Semantics of linear logic and higher-order model-checking

Charles Grellois    Paul-André Melliès

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

NII Shonan Meeting on HOMC
March 14, 2016
Model-checking higher-order programs

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

Example:

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

modelled as

\[
\text{if} \ \\
\text{Nil} \ \\
\text{if} \ \\
\text{data} \ \\
\text{Nil} \ \\
\text{if} \ \\
\text{data} \ \\
\text{Nil}
\]
Model-checking higher-order programs

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

Example:

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

modelled as

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

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

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

is abstracted as

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

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

\[
\begin{tikzpicture}
\node (root) {if}
child {node (nil) {Nil}}
child {node (if) {if}
child {node (data) {data : \text{\ldots}}}
child {node (nil) {Nil}}}
\end{tikzpicture}
\]
Higher-order recursion schemes

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

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

\[
S \rightarrow G
\]

\[
\text{L Nil}
\]
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

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

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

\[ \langle G \rangle = \]
Higher-order recursion schemes

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

“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

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

“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 $\langle G \rangle$.

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

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

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

\[ A_\varphi \text{ has a winning run-tree over } \langle G \rangle \text{ iff } \langle G \rangle \models \varphi. \]
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
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 $\text{if } \begin{array}{c} T_1 \quad T_2 : \end{array}$:

\[
\begin{array}{cccc}
\delta & \emptyset \vdash \text{if} : \emptyset \to (q_0 \land q_1) \to q_0 & \emptyset & \vdash : \\
\text{App} & \emptyset \vdash \text{if } T_1 : (q_0 \land q_1) \to q_0 & \Gamma_{21} \vdash T_2 : q_0 & \Gamma_{22} \vdash T_2 : q_1 \\
\text{App} & \Gamma_{21}, \Gamma_{22} \vdash \text{if } T_1 \ \ T_2 : q_0 & & \\
\end{array}
\]

Intersection types naturally lift to higher-order – and thus to $G$, which finitely represents $\langle G \rangle$.

**Theorem (Kobayashi)**

$S : q_0 \vdash S : q_0$ \iff the ATA $A_\varphi$ has a run-tree over $\langle G \rangle$. 
A type-system for verification: without parity conditions

Axiom

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

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

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

App

\[ \Delta \vdash t : (\theta_1 \wedge \cdots \wedge \theta_k) \to \theta :: \kappa \to \kappa' \]

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

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

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

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

\[ \text{fix} \]

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

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

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

Towards sequent calculus:

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

\[
\frac{\Delta_i \vdash u : \theta_i \quad \forall i \in \{1, \ldots, n\}}{\Delta_1, \ldots, \Delta_n \vdash t u : \theta'}
\]
A closer look at the Application rule

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

Right $\bigwedge$

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)

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

\[ \lbrack o \Rightarrow o \rbrack = \mathcal{P}_{fin}(Q) \times Q \]

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

**Order closure**

**Quantitative models** (Relational semantics)

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

\[ \lbrack o \Rightarrow o \rbrack = \mathcal{M}_{fin}(Q) \times Q \]

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

**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:

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

for a closed term.
Let $t$ be a term normalizing to a tree $\langle t \rangle$ and $A$ be an alternating automaton. 

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

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: if is a run-tree with two holes:

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

Set $\Omega(q_i) = i$.

\[
\begin{array}{c}
\lambda x \\
\lambda y \\
\lambda x
\end{array}
\]

\[
\begin{array}{cc}
a & q_1 \\
a & q_0 & a & q_1 \\
x & q_0 & y & q_1 & x & q_1 & x & q_1
\end{array}
\]

has type

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

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

Axiom

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

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

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

App

\[ \Delta \vdash t : (\bigwedge_{m_1} \epsilon \theta_1 \land \cdots \land \bigwedge_{m_k} \epsilon \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \]

\[ \Delta + \bigwedge_{m_1} \epsilon \Delta_1 + \cdots + \bigwedge_{m_k} \epsilon \Delta_k \vdash t u : \theta :: \kappa' \]

fix

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

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

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

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

A colored Application rule:

\[
\begin{align*}
\text{App} & \quad \Delta \vdash t : (\Box c_1 \theta_1 \land \cdots \land \Box c_k \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa \\
& \quad \Delta + \Box c_1 \Delta_1 + \ldots + \Box c_k \Delta_k \vdash t u : \theta :: \kappa'
\end{align*}
\]

