A semantic study of higher-order model-checking

Charles Grellois  Paul-André Melliès

PPS & LIAFA — Université Paris 7
University of Dundee

University of Aarhus
November 2nd, 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:

$$\varphi \mapsto A\varphi$$
Model-checking higher-order programs

For higher-order programs with recursion, $M$ is a higher-order tree.

Example:

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

modelled as

```
if
  if
    data if
      Nil data :
        Nil data |
          Nil |
            data |
              Nil
```
Model-checking higher-order programs

For higher-order programs with recursion, $\mathcal{M}$ is a higher-order tree.

Example:

```
Main = Listen Nil
Listen x = if end then x else Listen (data x)
```

modelled as

```
if
  if
    data
      if
        Nil
        data
            if
              Nil
              data
                  if
                    Nil
    data
        if
          Nil
```

How to represent this tree finitely?
Model-checking higher-order programs

For higher-order programs with recursion, $M$ 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?
Higher-order recursion schemes

Some regularity for infinite trees
Higher-order recursion schemes

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

is abstracted as

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

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

```
if
    Nil   if
        data:
            !
                Nil
```
Higher-order recursion schemes

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

Rewriting starts from the start symbol \( S \):

\[ S \rightarrow_{\mathcal{G}} L \text{ Nil} \]
Higher-order recursion schemes

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

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

\[ G = \left\{ \begin{array}{l}
S = L \text{ Nil} \\
L \times = \text{if } x (L (\text{data } x))
\end{array} \right. \]

\[ \langle G \rangle = \text{if} \]

\[ \begin{array}{c}
\text{Nil} \\
\text{if} \\
\text{data} \\
\text{if} \\
\text{Nil} \\
\text{data} \\
\text{Nil}
\end{array} \]

Charles Grellois (PPS - LIAFA - Dundee)
Higher-order recursion schemes

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

“Everything” is simply-typed, and

well-typed programs can’t go too wrong:

we can detect productivity, and enforce it (replace divergence by outputing a distinguished symbol \( \Omega \) in one step).
Higher-order recursion schemes

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

“Everything” is simply-typed, and

well-typed programs can’t go too wrong:

we can detect productivity, and enforce it (replace divergence by outputing a distinguished symbol Ω in one step).

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

simply-typed recursion operators \( Y_\sigma : (\sigma \rightarrow \sigma) \rightarrow \sigma \).
Higher-order recursion schemes

We can adapt to HORS the fact that coinductive parallel head reduction computes the normal form of infinite λ-terms:

\[(\lambda x. s) t \rightarrow_{g_w} s[x \leftarrow t] \quad s \rightarrow_{g_w} s' \quad s t \rightarrow_{g_w} s' t\]

\[F \rightarrow_{g_w} R(F)\]

\[t \rightarrow^*_{g_w} a \ t_1 \cdots \ t_n \quad t_i \rightarrow^\infty_{g} t'_i \quad (\forall i) \quad \frac{t \rightarrow^\infty_{g} a \ t'_1 \cdots \ t'_n}{t \rightarrow^\infty_{g} a \ t'_1 \cdots \ t'_n}\]

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

This presentation allows coinductive reasoning on rewriting.
Modal $\mu$-calculus and alternating parity tree automata
Modal $\mu$-calculus

Over trees we may use several logics: CTL, MSO,…

In this work we use modal $\mu$-calculus. It is equivalent to MSO over trees.

