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 $\mathcal{M}$ of a program
- Specify a property φ in an appropriate logic
- Interaction: the result is whether

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

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

$$\varphi \quad \mapsto \quad \mathcal{A}_{\varphi}$$

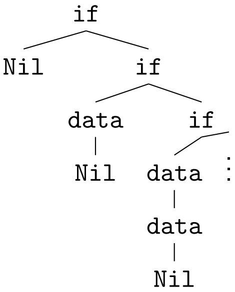
Model-checking higher-order programs

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

Example:

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen } x & = & \texttt{if } \mathit{end} \texttt{ then } x \texttt{ else Listen (data } x\texttt{)} \end{array}$$

modelled as



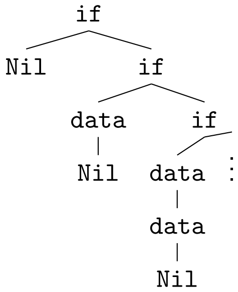
Model-checking higher-order programs

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

Example:

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen } x & = & \texttt{if } \mathit{end} \texttt{ then } x \texttt{ else Listen (data } x\texttt{)} \end{array}$$

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 tree

over which we run

an alternating parity tree automaton (APT) $\mathcal{A}_\varphi$

corresponding to a

monadic second-order logic (MSO) formula φ.

(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 :

$$open \;\cdot\; (read \cdot write)^2 \cdot close$$

A property to check: is every *read* immediately followed by a *write* ?

$\rightarrow$ automaton with two states: $Q \;=\; \{q_0,\, q_{read}\}$.

q_0 is both initial and final.

A type-theoretic intuition

$$\delta(q_0,\, read) \;=\; q_{read}$$

corresponds to the typing

$$read \;:\; q_{read} \rightarrow q_0$$

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

$$\dfrac{\vdash open \ : \ q_0 \to q_0 \qquad \dfrac{}{\vdash \ (read \cdot write)^2 \cdot close \ : \ q_0}}{\vdash open \ \cdot \ (read \cdot write)^2 \cdot close \ : \ q_0}$$

A type-theoretic intuition: a run of the automaton

$$\dfrac{\vdash read \;:\; q_{read} \rightarrow q_0 \qquad \dfrac{\vdash write \;:\; q_0 \rightarrow q_{read} \qquad \dfrac{}{\vdash \; read \cdot write \cdot close \;:\; q_0}}{\vdash \; write \cdot read \cdot write \cdot close \;:\; q_{read}}}{\dfrac{\vdash \; (read \cdot write)^2 \cdot close \;:\; q_0}{\vdots}}$$

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 $\mathcal{A}$ 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).

Automata and recognition

Given a language $L \subseteq A^*$,

there exists a finite automaton $\mathcal{A}$ 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

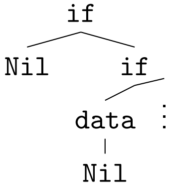
Higher-order recursion schemes

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen } x & = & \texttt{if } \textit{end} \texttt{ then } x \texttt{ else Listen (data } x\texttt{)} \end{array}$$

is abstracted as

$$\mathcal{G} \;=\; \begin{cases} \texttt{S} & = \quad \texttt{L Nil} \\ \texttt{L } x & = \quad \texttt{if } x \,(\texttt{L } (\texttt{data } x\,)\,) \end{cases}$$

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



Higher-order recursion schemes

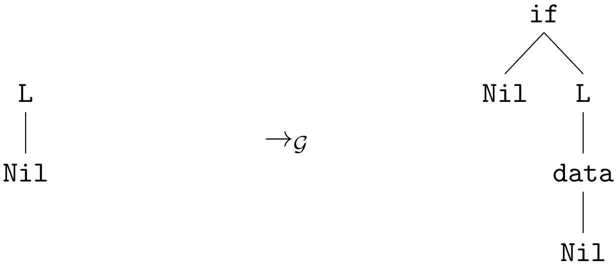
$$\mathcal{G} \quad = \quad \begin{cases} \mathtt{S} & = \quad \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$

