A semantic study of higher-order model-checking

Charles Grellois   Paul-André Melliès

PPS & LIAFA — Université Paris 7
University of Dundee

Journées Nationales GEOCAL-LAC-LTP 2015
October 14th, 2015

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
- Make them interact: the result is whether

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

When the model is a word, a tree... of actions: translate φ to an equivalent automaton:

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

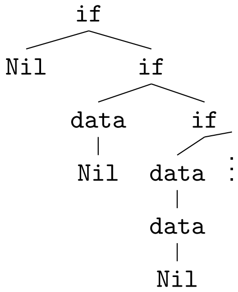
Model-checking higher-order programs

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



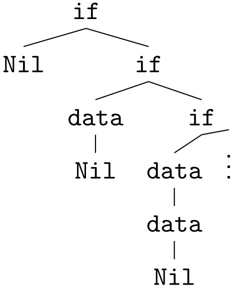
Model-checking higher-order programs

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

Example:

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

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)

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?

Higher-order recursion schemes

Some regularity for infinite trees

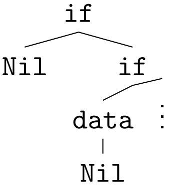
Higher-order recursion schemes

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

is abstracted as

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

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



Higher-order recursion schemes

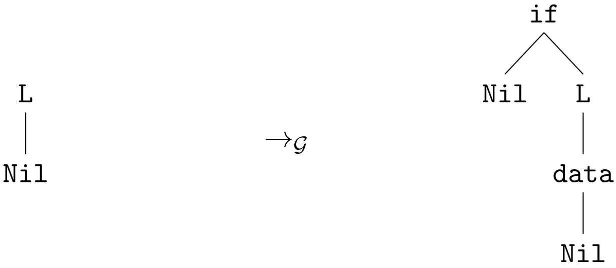
$$\mathcal{G} = \begin{cases} \mathtt{S} & = \quad \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$

Rewriting starts from the start symbol S:

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

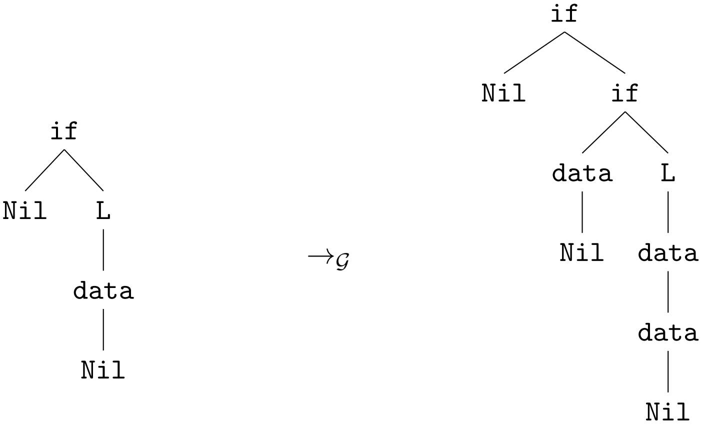
Higher-order recursion schemes

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



Higher-order recursion schemes

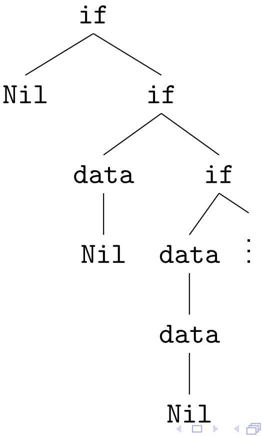
$$\mathcal{G} = \begin{cases} \mathtt{S} & = \quad \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



Higher-order recursion schemes

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

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



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

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

"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 $Y_\sigma \;:\; (\sigma \rightarrow \sigma) \rightarrow \sigma$.

Higher-order recursion schemes

We can adapt to HORS the fact that coinductive parallel head reduction computes the normal form of infinite λ-terms:

$$\frac{}{(\lambda x.\, s)\ t \quad \rightarrow_{\mathcal{G}w} \quad s[x \leftarrow t]} \qquad \frac{s \quad \rightarrow_{\mathcal{G}w} \quad s'}{s\ t \quad \rightarrow_{\mathcal{G}w} \quad s'\ t}$$

$$\frac{}{F \quad \rightarrow_{\mathcal{G}w} \quad \mathcal{R}(F)}$$

$$\frac{t \quad \rightarrow_{\mathcal{G}w}^{*} \quad a\ t_1 \cdots t_n \qquad t_i \quad \rightarrow_{\mathcal{G}}^{\infty} \quad t_i' \quad (\forall i)}{t \quad \rightarrow_{\mathcal{G}}^{\infty} \quad a\ t_1' \cdots t_n'}$$

This reduction computes $\langle \mathcal{G} \rangle$ whenever it exists (a decidable question).

This presentation allows coinductive reasoning on rewriting.

