Semantics of linear logic 
and higher-order model-checking

Charles Grellois     Paul-André Melliès

IRIF — Université Paris 7
FOCUS Team – INRIA & University of Bologna

GDRI-LL Meeting – University of Bologna
February 2, 2016
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
- Interaction: the result is whether

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

Typically: translate $\varphi$ to an equivalent automaton running over $\mathcal{M}$:

$$\varphi \mapsto \mathcal{A}_\varphi$$
Model-checking higher-order programs

For higher-order programs with recursion, $\mathcal{M}$ is a higher-order regular 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
    Nil  if
      data  if
        Nil  data
          data
            Nil
```

Model-checking higher-order programs

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

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

For higher-order programs with recursion, $\mathcal{M}$ is a higher-order regular 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 regular tree satisfies a MSO formula?
Higher-order recursion schemes

Some regularity for infinite trees
Higher-order recursion schemes

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

is abstracted as

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

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

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

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

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

\[
\begin{array}{c}
S \\
\rightarrow_G \\
\end{array}
\]
Higher-order recursion schemes

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

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

\[ \langle G \rangle = \begin{cases} 
  \text{if} \\
  \text{Nil} & \text{if} \\
  \text{data} & \text{if} \\
  \text{Nil} & \text{data} : \\
  \text{data} & \\
  \text{Nil} 
\end{cases} \]
Higher-order recursion schemes

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

“Everything” is simply-typed, and

\textit{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).
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).

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

simply-typed recursion operators \( Y_\sigma : (\sigma \rightarrow \sigma) \rightarrow \sigma \).
Alternating parity 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: $\delta(q_0, \text{if}) = (2, q_0) \land (2, q_1)$. 

\[
\begin{aligned}
\text{if } q_0 \\
\text{Nil} \\
\text{data} \\
\text{Nil} \\
A \varphi
\end{aligned}
\quad
\begin{aligned}
\text{if } q_0 \\
\text{if } q_0 \\
\text{if } q_1 \\
\text{data} \\
\text{Nil} \\
A \varphi
\end{aligned}
\]

\[
\begin{aligned}
\text{if } q_0 \\
\text{data} \\
\text{:} \\
\text{data} \\
\text{Nil}
\end{aligned}
\quad
\begin{aligned}
\text{data} \\
\text{:} \\
\text{Nil} \\
\text{data} \\
\text{Nil}
\end{aligned}
\]

\[
\begin{aligned}
\text{data} \\
\text{Nil}
\end{aligned}
\quad
\begin{aligned}
\text{data} \\
\text{Nil}
\end{aligned}
\]
### Alternating parity tree automata

MSO discriminates **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 $\varphi$:

$$\mathcal{A}_{\varphi} \text{ has a winning run-tree over } \langle G \rangle \text{ iff } \langle G \rangle \models \varphi.$$
Recognition by homomorphism
Automata and recognition

For the usual finite automata on words: given a regular 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 \( \phi : A^* \to M \) such that \( L = \phi^{-1}(K) \).

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

Let’s extend this to:

- higher-order recursion schemes
- alternating parity automata


How to model…

- Alternation?
- Recursion?
- Parity condition?
Intersection types and alternation
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 \( \text{if } T_1 \ T_2 : \)

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

\( \Gamma_{21}, \Gamma_{22} \vdash \text{if } T_1 \ T_2 : q_0 \)

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

**Theorem (Kobayashi)**

\( S : q_0 \vdash S : q_0 \text{ iff the ATA } \mathcal{A}_{\varphi} \text{ has a run-tree over } \langle \mathcal{G} \rangle. \)
A closer look at the Application rule

App

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

\[
\Delta, \Delta_1, \ldots, \Delta_k \vdash t u : \theta \rightsquigarrow \kappa'
\]

Towards sequent calculus:

\[
\Delta \vdash t : (\bigwedge_{i=1}^{n} \theta_i) \rightarrow \theta' \quad \Delta_i \vdash u : \theta_i \quad \forall i \in \{1, \ldots, n\}
\]

\[
\Delta_1, \ldots, \Delta_n \vdash u : \bigwedge_{i=1}^{n} \theta_i
\]

\[
\Delta, \Delta_1, \ldots, \Delta_n \vdash t u : \theta'
\]

Right \lang
A closer look at the Application rule

\[ \Delta \vdash t : \left( \bigwedge_{i=1}^{n} \theta_i \right) \rightarrow \theta' \] \[ \Delta_i \vdash u : \theta_i \quad \forall i \in \{1, \ldots, n\} \] \[ \Delta_1, \ldots, \Delta_n \vdash u : \bigwedge_{i=1}^{n} \theta_i \]