Rewriting starts from the start symbol S:

$$\mathtt{S} \quad \rightarrow_{\mathcal{G}} \quad \begin{array}{c} \mathtt{L} \\ | \\ \mathtt{Nil} \end{array}$$

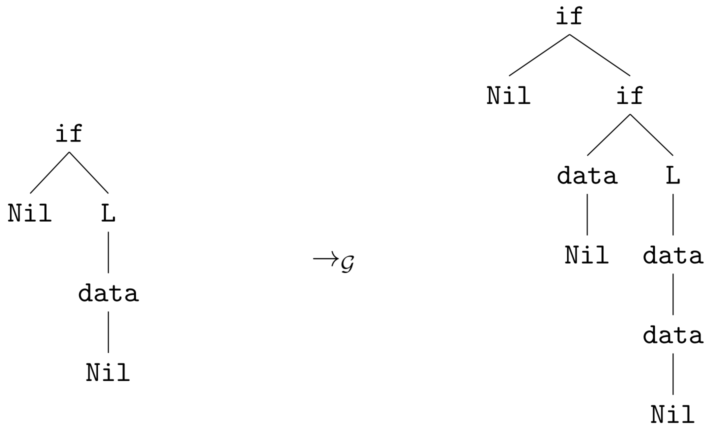
Higher-order recursion schemes

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$



Higher-order recursion schemes

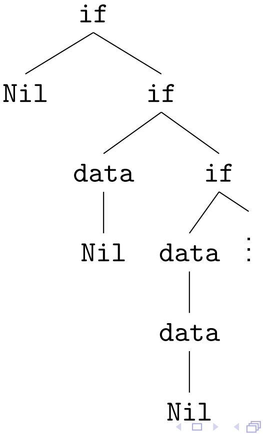
$$\mathcal{G} = \begin{cases} \mathtt{S} & = \quad \mathtt{L\ Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



Higher-order recursion schemes

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$

$\langle \mathcal{G} \rangle \quad =$



Higher-order recursion schemes

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$

"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 Ω in one step).

Higher-order recursion schemes

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

"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 Ω in one step).

HORS can alternatively be seen as simply-typed λ-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 φ,

$$\langle \mathcal{G} \rangle \quad \vDash \quad \varphi$$

iff an equivalent APT $\mathcal{A}_\varphi$ has a run over $\langle \mathcal{G} \rangle$.

APT $\quad = \quad$ alternating tree automata (ATA) + parity condition.

Alternating tree automata

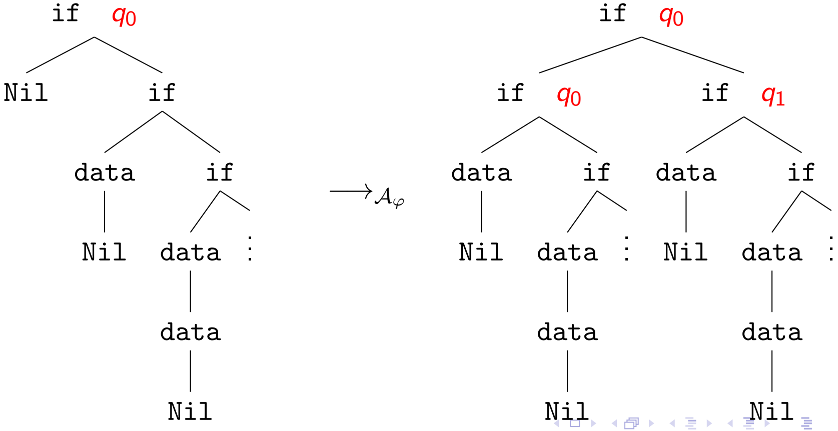
ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

Typically: $\delta(q_0, \texttt{if}) \ = \ (2, q_0) \wedge (2, q_1)$.