Grammar: $\phi, \psi ::= X | a | \phi \lor \psi | \phi \land \psi | \Box \phi | \Diamond_i \phi | \mu X. \phi | \nu X. \phi$
Modal $\mu$-calculus

**Grammar:**

\[
\phi, \psi ::= X \mid a \mid \phi \lor \psi \mid \phi \land \psi \mid \Box \phi \mid \Diamond_i \phi \mid \mu X. \phi \mid \nu X. \phi
\]

$X$ is a variable

$a$ is a predicate corresponding to a symbol of $\Sigma$

$\Box \phi$ means that $\phi$ should hold on every successor of the current node

$\Diamond_i \phi$ means that $\phi$ should hold on one successor of the current node (in direction $i$)

We can also define (variant) $\Diamond = \bigvee_i \Diamond_i$. 
Modal $\mu$-calculus

Grammar: $\phi, \psi ::= X | a | \phi \lor \psi | \phi \land \psi | \square \phi | \Diamond i \phi | \mu X. \phi | \nu X. \phi$

$\mu X. \phi$ is the least fixpoint of $\phi(X)$. It is computed by expanding finitely the formula:

$$\mu X. \phi(X) \rightarrow \phi(\mu X. \phi(X)) \rightarrow \phi(\phi(\mu X. \phi(X)))$$
Modal $\mu$-calculus

Grammar:  $\phi$, $\psi$ ::= $X$ | $a$ | $\phi \lor \psi$ | $\phi \land \psi$ | $\Box \phi$ | $\Diamond i \phi$ | $\mu X. \phi$ | $\nu X. \phi$

$\nu X. \phi$ is the greatest fixpoint of $\phi(X)$. It is computed by expanding infinitely the formula:

$$
\nu X. \phi(X) \quad \rightarrow \quad \phi(\nu X. \phi(X)) \quad \rightarrow \quad \phi(\phi(\nu X. \phi(X)))
$$
Modal $\mu$-calculus

Grammar: \( \phi, \psi ::= X \mid a \mid \phi \lor \psi \mid \phi \land \psi \mid \Box \phi \mid \Diamond i \phi \mid \mu X. \phi \mid \nu X. \phi \)

What does:

\[
\phi = \nu X. ( \text{if} \land \Diamond_1 ( \mu Y. ( \text{Nil} \lor \Box Y )) \land \Diamond_2 X )
\]

mean?

And how does it interact with a tree?

→ tree automata
Modal $\mu$-calculus

Grammar: $\phi, \psi ::= X \mid a \mid \phi \lor \psi \mid \phi \land \psi \mid \square \phi \mid \diamond_i \phi \mid \mu X. \phi \mid \nu X. \phi$

What does:

$$\phi = \nu X. (\text{if } \land \diamond_1 (\mu Y. (\text{Nil} \lor \square Y)) \land \diamond_2 X)$$

mean?

And how does it interact with a tree?

$\rightarrow$ tree automata
Alternating parity tree automata

Idea: the formula "starts" on the root

\[ \phi = \nu X. (\text{if} \land \Diamond_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \Diamond_2 X) \]
Alternating parity tree automata
Idea: the formula "starts" on the root

\[
\text{if } \text{if} \land \Diamond_1 \left( \mu Y. \left( \text{Nil} \lor \Box Y \right) \right) \land \Diamond_2 \phi
\]

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

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

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

where \( \phi = \nu X. \left( \text{if} \land \Diamond_1 \left( \mu Y. \left( \text{Nil} \lor \Box Y \right) \right) \right) \land \Diamond_2 X \)
Alternating parity tree automata

Idea: the formula "starts" on the root

\[
\text{if } \diamond_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \diamond_2 \phi
\]

where \( \phi = \nu X. (\text{if } \land \diamond_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \diamond_2 X) \)
Alternating parity tree automata

Idea: the formula "starts" on the root

where $\phi = \nu X. \left( \text{if} \land \lozenge_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \lozenge_2 X \right)$
Alternating parity tree automata

Idea: the formula "starts" on the root

\[
\text{if } \text{Nil } \mu Y. (\text{Nil } \lor \Box Y ) \text{ if } \phi
\]

where \( \phi = \nu X. (\text{if } \land \Diamond_1 (\mu Y. (\text{Nil } \lor \Box Y )) \land \Diamond_2 X ) \)
Alternating parity tree automata