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.
We now capture all MSO:

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

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

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

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

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

Towards sequent calculus:

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

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

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

the \textbf{finitary} model \( ScottL \) is the extensional collapse of \( Rel \).
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:

\[ \lambda_A : \downarrow \Box A \to \Box \downarrow A \]

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

Allows to compose comonads: $\downarrow \Box$ 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 $\Omega(q_i) = i$. 

\[
\begin{array}{c}
\lambda x \\
\lambda y \\
a \quad q_1 \\
a \quad q_0 \\
\quad x \quad q_0 \\
\quad y \\
\quad q_1 \\
a \quad q_1 \\
\quad q_1 \\
\end{array}
\]

has denotation 

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

(corresponding to the type $\square_0 q_0 \land \square_1 q_1 \rightarrow \square_1 q_1 \rightarrow q_1$)
An inductive-coinductive fixpoint operator

\[ \mathcal{Y} \text{ transports } f : \otimes X \otimes \otimes A \rightarrow A \]

into

\[ \mathcal{Y}_{X,A}(f) : \otimes X \rightarrow A. \]

via

\[ \mathcal{Y}_{X,A}(f) = \{ (w, a) | \exists \text{witness} \in \text{run-tree}(f, a) \text{ with } w = \text{leaves}(\text{witness}) \text{ and } \text{witness is accepting} \} \]
An inductive-coinductive fixpoint operator

\[ Y_{X,A} (f) = \{ (w, a) | \exists \text{witness} \in \text{run-tree}(f, a) \text{ with } w = \text{leaves}(\text{witness}) \text{ and } \text{witness} \text{ is accepting} \} \]

\text{witness} \text{ is built from finite pieces}

\begin{align*}
    (c, a') \\
    (c'_1, x_1) & \quad (c'_2, x_2) & \cdots & \quad (c_1, a_1) & \quad (c_2, a_2) & \cdots \\
\end{align*}

where

\[ ((([(c'_1, x_1), (c'_2, x_2), \ldots]], [(c_1, a_1), (c_2, a_2), \ldots]) , a') \in f \]

\text{leaves}(\text{witness}) \text{ is the colored multiset of the parameter leaves of } \text{witness}.

\( Y \) is a Conway operator, and Rel\( \downarrow \) is a model of the \( \lambda Y \)-calculus.
Conjecture

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

$q_0 \in \llbracket \lambda(\mathcal{G}) \rrbracket$

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

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 ∈ [[A(G)]]

**Corollary**

The higher-order model-checking problem is decidable.
Elements of proof

The proof proceeds by relating

- $\text{ScottL}_4$ to an intersection type system $S_{\text{fix}}(\mathcal{A})$ extending Terui’s with recursion and parity conditions,
- $S_{\text{fix}}(\mathcal{A})$ and the colored intersection type system presented earlier, on $\eta$-long $\beta$-normal forms
- and by using our modified version of the soundness-and-completeness theorem of Kobayashi and Ong.
Decidability

Checking whether $q_0 \in \llbracket G \rrbracket \iff$ solving a parity game on a finite fragment of $ScottL_\downarrow$, which is decidable.

We also obtain memoryless strategies which correspond to regular typings in $S_{fix}(\mathcal{A})$:

$$
\begin{align*}
\pi_1 \\
F_1 : \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\}, \ldots \vdash R(F_i) : \alpha \\
\emptyset \vdash \mu F_k'' : \beta_{j''_k} \\
\emptyset \vdash \mu F_k : \beta_{j_k} \\
\emptyset \vdash \mu F_k' : \beta'_{j'_k} \\
\emptyset \vdash \mu F_1 : q
\end{align*}
$$

A key to the selection property.
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...
- Towards the model-checking of other classes of properties?

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... 
- Towards the model-checking of other classes of properties?

Thank you for your attention!