Alternating tree automata

ATA: non-deterministic tree automata whose transitions may
duplicate or drop a subtree.

Typically: $\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$.



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

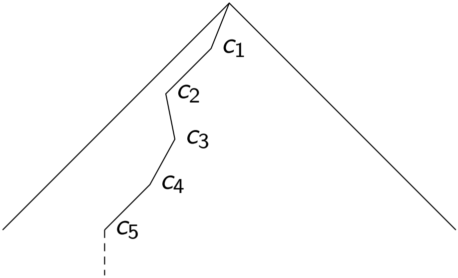
> "after a `read` operation, a `write` eventually occurs".

Alternating parity tree automata

Each state of an APT is attributed a color

$$\Omega(q) \in Col \subseteq \mathbb{N}$$

An infinite branch of a run-tree is winning iff the maximal color among the ones occuring infinitely often along it is even.



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 occuring infinitely often along it is even.

A run-tree is winning iff all its infinite branches are.

For a MSO formula φ:

$$\mathcal{A}_{\varphi} \text{ has a winning run-tree over } \langle \mathcal{G} \rangle \text{ iff } \langle \mathcal{G} \rangle \vDash \phi.$$

Intersection types and alternation

Alternating tree automata and intersection types

A key remark (Kobayashi 2009):

$$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$$

can be seen as the intersection typing

$$\texttt{if} \;:\; \emptyset \rightarrow (q_0 \wedge q_1) \rightarrow q_0$$

refining the simple typing

$$\texttt{if} \;:\; o \rightarrow o \rightarrow o$$

(this talk is NOT about filter models!)

Alternating tree automata and intersection types

In a derivation typing `if` $T_1 \; T_2$:

$$
\dfrac{\text{App}\;\dfrac{\delta\;\dfrac{}{\emptyset \vdash \mathtt{if} : \emptyset \to (q_0 \wedge q_1) \to q_0} \qquad \emptyset}{\emptyset \vdash \mathtt{if}\ T_1 : (q_0 \wedge q_1) \to q_0} \qquad \dfrac{\vdots}{\Gamma_{21} \vdash T_2 : q_0} \qquad \dfrac{\vdots}{\Gamma_{22} \vdash T_2 : q_1}}{\Gamma_{21}, \Gamma_{22} \vdash \mathtt{if}\ T_1\ T_2 : q_0}\;\text{App}
$$

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 $$\frac{}{x \,:\, \bigwedge_{\{i\}} \;\theta_i \,::\, \kappa \;\vdash\; x \,:\, \theta_i \,::\, \kappa}$$

δ $$\frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\,\} \quad \text{satisfies} \quad \delta_A(q, a)}{\emptyset \vdash a \,:\, \bigwedge_{j=1}^{k_1} \; q_{1j} \;\rightarrow\; \ldots \;\rightarrow\; \bigwedge_{j=1}^{k_n} \; q_{nj} \rightarrow q \,::\, o \rightarrow \cdots \rightarrow o}$$

App $$\frac{\Delta \vdash t : (\;\theta_1 \;\wedge \cdots \wedge\; \theta_k) \rightarrow \theta \,::\, \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u \,:\, \theta_i \,::\, \kappa}{\Delta\,,\,\Delta_1\,,\,\ldots\,,\,\Delta_k \;\vdash\; t\,u \,:\, \theta \,::\, \kappa'}$$

λ $$\frac{\Delta\,,\,x \,:\, \bigwedge_{i \in I} \;\theta_i \,::\, \kappa \;\vdash\; t \,:\, \theta \,::\, \kappa'}{\Delta \;\vdash\; \lambda x\,.\,t \,:\, \left(\bigwedge_{i \in I} \;\theta_i\right) \rightarrow \theta \,::\, \kappa \rightarrow \kappa'}$$

fix $$\frac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{\overline{F \,:\, \theta \,::\, \kappa \;\vdash\; F \,:\, \theta \,::\, \kappa}}$$

