text{if} : \emptyset \rightarrow (\square_{\Omega(q_0)} q_0 \land \square_{\Omega(q_1)} q_1) \rightarrow q_0 \]

corresponds to

\[ [\text{]} \rightarrow [(\Omega(q_0), q_0), (\Omega(q_1), q_1)] \rightarrow q_0 \in [\text{if}] \]

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)**

\[ A \text{ accepts } \langle G \rangle \text{ iff } q_0 \in \llbracket G \rrbracket. \]

But this model is infinitary, how to get decidability?
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 \textit{decidability}, by \textit{finiteness} of the semantics.
The selection problem

Even better: the selection problem is decidable.

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

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

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

\[
\begin{cases}
S &= L \text{ Nil} \\
L &= \lambda x. \text{if } x (L \text{ (data } x))
\end{cases}
\]

becomes e.g.

\[
\begin{cases}
S^{q_0} &= L^{\{q_0, q_1\} \xrightarrow{\circ} q_0} \text{ Nil } q_0 \text{ Nil } q_1 \\
L^{\{q_0\} \xrightarrow{\circ} q_1} &= \lambda x^{\{q_0, q_1\}}.
\end{cases}
\]
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!