Alternating 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 $\langle \mathcal{G} \rangle$.

$$\text{APT} \quad = \quad \underbrace{\underbrace{\text{alternating tree automata (ATA)}}_{\text{weak MSO}} + \text{parity condition.}}_{\text{MSO}}$$

Alternating tree automata

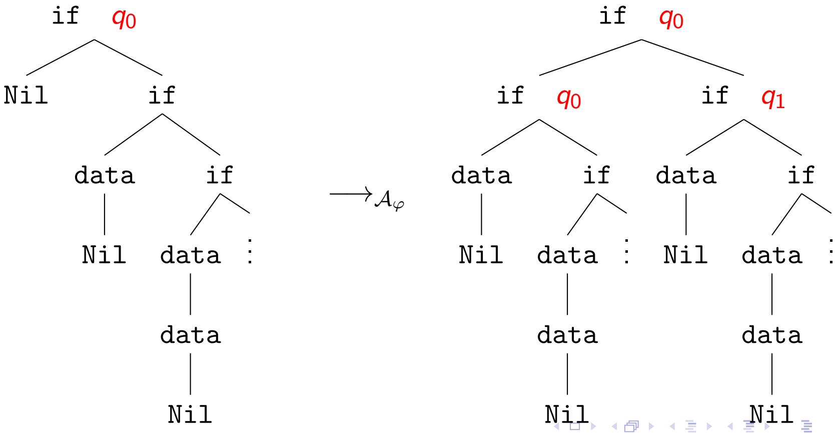
ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

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



Alternating tree automata

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

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

This infinite process produces a run-tree of $\mathcal{A}_\varphi$ over $\langle \mathcal{G} \rangle$.

It is an infinite, unranked tree.

ATA vs. HORS

$$\frac{}{(\,\lambda x.\,s\,)\ t \quad \rightarrow_{\mathcal{G}w} \quad s[x \leftarrow t]} \qquad \frac{s \quad \rightarrow_{\mathcal{G}w} \quad s'}{s\ t \quad \rightarrow_{\mathcal{G}w} \quad s'\ t}$$

$$\frac{}{F \quad \rightarrow_{\mathcal{G}w} \quad \mathcal{R}(F)}$$

$$\frac{t \quad \rightarrow_{\mathcal{G}w}^{*} \quad a\ t_1\ \cdots\ t_n \qquad t_i\ :\ q_{ij} \quad \rightarrow_{\mathcal{G},\mathcal{A}}^{\infty} \quad t'_i\ :\ q_{ij}}{\overline{t\ :\ q \quad \rightarrow_{\mathcal{G},\mathcal{A}}^{\infty} \quad (a\ (t'_{11}\ :\ (1, q_{11}))\ \cdots\ (t'_{nk_n}\ :\ (n, q_{nk_n})))\ :\ q}}$$

where the duplication "conforms to δ" (there is non-determinism).

Starting from $S\ :\ q_0$, this computes run-trees of an ATA $\mathcal{A}$ over $\langle \mathcal{G} \rangle$.

We get closer to type theory...

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 $\texttt{if}\ \ T_1\ \ T_2$:

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

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

An alternate proof

Theorem

$S : q_0 \vdash S : q_0$ *iff the ATA* $\mathcal{A}_\phi$ *has a run-tree over* $\langle \mathcal{G} \rangle$.

Proof: coinductive subject reduction/expansion + head reduction of derivations with non-idempotent intersection types.

$$\dfrac{\begin{array}{c}\pi \\ \vdots\end{array}}{S : q_0 \vdash S : q_0} \quad \longleftrightarrow \quad \dfrac{\begin{array}{c}\pi' \\ \vdots\end{array}}{\emptyset \vdash \langle \mathcal{G} \rangle : q_0} \quad \Longleftrightarrow$$

$\langle \mathcal{G} \rangle$ is accepted by $\mathcal{A}$.

Parity conditions

Alternating parity tree automata

MSO allows to discriminate inductive from coinductive behaviour.

This allows to express properties as

"a given operation is executed infinitely often in some execution"

or

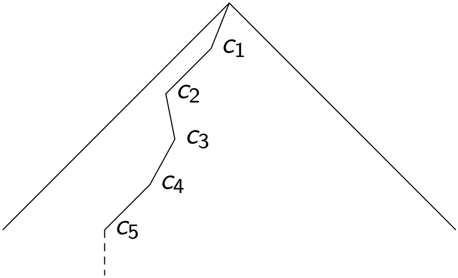
"after a `read` operation, a `write` eventually occurs".

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

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

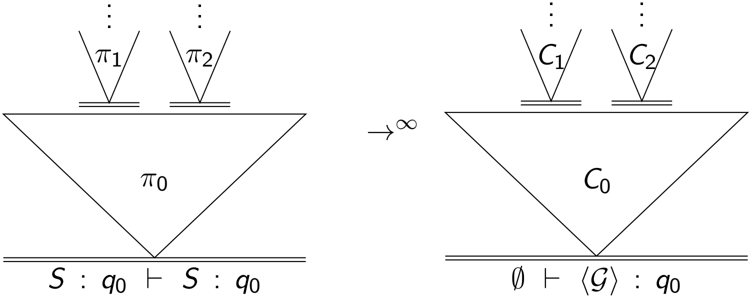
An infinite branch of a run-tree is winning iff the maximal color among the ones occuring infinitely often along it is even.