A closer look at the Application rule

$$\text{App} \qquad \frac{\Delta \vdash t : (\,\theta_1 \ \wedge \cdots \wedge \ \theta_k) \rightarrow \theta \ :: \ \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u \ : \ \theta_i \ :: \ \kappa}{\Delta\,,\, \Delta_1\,,\, \ldots\,,\, \Delta_k \ \vdash \ \ t\, u \ : \ \theta :: \ \kappa'}$$

Towards sequent calculus:

$$\dfrac{\Delta \ \vdash \ t \ : \ (\,\bigwedge_{i=1}^{n} \ \theta_i\,) \rightarrow \theta' \qquad \dfrac{\Delta_i \ \vdash \ u \ : \ \theta_i \qquad \forall i \in \{1, \ldots\, n\}}{\Delta_1, \ldots, \Delta_n \ \vdash \ u \ : \ \bigwedge_{i=1}^{n} \ \theta_i} \ \text{Right} \bigwedge}{\Delta,\, \Delta_1, \ldots,\, \Delta_n \ \vdash \ t \ u \ : \ \theta'}$$

A closer look at the Application rule

$$\dfrac{\Delta \;\vdash\; t : (\bigwedge_{i=1}^{n}\ \theta_i) \to \theta' \qquad \dfrac{\Delta_i \;\vdash\; u : \theta_i \qquad \forall i \in \{1, \ldots n\}}{\Delta_1, \ldots, \Delta_n \;\vdash\; u : \bigwedge_{i=1}^{n}\ \theta_i}\ \text{Right } \bigwedge}{\Delta, \Delta_1, \ldots, \Delta_n \;\vdash\; t\ u : \theta'}$$

Linear decomposition of the intuitionnistic 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)

$$!\,A \;=\; \mathcal{P}_{fin}(A)$$

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

$$\{q_0,\, q_0,\, q_1\} \;=\; \{q_0,\, q_1\}$$

Order closure

Quantitative models
(Relational semantics)

$$!\,A \;=\; \mathcal{M}_{fin}(A)$$

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

$$[q_0,\, q_0,\, q_1] \;\neq\; [q_0,\, q_1]$$

Unbounded multiplicities

Intersection types and semantics of linear logic

$$\begin{tikzcd}
Rel_! \arrow[rr, leftrightarrow, "Bucciareli-Ehrhard", "de\ \ Carvalho"'] \arrow[d, "Ehrhard"'] & & \text{Non-idempotent types} \arrow[d, "{Ehrhard,\ G-M}"] \\
ScottL_! \arrow[rr, leftrightarrow, "Ehrhard", "Terui"'] & & \text{Idempotent types}
\end{tikzcd}$$

Fundamental idea:

$$[\![t]\!] \;\cong\; \{\, \theta \mid \emptyset \vdash t : \theta \,\}$$

and similarly for open terms.

Intersection types and semantics of linear logic

$$\begin{array}{ccc} Rel_! & \xleftrightarrow[de\ \ Carvalho]{Bucciareli-Ehrhard} & \text{Non-idempotent types} \\ \downarrow{\scriptstyle Ehrhard} & & \downarrow{\scriptstyle Ehrhard,\ G-M} \\ ScottL_! & \xleftrightarrow[Terui]{Ehrhard} & \text{Idempotent types} \end{array}$$

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

$$\mathcal{A} \text{ accepts } \langle t \rangle \text{ from } q \quad \Leftrightarrow \quad q \in [\![t]\!] \quad \Leftrightarrow \quad \emptyset \ \vdash \ t \ : \ q \ :: \ o$$

Extension with recursion and parity condition?