Idea: the formula "starts" on the root

where $\phi = \nu X. (\text{if } \land \Diamond_1 (\mu Y. (\text{Nil } \lor \Box Y)) \land \Diamond_2 X)$
Alternating parity tree automata

Idea: the formula "starts" on the root

\[
\phi = \nu X. ( \text{if} \land \Diamond_1 ( \mu Y. ( \text{Nil} \lor \Box Y )) ) \land \Diamond_2 X
\]
Alternating parity tree automata

Idea: the formula "starts" on the root

where $\phi = \nu X. (\text{if } \land \Diamond_1 (\mu Y. (\text{Nil } \lor \Box Y)) \land \Diamond_2 X)$
Alternating parity tree automata

Idea: the formula "starts" on the root

\[
\begin{align*}
\text{if} & \quad \text{if} \\
\text{Nil} & \quad \text{Nil} \\
\text{data} & \quad \text{Nil} \lor □(\mu Y. (\text{Nil} \lor □ Y)) \\
\text{if} & \quad \phi \\
\text{Nil} & \quad \text{data} : \\
\text{data} & \quad \text{data} \\
& \quad \text{Nil}
\end{align*}
\]

where \( \phi = \nu X. (\text{if} \land □_1 (\mu Y. (\text{Nil} \lor □ Y)) \land □_2 X) \)
Alternating parity tree automata

Idea: the formula "starts" on the root

where $\phi = \nu X. (\text{if} \land \lozenge_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \Box_2 X)$
Alternating parity tree automata

Idea: the formula "starts" on the root

where $\phi = \nu X. (\text{if} \land \diamond_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \diamond_2 X)$
Alternating parity tree automata

Idea: the formula "starts" on the root

\[
\text{if} \\
\text{Nil Nil} \\
\text{if} \\
\text{data} \\
\text{Nil Nil} \\
\text{if} \\
\text{\( \phi \)} \\
\text{\( \text{Nil} \)} \\
\text{\( \text{Nil} \lor \Box(\mu Y. (\text{Nil} \lor \Box Y)) \)} \\
\text{data} \\
\text{\( \text{Nil} \)} \\
\text{\( \phi \)}
\]

where \( \phi = \nu X. (\text{if} \land \Diamond_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \Diamond_2 X) \)
Alternating parity tree automata

Idea: the formula "starts" on the root

where $\phi = \nu X. (\text{if} \land \diamond_1 (\mu Y. (\text{Nil} \lor \Box Y)) \land \diamond_2 X)$
Alternating parity tree automata

Conversion to an automaton?

- Unfold the formula over the tree, but *always* by reading a letter: synchronization with the tree.
- States $\Leftrightarrow$ subformulas
- Needs non-determinism for $\lor$ and alternation for $\land$
- Needs a parity condition for distinguishing $\mu$ and $\nu$
Alternating parity tree automata

For a MSO formula $\varphi$, 

$$\langle G \rangle \models \varphi$$

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

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

- weak MSO
- MSO
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.
Alternating parity tree automata

MSO allows to discriminate \textit{inductive} from \textit{coinductive} behaviour.

This allows to express properties as

- “a given operation is executed infinitely often in some execution”
- “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 occurring infinitely often along it is even.
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 \( \varphi \):

\[ \mathcal{A}_\varphi \text{ has a winning run-tree over } \langle G \rangle \text{ iff } \langle G \rangle \models \varphi \]
Intersection types and alternation
ATA vs. HORS

\[
\begin{align*}
(\lambda x.s) \ t & \xrightarrow{_{G_w}} s[x \leftarrow t] \\
\begin{array}{c}
s \xrightarrow{_{G_w}} s' \\
\end{array} & \begin{array}{c}
s \ t \xrightarrow{_{G_w}} s' \ t \\
\end{array}
\end{align*}
\]

\[
F \xrightarrow{_{G_w}} \mathcal{R}(F)
\]

\[
\begin{align*}
t & \xrightarrow{^*_w} a \ t_1 \cdots t_n \\
t_i : q_{ij} & \xrightarrow{^\infty} t'_i : q_{ij} \\
t : q & \xrightarrow{^\infty} (a \ (t'_{11} : (1, q_{11})) \cdots (t'_{nk_n} : (n, q_{nk_n}))) \ : q
\end{align*}
\]

