Semantics of linear logic
and higher-order model-checking

Charles Grellois  Paul-André Melliès

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

Linear Logic 2016
November 9, 2016
Model-checking higher-order programs

For higher-order programs with recursion, the model $\mathcal{M}$ of interest 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

```
  if
   /
  Nil    if
   /
  data   if
   /
  Nil    data :
   /
  data
   /
  Nil
```

Charles Grellois (IRIF - Bologna)  Semantics of linear logic and model-checking  Nov 9, 2016  2 / 34
Model-checking higher-order programs

For higher-order programs with recursion, the model $\mathcal{M}$ of interest is a higher-order regular tree.

Example:

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

modelled as

Finite representation: HORS
Model-checking higher-order programs

For higher-order programs with recursion, the model \( M \) of interest 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

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

is abstracted as

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

which produces the higher-order tree of actions

```
if
  if
    data : 
      if
        Nil
      else
        Nil
  else
    Listen (data x)
```

Charles Grellois (IRIF - Bologna)  Semantics of linear logic and model-checking  Nov 9, 2016  4 / 34
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 \(S\):

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

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

\[ \begin{array}{llllllll}
  | & \text{if} & \text{Nil} & L & \\
  | & \text{data} & \text{Nil} & \end{array} \]
Higher-order recursion schemes

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

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

\[
\langle G \rangle = \begin{cases} 
  \text{if} \\
  \text{Nil} & \text{if} \\
  \text{data} & \text{if} \\
  \text{Nil} & \text{data} \\
  \text{Nil} 
\end{cases}
\]
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 \( Y_{\sigma} : (\sigma \to \sigma) \to \sigma \).
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 $\langle G \rangle$.

$$\text{APT } = \text{ alternating tree automata (ATA) } + \text{ parity condition.}$$
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: $\delta(q_0, \text{if}) = (2, q_0) \land (2, q_1)$. 

\[
\begin{array}{c}
\text{if } q_0 \\
\text{Nil } \\
\text{data } \\
\text{Nil } \\
\text{Nil}
\end{array} \quad \text{if } q_0 \\
\text{Nil } \\
\text{data } \\
\text{Nil } \\
\text{Nil}
\]

\[
\begin{array}{c}
\text{if } q_0 \\
\text{Nil } \\
\text{data } \\
\text{Nil } \\
\text{Nil}
\end{array} \quad \rightarrow A_{\phi} \quad \begin{array}{c}
\text{if } q_0 \\
\text{Nil } \\
\text{data : } \\
\text{data } \\
\text{Nil}
\end{array} \quad \text{if } q_0 \\
\text{Nil } \\
\text{data : } \\
\text{data } \\
\text{Nil}
\]

\[
\begin{array}{c}
\text{if } q_0 \\
\text{Nil } \\
\text{data : } \\
\text{data } \\
\text{Nil}
\end{array} \quad \text{if } q_1 \\
\text{Nil } \\
\text{data : } \\
\text{data } \\
\text{Nil}
\]

Charles Grellois (IRIF - Bologna)  Semantics of linear logic and model-checking  Nov 9, 2016  8 / 34
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”.

Charles Grellois (IRIF - Bologna)
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 $\varphi$:

$$A_\varphi \text{ has a winning run-tree over } \langle G \rangle \text{ iff } \langle G \rangle \models \varphi.$$
Intersection types and alternation
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 \( \text{if } T_1 \ T_2 : \)

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

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 \( A_{\varphi} \) has a run-tree over \( \langle \mathcal{G} \rangle \).
A type-system for verification: without parity conditions

Axiom

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

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

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

App

\[ \Delta \vdash t : (\bigwedge_{i=1}^{k} \theta_i) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa \]

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

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

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

fix

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

\[ F : \theta :: \kappa \vdash F : \theta :: \kappa \]
A closer look at the Application rule

\[
\text{App} \quad \frac{\Delta \vdash t : (\theta_1 \land \cdots \land \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa}{\Delta, \Delta_1, \ldots, \Delta_k \vdash t \ u : \theta :: \kappa'}
\]

can be decomposed as:

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

Right $\land$
A closer look at the Application rule

\[
\Delta \vdash t : (\bigwedge_{i=1}^{n} \theta_i) \to \theta' \\
\Delta, \Delta_1, \ldots, \Delta_n \vdash u : \bigwedge_{i=1}^{n} \theta_i \\
\Delta \vdash t u : \theta'
\]

Linear decomposition of the intuitionistic arrow:

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

Two steps: duplication / erasure, then linear use.

