Coinductive semantics of linear logic
and higher-order model-checking

Charles Grellois — joint work with Paul-André Melliès

PPS & LIAFA — Université Paris 7
University of Dundee

Scottish Theorem Proving
Dundee University — Oct 7, 2015
Model-checking higher-order programs

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

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

$$M \models \varphi$$

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
```
Model-checking higher-order programs

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

Example:

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

modelled as

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

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

For higher-order programs with recursion, $\mathcal{M}$ 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)
Model-checking higher-order programs

For higher-order programs with recursion, $\mathcal{M}$ is a higher-order tree over which we run

\[ \text{an alternating parity tree automaton (APT) } \mathcal{A}_\varphi \]

corresponding to a

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

(safety, liveness properties, etc)

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

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

is abstracted as

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

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

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

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

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

\[ S \rightarrow_G L \text{ Nil} \]
Higher-order recursion schemes

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

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

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

\[ \langle G \rangle = \]

[Diagram of a tree structure representing the recursion scheme]
Higher-order recursion schemes

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

Finite representation of “higher-order regular” infinite trees: rewriting produces a tree \(\langle \mathcal{G} \rangle\).

“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

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

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

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

\[
\begin{align*}
(\lambda x. s) t & \rightarrow_{G_w} s[x \leftarrow t] \\
F & \rightarrow_{G_w} R(F)
\end{align*}
\]

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

![Diagram of alternating tree automata]

Charles Grellois (PPS - LIAFA - Dundee)
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

\[
\begin{align*}
(\lambda x. s)\ t & \rightarrow_{Gw} s[x \leftarrow t] \\
\frac{s \rightarrow_{Gw} s'}{s\ t \rightarrow_{Gw} s'\ t}
\end{align*}
\]

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

\[
\begin{align*}
t & \rightarrow^*_{Gw} a\ t_1 \cdots\ t_n \\
t_i & : q_{ij} \rightarrow^\infty_{G,A} t'_i : q_{ij} \\
t & : q \rightarrow^\infty_{G,A} (a\ (t_{11}' : (1, q_{11})) \cdots\ (t_{nk_n}' : (n, q_{nk_n}))) : q
\end{align*}
\]

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 \( \text{if } T_1 \ T_2 : \)

\[
\begin{align*}
\delta & \quad \emptyset \vdash \text{if} : \emptyset \rightarrow (q_0 \wedge q_1) \rightarrow q_0 & \emptyset \\
\text{App} & \quad \emptyset \vdash \text{if} T_1 : (q_0 \wedge q_1) \rightarrow q_0 & \emptyset \\
\text{App} & \quad \emptyset \vdash \text{if} T_1 T_2 : q_0 & \emptyset \\
\end{align*}
\]

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

\[ \text{Axiom} \quad \frac{x : \bigwedge_{i \in I} \theta_i :: \kappa \vdash x : \theta_i :: \kappa}{\} \]

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

App

\[ \frac{\Delta \vdash t : (\theta_1 \land \cdots \land \theta_k) \to \theta :: \kappa \to \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa}{\Delta + \Delta_1 + \cdots + \Delta_k \vdash tu : \theta :: \kappa'} \]

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

fix

\[ \frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \theta :: \kappa \vdash F : \theta :: \kappa} \]
An alternate proof

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

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

rewrites to

\[
\begin{align*}
\pi[x \leftarrow (\pi_i)_i] \\
\vdots \\
\Gamma + \sum_i \Gamma_i \vdash s[x \leftarrow t] : \sigma \\
\end{align*}
\]

Lifting of the alternating behavior to higher-order.
An alternate proof

The head reduction of derivations computes prefixes of the run-tree:

\[
\begin{align*}
\pi_1 \\
\vdots \\
S : q_0 & \vdash S : q_0
\end{align*}
\]
An alternate proof

The head reduction of derivations computes prefixes of the run-tree:

\[
\begin{align*}
\pi_1 & \\
\vdots & \\
\text{fix} & \quad S : q_0 \vdash S : q_0
\end{align*}
\]

\[
\begin{align*}
\pi_2 & \\
\vdots & \\
\text{fix} & \quad L : (q_0 \land q_0 \land q_1) \rightarrow q_0 \vdash L : (q_0 \land q_0 \land q_1) \rightarrow q_0 \\