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

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

refining the simple typing

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

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

In a derivation typing \( \text{if } T_1 \rightarrow T_2 : \)

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

\[
\text{App} \quad \emptyset \vdash \text{if } T_1 : (q_0 \land q_1) \rightarrow q_0
\]

\[
\text{App} \quad \Gamma_{21}, \Gamma_{22} \vdash \text{if } T_1 T_2 : q_0
\]

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

Theorem (Kobayashi)

\( S : q_0 \vdash S : q_0 \) iff the ATA \( \mathcal{A}_{\varphi} \) has a run-tree over \( \langle \mathcal{G} \rangle \).
A type-system for verification: without parity conditions

Axiom

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

\( \delta \)

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

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

\( \text{App} \)

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

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

\( \lambda \)

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

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

\( \text{fix} \)

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

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

**Theorem**
\[
S : q_0 \vdash S : q_0 \iff \text{the ATA } A_{\phi} \text{ has a run-tree over } \langle G \rangle.
\]

Proof: coinductive subject reduction/expansion along the head reduction of derivations with non-idempotent intersection types.

\[
\begin{array}{c}
\pi \\
\vdots \\
\pi' \\
\hline
S : q_0 \vdash S : q_0 \\
\emptyset \vdash \langle G \rangle : q_0
\end{array}
\iff
\begin{array}{c}
\langle G \rangle \text{ is accepted by } A.
\end{array}
\]
Adding parity conditions to the type system
where the $C_i$ are the tree contexts obtained by normalizing each $\pi_i$.

$C_0[C_1[], C_2[]]$ is a prefix of a run-tree of $\mathcal{A}$ over $\langle \mathcal{G} \rangle$. 
One more word on proof rewriting

Theorem

*In this quantitative setting, there is a correspondence between infinite branches of the typing of $\mathcal{G}$ and of the run-tree over $\langle \mathcal{G} \rangle$ obtained by normalization.*
One more word on proof rewriting

The goal now: **add information in** $\pi_i$ **about the maximal color seen in** $C_i$.

One extra color: $\epsilon$ for the case $C_i = []$. 
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

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

Application computes the “local” maximum of colors, and the fixpoint deals with the acceptance condition.
A type-system for verification (Grellois-Melliès 2014)

\[ \Delta \vdash t : (\Box c_1 \theta_1 \land \cdots \land \Box c_k \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa \]

\[ \Delta + \Box c_1 \Delta_1 + \ldots + \Box c_k \Delta_k \vdash t u : \theta :: \kappa' \]

Subject reduction: the contraction of a redex

\[ x : \Box \epsilon \theta_1 \vdash x : \theta_1 \quad x : \Box \epsilon \theta_2 \vdash x : \theta_2 \]

\[ \Delta, x : \Box c_1 \theta_1 \land \cdots \land \Box c_k \theta_k \vdash t : \theta \]

\[ \Delta \vdash \lambda x. t : (\Box c_1 \theta_1 \land \cdots \land \Box c_k \theta_k) \rightarrow \theta \]

\[ \Delta + \Box c_1 \Delta_1 + \ldots + \Box c_k \Delta_k \vdash (\lambda x. t) \ u : \theta \]
A type-system for verification (Grellois-Melliès 2014)

\[
\frac{\Delta \vdash t : (\square c_1 \theta_1 \land \cdots \land \square c_k \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa'}{\Delta + \square c_1 \Delta_1 + \ldots + \square c_k \Delta_k \vdash t u : \theta :: \kappa'}
\]

gives a proof of the same sequent:

\[
\Delta + \square c_1 \Delta_1 + \ldots + \square c_k \Delta_k \vdash t[u/x] : \theta
\]
A type-system for verification (Grellois-Melliès 2014)