Adding parity conditions to the type system

Alternating parity tree automata

We add coloring annotations to intersection types:

$$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$$

now corresponds to

$$\texttt{if} \;:\; \emptyset \rightarrow \left(\Box_{\Omega(q_0)}\, q_0 \wedge \Box_{\Omega(q_1)}\, q_1 \right) \rightarrow q_0$$

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

$$\begin{array}{c} \texttt{if} \\ \diagup \quad \diagdown \\ [\,]_{q_0} \quad [\,]_{q_1} \end{array}$$

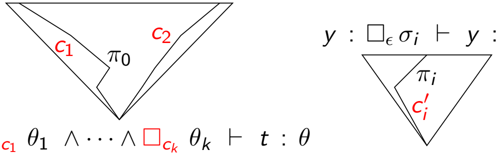
A new neutral color: ϵ for an empty term $[\,]_q$. Goal: subject reduction/expansion.

A type-system for verification

App $$\frac{\Delta \vdash t : (\Box_{c_1}\ \theta_1\ \wedge \cdots \wedge \Box_{c_k}\ \theta_k) \rightarrow \theta\ ::\ \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u\ :\ \theta_i\ ::\ \kappa}{\Delta\ +\ \Box_{c_1}\Delta_1\ +\ \ldots\ +\ \Box_{c_k}\Delta_k\ \vdash\ \ t\,u\ :\ \theta :: \ \kappa'}$$

Subject reduction: the contraction of a redex

$$x\ :\ \Box_\epsilon\,\theta_1\ \vdash\ x\ :\ \theta_1 \qquad x\ :\ \Box_\epsilon\,\theta_2\ \vdash\ x\ :\ \theta_2 \qquad\qquad y\ :\ \Box_\epsilon\,\sigma_i\ \vdash\ y\ :\ \sigma_i$$

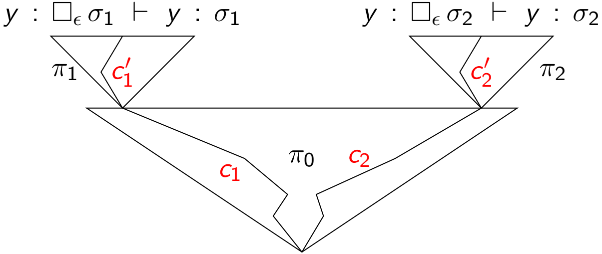


$$\cfrac{\cfrac{\Delta,\, x\ :\ \Box_{c_1}\ \theta_1\ \wedge \cdots \wedge \Box_{c_k}\ \theta_k\ \vdash\ \ t\ :\ \theta}{\Delta\ \vdash\ \ \lambda x.\, t\ :\ (\Box_{c_1}\ \theta_1\ \wedge \cdots \wedge \Box_{c_k}\ \theta_k) \rightarrow \theta} \qquad \Delta_i\ \vdash\ \ u\ :\ \theta_i}{\Delta\ +\ \Box_{c_1}\Delta_1\ +\ \ldots\ +\ \Box_{c_k}\Delta_k \vdash\ (\,\lambda x.\, t\,)\ \ u\ :\ \theta}$$

A type-system for verification

$$\text{App} \quad \frac{\Delta \vdash t : (\square_{c_1}\ \theta_1\ \wedge \cdots \wedge \square_{c_k}\ \theta_k) \rightarrow \theta\ ::\ \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u\ :\ \theta_i\ ::\ \kappa}{\Delta + \square_{c_1}\Delta_1 + \ldots + \square_{c_k}\Delta_k\ \vdash\ t\,u\ :\ \theta ::\ \kappa'}$$

gives a proof of the same sequent:



$$\overline{\Delta + \square_{c_1}\Delta_1 + \ldots + \square_{c_k}\Delta_k \vdash\ t[x \leftarrow u]\ :\ \theta}$$