& \quad \vdash \text{Nil} : q_0 \quad \vdash \text{Nil} : q_0 \quad \vdash \text{Nil} : q_1
\end{align*}
\]

\[
L : (q_0 \land q_0 \land q_1) \rightarrow q_0 \quad \text{informs that its argument will be used three times in the proof-tree: twice with state } q_0, \text{ once with state } q_1.
\]

Recall that \(x\) only occurs twice, but alternation makes additional duplications.
An alternate proof

The **head reduction** of derivations computes prefixes of the run-tree:

\[
\begin{align*}
\pi_1 \\
\text{fix} \\
\vdash S : q_0 \rightarrow S : q_0 \\
\vdash L : (q_0 \land q_0 \land q_1) \rightarrow q_0 \\
\vdash L : (q_0 \land q_0 \land q_1) \rightarrow q_0 \\
\vdash L \text{ Nil} : q_0 \\
\vdash \text{if} : (q_0 \land q_1) \rightarrow q_1 \rightarrow q_0 \\
\vdash \text{if} \text{ Nil} : q_1 \\
\vdash \text{if Nil} : q_0 \\
\vdash L : q_0 \rightarrow q_1 \\
\vdash \text{if} \text{Nil} L \ (\text{data Nil}) : q_0
\end{align*}
\]

This time \( L : q_0 \rightarrow q_1 \) implies that one of the occurrences of \( x \) will not be “visited” by the automaton.
An alternate proof

The head reduction of derivations computes prefixes of the run-tree:

\[ \pi_1 \]

\[ \text{fix} \]

\[ \vdash S : q_0 \Rightarrow S : q_0 \]

\[ \pi_2 \]

\[ \vdash L : (q_0 \land q_0 \land q_1) \Rightarrow q_0 \Rightarrow L : (q_0 \land q_0 \land q_1) \Rightarrow q_0 \]

\[ \text{fix} \]

\[ \vdash \text{Nil} : q_0 \Rightarrow \text{Nil} : q_0 \Rightarrow \text{Nil} : q_1 \]

\[ L : (q_0 \land q_0 \land q_1) \Rightarrow q_0 \Rightarrow L \text{ Nil} : q_0 \]

\[ \pi_3 \]

\[ \vdash \text{if} : (q_0 \land q_1) \Rightarrow q_1 \Rightarrow q_0 \Rightarrow \text{Nil} : q_0 \Rightarrow \text{Nil} : q_1 \]

\[ \vdash \text{if Nil} : q_1 \Rightarrow q_0 \]

\[ \vdash \text{if Nil L} (\text{data Nil}) : q_0 \]

\[ \Rightarrow \infty \]

run-tree of \( \mathcal{A} \) over \( \langle G \rangle \).
An alternate proof

Theorem

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

Proof: coinductive subject reduction/expansion + head reduction of derivations.

\[
\begin{array}{c}
\pi \\
\vdots \\
\hline
\vdash S : q_0 \\
\hline
\end{array} \iff \begin{array}{c}
\pi' \\
\vdots \\
\hline
\vdash \emptyset \vdash \langle G \rangle : q_0 \\
\hline
\end{array} \iff \langle G \rangle \text{ is accepted by } A.
\]
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 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 \( \varphi \):

\[ \mathcal{A}_\varphi \text{ has a winning run-tree over } \langle G \rangle \text{ iff } \langle G \rangle \models \varphi \]
One more word on proof rewriting

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 $A$ over $\langle G \rangle$. 

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

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

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

**Axiom**
\[ x : \bigwedge_{\{i\}} \Box \epsilon \theta_i :: \kappa \vdash x : \theta_i :: \kappa \]

\[ \delta \]
\[ \{ (i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i \} \text{ satisfies } \delta_A(q, a) \]
\[ \emptyset \vdash a : \bigwedge_{j=1}^{k_1} \Box \Omega(q_{1j}) q_{1j} \rightarrow \cdots \rightarrow \bigwedge_{j=1}^{k_n} \Box \Omega(q_{nj}) q_{nj} \rightarrow q :: o \rightarrow \cdots \rightarrow o \rightarrow o \]

**App**
\[ \Delta \vdash t : (\bigwedge_{m_1} \theta_1 \land \cdots \land \bigwedge_{m_k} \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa \]
\[ \Delta + \bigwedge_{m_1} \Delta_1 + \cdots + \bigwedge_{m_k} \Delta_k \vdash tu : \theta :: \kappa' \]

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

\[ \lambda \]
\[ \Delta, x : \bigwedge_{i \in I} \bigwedge_{m_i} \theta_i :: \kappa \vdash t : \theta :: \kappa' \]
\[ \Delta \vdash \lambda x. t : (\bigwedge_{i \in I} \bigwedge_{m_i} \theta_i) \rightarrow \theta :: \kappa \rightarrow \kappa' \]
We 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.
A word on linear logic

Linear logic very naturally handles alternation via

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

and we can extend it with a coloring modality \( \Box \).

We design two kind of semantics, whose denotations are the refined types terms admit:

- an infinitary semantics, corresponding to non-idempotent colored types,
- and a finitary one, which is decidable.

Both models are natural extensions of well-known models of linear logic, with coloring and fixpoint.

For more: come at SPLS in two weeks!
A word on linear logic

Linear logic very naturally handles alternation via

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

and we can extend it with a coloring modality \( \square \).

We design two kind of semantics, whose denotations are the refined types terms admit:

- an infinitary semantics, corresponding to non-idempotent colored types,
- and a finitary one, which is decidable.

Both models are natural extensions of well-known models of linear logic, with coloring and fixpoint.

For more: come at SPLS in two weeks!
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!