Axiom
\[
\forall \{i\} \quad \square \in \theta_i :: \kappa \vdash x : \theta_i :: \kappa
\]

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

\[
\emptyset \vdash a : \bigwedge_{j=1}^{k_1} \square_{\Omega(q_{1j})} q_{1j} \to \cdots \to \bigwedge_{j=1}^{k_n} \square_{\Omega(q_{nj})} q_{nj} \to q :: o \to \cdots \to o \to o
\]

App
\[
\Delta \vdash t : (\bigwedge_{m_1} \theta_1 \land \cdots \land \bigwedge_{m_k} \theta_k) \to \theta :: \kappa \to \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa
\]

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

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

\[
\square \in \theta :: \kappa \vdash F : \theta :: \kappa
\]

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

\[
\Delta \vdash \lambda x . t : \bigwedge_{i \in I} \bigwedge_{m_i} \theta_i \to \theta :: \kappa \to \kappa'
\]
A type-system for verification (Grellois-Melliès 2014)

We rephrase the parity condition to typing trees, and now capture all MSO:

**Theorem (G.-Melliès 2014)**

\[ S : q_0 \vdash S : q_0 \text{ admits a winning typing derivation iff the alternating parity automaton } A \text{ has a winning run-tree over } \langle G \rangle. \]

We obtain **decidability** by collapsing to idempotent types.

Non-idempotency is very helpful for proofs, but leads to infinitary constructions.
Models of linear logic
It was linear logic all the way!

Linear logic very naturally handles alternation via

\[ A \Rightarrow B = !A \rightarrow B \]

In the relational semantics of linear logic, with \([o] = Q\),

\[ [!A] = M_{\text{fin}}([A]) \quad \text{and} \quad [A \rightarrow B] = [A] \times [B] \]

so that

\[ [o \rightarrow o \rightarrow o] = M_{\text{fin}}(Q) \times M_{\text{fin}}(Q) \times Q \]

and

\[ ([], [q_0, q_1], q_0) \in [\text{if}] \subseteq M_{\text{fin}}(Q) \times M_{\text{fin}}(Q) \times Q \]

models \( \delta(q_0, \text{if}) = (2, q_0) \land (2, q_1) \).
Duality between trees and automata

Church encoding of trees:

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

\[o(\Sigma) \rightarrow o = (o \rightarrow o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o\]

where “\(\lambda \Sigma\)” stands for \(\lambda \text{if.} \lambda \text{data.} \lambda \text{Nil.}\), and
Duality between trees and automata

Church encoding of trees:

\[ \lambda \Sigma \]
\[ \text{if} \]
\[ \text{Nil} \quad \text{if} \]
\[ \text{data} \quad \text{if} \]
\[ \text{Nil} \quad \text{data} : \]
\[ \text{data} \]
\[ \text{Nil} \]

Now, a term of type \( o(\Sigma) \rightarrow o \) normalizes to a \( \Sigma \)-labelled tree.
Model-checking I

An alternating tree automaton over $\Sigma$, with set of states $Q$, of transition function $\delta$, provides

$$[[\delta]] = [[\text{if}]] \cup [[\text{data}]] \cup [[\text{Nil}]] \subseteq [[o(\Sigma)]]$$

while a tree $t$ over $\Sigma$ gives, under Church encoding:

$$[[t]] \subseteq [[o(\Sigma) \rightarrow o]] = M_{\text{fin}}([[o(\Sigma)]]) \times Q$$