Right \ \bigwedge

\[ \Delta, \Delta_1, \ldots, \Delta_n \vdash t \ u : \theta' \]

Linear decomposition of the intuitionistic arrow:

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

Two steps: duplication / erasure, then linear use.

Right \ \bigwedge \ corresponds to the Promotion rule of indexed linear logic.
Intersection types and semantics of linear logic

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

Two interpretations of the exponential modality:

Qualitative models (Scott semantics)

\[ ! A = \mathcal{P}_{\text{fin}}(A) \]

\[ \llbracket o \Rightarrow o \rrbracket = \mathcal{P}_{\text{fin}}(Q) \times Q \]

\[ \{q_0, q_0, q_1\} = \{q_0, q_1\} \]

Order closure

Quantitative models (Relational semantics)

\[ ! A = \mathcal{M}_{\text{fin}}(A) \]

\[ \llbracket o \Rightarrow o \rrbracket = \mathcal{M}_{\text{fin}}(Q) \times Q \]

\[ [q_0, q_0, q_1] \neq [q_0, q_1] \]

Unbounded multiplicities
Intersection types and semantics of linear logic

Fundamental idea:

\[ [t] \cong \{ \theta \mid \emptyset \vdash t : \theta \} \]

and similarly for open terms.
Let $t$ be a term normalizing to a tree $\langle t \rangle$ and $\mathcal{A}$ be an alternating automaton.

$$\mathcal{A} \text{ accepts } \langle t \rangle \text{ from } q \iff q \in \llbracket t \rrbracket \iff \emptyset \vdash t : q :: o$$

Extension with recursion and parity condition?
Adding parity conditions to the type system
Alternating parity tree automata

We add coloring annotations to intersection types:

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

now corresponds to

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

Idea: if is a run-tree with two holes:

A new neutral color: $\epsilon$ for an empty run-tree context $[]_q$. 
A type-system for verification

A colored Application rule:

\[
\Delta \vdash t : (\Box_{c_1} \theta_1 \land \cdots \land \Box_{c_k} \theta_k) \to \theta :: \kappa \to \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa
\]

\[
\Delta + \Box_{c_1} \Delta_1 + \ldots + \Box_{c_k} \Delta_k \vdash t u : \theta :: \kappa'
\]
A type-system for verification

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 considering **idempotent** types.

Non-idempotency is very helpful for proofs, but leads to infinitary constructions.
Colored models of linear logic
A closer look at the Application rule

\[
\begin{align*}
\Delta & \vdash t : (\Box m_1 \theta_1 \land \cdots \land \Box m_k \theta_k) \to \theta : \kappa \to \kappa' \quad \Delta_i \vdash u : \theta_i : \kappa \\
\Delta + \Box m_1 \Delta_1 + \cdots + \Box m_k \Delta_k & \vdash t \circ u : \theta : \kappa'
\end{align*}
\]

Towards sequent calculus:

\[
\begin{align*}
\Delta_1 & \vdash u : \theta_1 \\
\Box m_1 \Delta_1 & \vdash u : \Box m_1 \theta_1 \\
\cdots & \\
\Box m_n \Delta_n & \vdash u : \Box m_n \theta_1 \\
\Delta, \Box m_1 \Delta_1, \cdots, \Box m_n \Delta_n & \vdash t \circ (\land_{i=1}^n \Box m_i \theta_i) \to \theta \\
\Delta, \Box m_1 \Delta_1, \cdots, \Box m_n \Delta_n & \vdash t \circ u : \theta
\end{align*}
\]

Right \(\Box\) looks like a promotion. In linear logic:

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

Our reformulation of the Kobayashi-Ong type system shows that \(\Box\) is a modality (in the sense of S4) which distributes with the exponential in the semantics.
Colored semantics

We extend:

- \( Rel \) with \textit{countable} multiplicities, \textit{coloring} and an \textit{inductive-coinductive} fixpoint
- \( ScottL \) with \textit{coloring} and an \textit{inductive-coinductive} fixpoint.

Methodology: think in the relational semantics, and adapt to the Scott semantics using Ehrhard’s 2012 result:

the \textit{finitary} model \( ScottL \) is the extensional collapse of \( Rel \).
Model-checking and finitary semantics

Let $G$ be a HORS representing the tree $\langle G \rangle$ and $A$ be an alternating parity automaton.

Conjecture in infinitary $Rel$, but theorem in colored $ScottL$:

$$A \text{ accepts } \langle G \rangle \text{ from } q \iff q \in \llbracket t \rrbracket$$

A similar theorem holds for a companion intersection type system to colored $ScottL$. Since the semantics are finitary:

**Corollary**

*The higher-order model-checking problem is decidable.*

Thank you for your attention!
Model-checking and finitary semantics

Let $G$ be a HORS representing the tree $\langle G \rangle$ and $A$ be an alternating parity automaton.

Conjecture in infinitary $Rel$, but theorem in colored $ScottL$:

$$A \text{ accepts } \langle G \rangle \text{ from } q \iff q \in [t]$$

A similar theorem holds for a companion intersection type system to colored ScottL. Since the semantics are finitary:

**Corollary**

The higher-order model-checking problem is decidable.

Thank you for your attention!