Right \( \wedge \) 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)

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

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

\[ \llbracket q_0, q_0, q_1 \rrbracket \neq \llbracket q_0, q_1 \rrbracket \]

**Unbounded multiplicities**
An example of interpretation

In \( \text{Rel} \), one denotation:

\[
([q_0, q_1, q_1], [q_1], q_0)
\]

In \( \text{ScottL} \), a set containing the principal type

\[
([q_0, q_1], [q_1], q_0)
\]

but also

\[
([q_0, q_1, q_2], [q_1], q_0)
\]

and

\[
([q_0, q_1], [q_0, q_1], q_0)
\]

and ...
Intersection types and semantics of linear logic

Fundamental idea:

\[
\llbracket t \rrbracket \equiv \{ \theta \mid \emptyset \vdash t : \theta \}
\]

for a closed term.
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

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

Idea: $\text{if}$ is a run-tree with two holes:

$$\text{if} \quad \begin{array}{c} \text{if} \\ []q_0 \\ []q_1 \end{array}$$

A new neutral color: $\epsilon$ for an empty run-tree context $[]q$. 
An example of colored intersection type

Set $\Omega(q_i) = i.$

\[
\lambda x \\
\lambda y \\
\lambda x
\]

has type

\[
\Box_0 q_0 \land \Box_1 q_1 \to \Box_1 q_1 \to q_1
\]

Note the color 0 on $q_0$...
A type-system for verification (Grellois-Melliès 2014)

Axiom

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

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

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

App

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

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

fix

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

\[ F : \Box \epsilon \theta :: \kappa \vdash F : \theta :: \kappa \]

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

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

A colored Application rule:

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

inducing a winning condition on infinite proofs: the node

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

has color \( c_i \), others have color \( \epsilon \), and we use the parity condition.
A type-system for verification

We now capture all MSO:

**Theorem (G.-Melliès 2014, from Kobayashi-Ong 2009)**

$S : q_0 \vdash S : q_0$ admits a winning typing derivation iff the alternating parity automaton $A$ 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

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

\[ \Delta + \Box_{m_1} \Delta_1 + \ldots + \Box_{m_k} \Delta_k \vdash t u : \theta :: \kappa' \]

can be decomposed as:

\[ \Delta_1 \vdash u : \theta_1 \quad \Box_{m_1} \Delta_1 \vdash u : \Box_{m_1} \theta_1 \quad \ldots \quad \Box_{m_n} \Delta_n \vdash u : \Box_{m_n} \theta_1 \]

\[ \Delta, \Box_{m_1} \Delta_1, \ldots, \Box_{m_n} \Delta_n \vdash t u : \theta \]

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:

- \( \text{Rel} \) with \textit{countable} multiplicities, \textit{coloring} and an \textit{inductive-coinductive} fixpoint
- \textit{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 \textit{ScottL} is the extensional collapse of \textit{Rel}. 
Infinitary relational semantics

Extension of $\text{Rel}$ with infinite multiplicities:

$$\downarrow\ A = \mathcal{M}_{\text{count}}(A)$$

and coloring modality

$$\Box\ A = \text{Col} \times A$$

Distributive law:

$$\lambda_{\downarrow} : \downarrow\Box\ A \rightarrow \Box\downarrow\ A$$

$$\{(((c, a_1), (c, a_2), \ldots), (c, [a_1, a_2, \ldots]))) | a_i \in A, c \in \text{Col}\}$$

Allows to compose comonads: $\downarrow = \downarrow\Box$ is an exponential in the infinitary relational semantics.

This induces a colored CCC $\text{Rel}_{\downarrow}$ ($\rightarrow$ model of the $\lambda$-calculus).
An example of interpretation

Set $\Omega(q_i) = i$. 

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

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

has denotation

\[
([[(0, q_0), (1, q_1), (1, q_1)], [(1, q_1)], q_1])
\]

(corresponding to the type $\Box_0 q_0 \land \Box_1 q_1 \rightarrow \Box_1 q_1 \rightarrow q_1$)
An inductive-coinductive fixpoint operator

$Y$ transports

$$f : \downarrow X \otimes \downarrow A \rightarrow A$$

into

$$Y_{X,A}(f) : \downarrow X \rightarrow A.$$ 

by composing together denotations of $f$ in a way which satisfies the parity condition.

$Y$ is a Conway operator, and $Rel_\downarrow$ is a model of the $\lambda Y$-calculus.
Conjecture

An APT $A$ has a winning run from $q_0$ over $\langle G \rangle$ if and only if

$$q_0 \in \llbracket \lambda(G) \rrbracket$$

where $\lambda(G)$ is a $\lambda Y$-term corresponding to $G$.

Using Church encoding, we can also design an interpretation independent of the automaton of interest.
Finitary semantics

In ScottL, we define □, λ and Y similarly (using downward-closures).

ScottL⊥ is a model of the λY-calculus.

Theorem
An APT A has a winning run from q₀ over ⟨G⟩ if and only if

\[ q₀ ∈ [\lambda(G)] \]

Corollary
The higher-order model-checking problem is decidable.
Conclusion

- Connections between intersection types and linear logic
- Refinement of the Kobayashi-Ong type system: coloring is a modality
- Colored models of the $\lambda Y$-calculus coming from linear logic
- Decidability using the finitary Scott semantics
- Raises interesting questions in semantics: infinitary models, coeffects...
- Ongoing work: a probabilistic extension

Thank you for your attention!
Conclusion

- Connections between **intersection types** and **linear logic**
- Refinement of the Kobayashi-Ong type system: **coloring is a modality**
- **Colored models** of the $\lambda Y$-calculus coming from **linear logic**
- **Decidability** using the finitary Scott semantics
- Raises interesting questions in semantics: **infinitary models, coeffects** . . .
- Ongoing work: a **probabilistic extension**

Thank you for your attention!