Relational composition:

$$[[t]] \circ M_{\text{fin}}([[[\delta]]]) \subseteq Q$$

Interactive interpretation?
Model-checking I

An alternating tree automaton over $\Sigma$, with set of states $Q$, of transition function $\delta$, provides

$$\[\delta\] = \[\text{if}\] \cup \[\text{data}\] \cup \[\text{Nil}\] \subseteq [o(\Sigma)]$$

while a tree $t$ over $\Sigma$ gives, under Church encoding:

$$[t] \subseteq [o(\Sigma) \rightarrow o] = M_{\text{fin}}([o(\Sigma)]) \times Q$$

Relational composition:

$$[t] \circ M_{\text{fin}}([\delta]) \subseteq Q$$

Interactive interpretation?
Relational composition:

\[ [t] \circ M_{\text{fin}}([\delta]) \subseteq Q \]

**Proposition**

\[ [t] \circ M_{\text{fin}}([\delta]) \]

is the set of states \( q \) from which

\[ \mathcal{A} = \langle \Sigma, Q, \delta \rangle \]

accepts the tree \( t \).
Model-checking I

Rel is a denotational model:

\[ t \rightarrow_{\beta} t' \quad \implies \quad \llbracket t \rrbracket = \llbracket t' \rrbracket \]

Corollary

For a term

\[ t : o(\Sigma) \rightarrow o \]

the set of states \( q \) from which

\[ \mathcal{A} = \langle \Sigma, Q, \delta \rangle \]

accepts the tree generated by the normalization of \( t \) is

\[ \llbracket t \rrbracket \circ \mathcal{M}_{\text{fin}}(\llbracket \delta \rrbracket) \]

Static analysis, directly on the term.
Model-checking II

Generalizing to trees generated by HORS? → add a fixpoint.

Finite iteration → inductive fixpoint operator on $Rel$.

**Theorem**

The infinitary normal form of a $\lambda Y$-term $t : o(\Sigma) \rightarrow o$ is accepted by $A = \langle \Sigma, Q, \delta \rangle$ from the set of states $[t] \circ M_{\text{fin}}([\delta])$.
Model-checking II

Generalizing to trees generated by HORS? $\rightarrow$ add a fixpoint.

Finite iteration $\rightarrow$ inductive fixpoint operator on $Rel$.

**Theorem**

*The infinitary normal form of a $\lambda Y$-term*

$t : o(\Sigma) \rightarrow o$

is accepted by

$A = \langle \Sigma, Q, \delta \rangle$

from the set of states

$[t] \circ M_{\text{fin}}([\delta])$

*after a finite execution of the automaton.*
On finiteness

Infinite trees need infinite multisets: tree constructors may be used countably.

Defining a new exponential

\[ \mathcal{L} : A \mapsto M_{\text{count}}(A) \]

gives a relational model of linear logic with a coinductive fixpoint operator (infinite fixpoint unfolding).

New interpretation of terms: \[ [ t ]_{gfp} \].
Theorem

The infinitary normal form of a \( \lambda Y \)-term

\[
t : o(\Sigma) \to o
\]

is accepted by

\[
A = \langle \Sigma, Q, \delta \rangle
\]

from the set of states

\[
\llbracket t \rrbracket_{gfp} \circ M_{count}(\llbracket \delta \rrbracket)
\]
The coloring comonad

The coloring modality of the type system corresponds to a comonad in the semantics:

\[ \Box A = Col \times A \]

Structural morphisms:

\[
\begin{align*}
(m_1, m_2, a) \leadsto (m_1, (m_2, a)) &: \Box A \to \Box \Box A \\
(\epsilon, a) \leadsto a &: \Box A \to A
\end{align*}
\]
Parity conditions

The modality □ distributes over the exponential §: there is a natural transformation

§ □ → □ §

satisfying some coherence diagram.

It follows that the composite

§ = § □

is an exponential, so that we automatically obtain a model of the λ-calculus associated to the coloured typings.