A type-system for verification

$$\text{Axiom} \qquad \frac{}{x \,:\, \bigwedge_{\{i\}} \square_{\epsilon}\ \theta_i\ ::\ \kappa\ \vdash\ x\,:\,\theta_i\ ::\ \kappa}$$

$$\delta \qquad \frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\}\quad \text{satisfies}\quad \delta_A(q, a)}{\emptyset \vdash a\,:\, \bigwedge_{j=1}^{k_1} \square_{\Omega(q_{1j})}\ q_{1j} \rightarrow \ldots \rightarrow \bigwedge_{j=1}^{k_n} \square_{\Omega(q_{nj})}\ q_{nj} \rightarrow q\ ::\ o \rightarrow \cdots \rightarrow o \rightarrow o}$$

$$\text{App} \qquad \frac{\Delta \vdash t : (\square_{m_1}\ \theta_1\ \wedge \cdots \wedge \square_{m_k}\ \theta_k) \rightarrow \theta\ ::\ \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u\,:\,\theta_i\ ::\ \kappa}{\Delta + \square_{m_1}\Delta_1 + \ldots + \square_{m_k}\Delta_k\ \vdash\ t\,u\,:\,\theta :: \kappa'}$$

$$\mathit{fix} \qquad \frac{\Gamma \vdash \mathcal{R}(F)\,:\,\theta\ ::\ \kappa}{F\,:\,\square_{\epsilon}\ \theta\ ::\ \kappa\ \vdash\ F\,:\,\theta\ ::\ \kappa}$$

$$\lambda \qquad \frac{\Delta\,,\,x\,:\,\bigwedge_{i \in I} \square_{m_i}\ \theta_i\ ::\ \kappa\ \vdash\ t\,:\,\theta\ ::\ \kappa'}{\Delta\ \vdash\ \lambda x\,.\,t\,:\,\left(\bigwedge_{i \in I} \square_{m_i}\ \theta_i\right) \rightarrow \theta :: \kappa \rightarrow \kappa'}$$

A type-system for verification

We rephrase the parity condition to typing trees, and now capture all MSO:

Theorem (G.-Melliès 2014)

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

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

$$\frac{\Delta \vdash t : (\Box_{m_1}\ \theta_1\ \wedge \cdots \wedge \Box_{m_k}\ \theta_k) \rightarrow \theta\ ::\ \kappa \rightarrow \kappa' \qquad \Delta_i \vdash u\ :\ \theta_i\ ::\ \kappa}{\Delta\ +\ \Box_{m_1}\Delta_1\ +\ \ldots\ +\ \Box_{m_k}\Delta_k\ \vdash\ t\,u\ :\ \theta :: \kappa'}$$

Towards sequent calculus:

$$\dfrac{\Delta \vdash t : (\bigwedge_{i=1}^{n}\ \Box_{m_i}\ \theta_i\,) \rightarrow \theta \qquad \dfrac{\dfrac{\Delta_1 \vdash u\ :\ \theta_1}{\Box_{m_1}\ \Delta_1\ \vdash\ u\ :\ \Box_{m_1}\ \theta_1} \quad \ldots \quad \dfrac{\Delta_n \vdash u\ :\ \theta_n}{\Box_{m_n}\ \Delta_n\ \vdash\ u\ :\ \Box_{m_n}\ \theta_1}\ \text{Right}\ \Box}{\Box_{m_1}\Delta_1,\ \ldots,\ \Box_{m_n}\Delta_n\ \vdash\ u\ :\ \bigwedge_{i=1}^{n}\ \Box_{m_i}\ \theta_i}\ \text{Right}\ \bigwedge}{\Delta,\ \Box_{m_1}\Delta_1,\ \ldots,\ \Box_{m_n}\Delta_n\ \vdash\ t\,u\ :\ \theta}$$

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:

- *Rel* with countable multiplicites, coloring and an inductive-coinductive fixpoint
- *ScottL* with coloring and an inductive-coinductive fixpoint.

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

the finitary model *ScottL* is the extensional collapse of *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*:

$$\mathcal{A} \text{ accepts } \langle \mathcal{G} \rangle \text{ from } q \quad \Leftrightarrow \quad q \in [\![t]\!]$$

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.

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!