Semantics of linear logic
and higher-order model-checking

Charles Grellois    Paul-André Melliès

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

University of Bologna
January 20, 2016
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
- Interaction: the result is whether

$$M \models \varphi$$

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

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

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

Example:

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

modelled as

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

Model-checking higher-order programs

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

Example:

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

modelled as

```
if
   if
      data
         if
            Nil
            data
            Nil
   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)

Can we decide whether a higher-order tree satisfies a MSO formula?
Automata theory, typing, and recognition by homomorphism
A very naive model-checking problem

A simpler problem first: execution traces as finite words, properties as finite automata.

A word of actions:

\[ \text{open} \cdot (\text{read} \cdot \text{write})^2 \cdot \text{close} \]

A property to check: is every \textit{read} immediately followed by a \textit{write}?

→ automaton with two states: \( Q = \{ q_0, q_{read} \} \).

\( q_0 \) is both initial and final.
A type-theoretic intuition

\[ \delta(q_0, \text{read}) = q_{\text{read}} \]

corresponds to the typing

\[ \text{read} : q_{\text{read}} \rightarrow q_0 \]

refining the simple type

\[ o \rightarrow o \]

Type of a word: a state from which it is accepted.
A type-theoretic intuition: a run of the automaton

\[ \vdash open : q_0 \to q_0 \quad \vdash (read \cdot write)^2 \cdot close : q_0 \]

\[ \vdash open \cdot (read \cdot write)^2 \cdot close : q_0 \]
A type-theoretic intuition: a run of the automaton

\[\vdash \text{write} : q_0 \rightarrow q_{\text{read}} \quad \vdash \text{read} \cdot \text{write} \cdot \text{close} : q_0\]

\[\vdash \text{read} : q_{\text{read}} \rightarrow q_0 \quad \vdash \text{write} \cdot \text{read} \cdot \text{write} \cdot \text{close} : q_{\text{read}}\]

\[\vdash (\text{read} \cdot \text{write})^2 \cdot \text{close} : q_0\]

and so on.

Typing naturally extends to terms. Subject reduction/expansion allow some static analysis.

Let’s do the same for recursion schemes – which compute trees.
Automata and recognition

Given a 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 \( \phi : A^* \rightarrow M \) such that \( L = \phi^{-1}(K) \).

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

Extension to terms of this recognition by morphism, using domains (Salvati 2009).
Automata and recognition

Given a 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.

Extension to terms of this recognition by morphism, using domains
(Salvati 2009).
Higher-order recursion schemes

Some regularity for infinite trees
Higher-order recursion schemes

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

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

![Tree Diagram]
Higher-order recursion schemes

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

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

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

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

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

\[
\langle G \rangle = 
\]

\[
\text{if}
\]

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

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

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

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

“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

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

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

\[ \mathcal{A}_\varphi \text{ has a winning run-tree over } \langle G \rangle \text{ iff } \langle G \rangle \models \varphi. \]
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 if \( T_1 \ T_2 \):

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

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

\[
\frac{\forall \{i\} \theta_i :: \kappa}{\vdash x : \theta_i :: \kappa}
\]

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

\[
\frac{\emptyset \vdash a : \bigwedge_{j=1}^{k_1} q_{1j} \rightarrow \cdots \rightarrow \bigwedge_{j=1}^{k_n} q_{nj} \rightarrow q :: o \rightarrow \cdots \rightarrow o}{\Delta \vdash t : (\theta_1 \land \cdots \land \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa}
\]

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

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

\[
\frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \theta :: \kappa \vdash F : \theta :: \kappa}
\]
A closer look at the Application rule

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

Towards sequent calculus:

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

\[
\Delta, \Delta_1, \ldots, \Delta_n \vdash t \ u : \theta'
\]
A closer look at the Application rule

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

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

Linear decomposition of the intuitionistic arrow:

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

Two steps: duplication / erasure, then linear use.

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

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

Two interpretations of the exponential modality:

**Qualitative models** (Scott semantics)

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

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

\( \{q_0, q_0, q_1\} = \{q_0, q_1\} \)

Order closure

**Quantitative models** (Relational semantics)

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

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

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

Unbounded multiplicities
Intersection types and semantics of linear logic

Fundamental idea:

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

and similarly for open terms.
Intersection types and semantics of linear logic

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: if is a run-tree with two holes:

\[ \text{if} \]

\[ [\ ]q_0 \quad [\ ]q_1 \]

A new neutral color: \( \epsilon \) for an empty term \([\ ]_q\). Goal: subject reduction/expansion.
A type-system for verification

\[
\Delta \vdash t : (\square c_1 \theta_1 \land \cdots \land \square c_k \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa
\]
\[
\Delta + \square c_1 \Delta_1 + \cdots + \square c_k \Delta_k \vdash t u : \theta :: \kappa'
\]

Subject reduction: the contraction of a redex

\[
\Delta, x : \square \epsilon \theta_1 \land \cdots \land \square \epsilon \theta_2 \vdash x : \theta
\]
\[
\Delta \vdash \lambda x. t : (\square c_1 \theta_1 \land \cdots \land \square c_k \theta_k) \rightarrow \theta
\]
\[
\Delta + \square c_1 \Delta_1 + \cdots + \square c_k \Delta_k \vdash (\lambda x. t) \ u : \theta
\]
A type-system for verification

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

\[
\Delta + \Box c_1 \Delta_1 + \cdots + \Box c_k \Delta_k \vdash t u : \theta :: \kappa'
\]
gives a proof of the same sequent:

\[
\Delta + \Box c_1 \Delta_1 + \cdots + \Box c_k \Delta_k \vdash t[x \leftarrow u] : \theta
\]
A type-system for verification

Axiom

\[ x : \bigwedge \{i\} \square_{\epsilon} \theta_i :: \kappa \vdash x : \theta_i :: \kappa \]

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

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

App

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

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

fix

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

\[ F : \square_{\epsilon} \theta :: \kappa \vdash F : \theta :: \kappa \]

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

\[ \Delta \vdash \lambda x . t : \left( \bigwedge_{i \in I} \square_{m_i} \theta_i \right) \rightarrow \theta :: \kappa \rightarrow \kappa' \]
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 collapsing to *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

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

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

Towards sequent calculus:

\[
\Delta_1 \vdash u : \theta_1 \\
\Box m_1 \Delta_1 \vdash u : \Box m_1 \theta_1 \\
\vdots \\
\Delta_n \vdash u : \theta_n \\
\Box m_n \Delta_n \vdash u : \Box m_n \theta_1
\]

Right \Box

Right \land

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

\[
\Delta, \Box m_1 \Delta_1, \ldots, \Box m_n \Delta_n \vdash t u : \theta
\]

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 which distributes with the exponential in the semantics.
Colored semantics

We extend:

- $\text{Rel}$ with \textbf{countable} multiplicities, \textbf{coloring} and an \textbf{inductive-coinductive} fixpoint
- $\text{ScottL}$ 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 $\text{ScottL}$ is the extensional collapse of $\text{Rel}$. 
Model-checking and finitary semantics

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

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

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

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.*
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!