Colored interpretation: [[t]]_{col}.
Linear decomposition of the intuitionistic arrow

Kleisli composition: consider

\[ f : \not{} \lozenge A \rightarrow B \]

and

\[ g : \not{} \lozenge B \rightarrow C \]

Their composite is defined as

\[ \not{} \lozenge B \xrightarrow{g} C \]

where \( \lambda \) is the distributivity law between \( ! \) and \( \square \).
Linear decomposition of the intuitionistic arrow

Kleisli composition: consider

\[ f : \downarrow \Box A \to B \]

and

\[ g : \downarrow \Box B \to C \]

Their composite is defined as

\[ \downarrow \Box \downarrow \Box A \xrightarrow{\downarrow \Box f} \downarrow \Box B \xrightarrow{g} C \]

where \( \lambda \) is the distributivity law between \( \downarrow \) and \( \Box \).
Linear decomposition of the intuitionistic arrow

Kleisli composition: consider

\[ f : \not \sqcap A \rightarrow B \]

and

\[ g : \not \sqcap B \rightarrow C \]

Their composite is defined as

\[ \not \not \not \sqcap A \xrightarrow{\lambda} \not \sqcap \not \not \sqcap A \xrightarrow{\not \sqcap f} \not \sqcap B \xrightarrow{g} C \]

where \( \lambda \) is the distributivity law between \( ! \) and \( \square \).
Linear decomposition of the intuitionistic arrow

Kleisli composition: consider

\[ f : \diamond \Box A \rightarrow B \]

and

\[ g : \diamond \Box B \rightarrow C \]

Their composite is defined as

\[
\begin{align*}
\diamond \Box \Box A & \rightarrow \diamond \Box \Box A \\
& \lambda \\
& \diamond \Box f \rightarrow \Box B g \\
& \rightarrow C
\end{align*}
\]

where \( \lambda \) is the distributivity law between \( \Box \) and \( \Box \).
Kleisli composition: consider

\[ f : \not\not\Box A \to B \]

and

\[ g : \not\not\Box B \to C \]

Their composite is defined as

\[
\not\not\Box A \to \not\not\Box\Box A \to \not\not\not\Box\Box A \xrightarrow{\lambda} \not\not\Box\not\not\Box\Box A \xrightarrow{\not\not\Box f} \not\not\Box B \xrightarrow{g} C
\]

where \( \lambda \) is the distributivity law between \( ! \) and \( \Box \).
Parity conditions

We obtain a very natural colored interpretation of types:

$$
\llbracket A \Rightarrow B \rrbracket = M_{\text{count}}(\text{Col} \times \llbracket A \rrbracket) \times \llbracket B \rrbracket
$$

and we can relate the typing derivations in the colored intersection type system with the construction of denotations in the resulting model.
An example of coloured interpretation

Suppose $\Omega(q_0) = 0$ and $\Omega(q_1) = 1$.

This rule will be interpreted in the model as

$$([(0, q_0), (1, q_1), (1, q_1)], [(1, q_1)], q_0)$$
Connection with the coloured relational model

To obtain the acceptance theorem for alternating parity automata, we need a fixpoint which reflects the parity condition.

This operator composes denotations infinitely, and only keeps the result if it comes from a winning composition tree.
Theorem

The infinitary normal form of a $\lambda Y$-term

$$t : o(\Sigma) \to o$$

is accepted by the parity automaton

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

from the set of states

$$[t]_{col} \circ M_{col}([\delta])$$
Ehrhard 2012: the \textit{finitary} modal $ScottL$ is the extensional collapse of $Rel$.

Two essential differences:

- $\lfloor ! A \rfloor = \mathcal{P}_{\text{fin}}(A)$
- necessity of “subtyping”

We adapted to $ScottL$ the theoretical approach of this work.

\textbf{Corollary}

\textit{The higher-order model-checking problem is decidable.}
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 idempotent type systems / finitary semantics, we obtain **decidability** of higher-order model-checking.

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 idempotent type systems / finitary semantics, we obtain decidability of higher-order model-checking.

Thank you for your attention!