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

For a MSO formula φ:

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

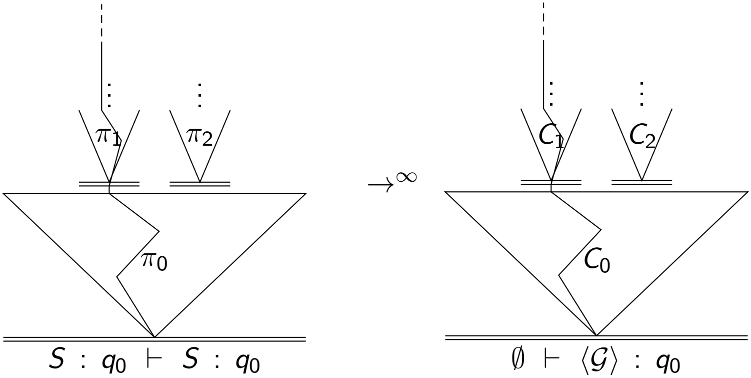
One more word on proof rewriting



where the C_i are the tree contexts obtained by normalizing each π_i.

$C_0[C_1[], C_2[]]$ is a prefix of a run-tree of $\mathcal{A}$ over $\langle \mathcal{G} \rangle$.

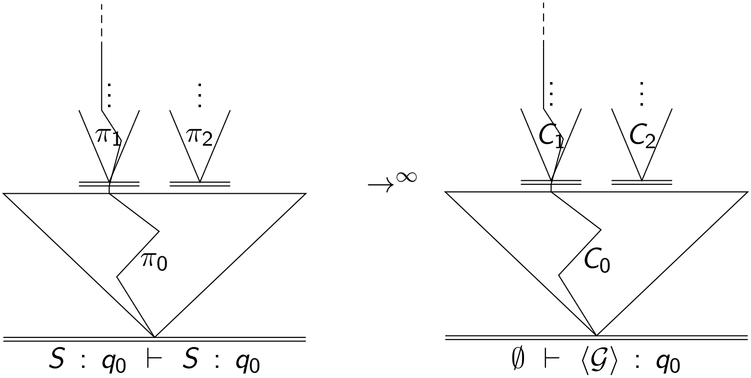
One more word on proof rewriting



Theorem

In this quantitative setting, there is a correspondence between infinite branches of the typing of $\mathcal{G}$ and of the run-tree over $\langle \mathcal{G} \rangle$ obtained by normalization.

One more word on proof rewriting



The goal now: add information in π_i about the maximal color seen in C_i.

One extra color: ϵ for the case $C_i = []$.

Alternating parity tree automata

We add coloring informations to intersection types:

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

now corresponds to

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

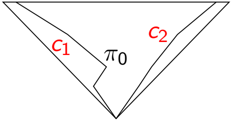
Application computes the “local” maximum of colors, and the fixpoint deals with the acceptance condition.

A type-system for verification (Grellois-Melliès 2014)

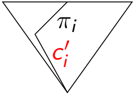
$$\text{App} \qquad \frac{\Delta \vdash t : (\Box_{c_1}\ \theta_1\ \wedge \cdots \wedge \Box_{c_k}\ \theta_k) \to \theta\ ::\ \kappa \to \kappa' \qquad \Delta_i \vdash u\ :\ \theta_i\ ::\ \kappa}{\Delta\ +\ \Box_{c_1}\Delta_1\ +\ \ldots\ +\ \Box_{c_k}\Delta_k\ \vdash\ t\,u\ :\ \theta :: \kappa'}$$

Subject reduction: the contraction of a redex

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



$$y\ :\ \Box_\epsilon\,\sigma_i\ \vdash\ y\ :\ \sigma_i$$



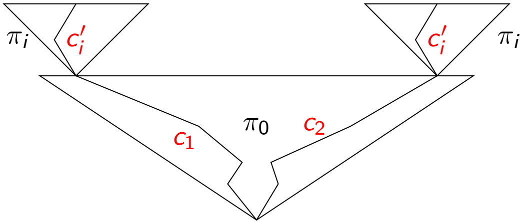
$$\frac{\dfrac{\Delta, x\ :\ \Box_{c_1}\ \theta_1\ \wedge \cdots \wedge \Box_{c_k}\ \theta_k\ \vdash\ t\ :\ \theta}{\Delta\ \vdash\ \lambda x.\,t\ :\ (\Box_{c_1}\ \theta_1\ \wedge \cdots \wedge \Box_{c_k}\ \theta_k) \to \theta} \qquad \Delta_i\ \vdash\ u\ :\ \theta_i}{\Delta\ +\ \Box_{c_1}\Delta_1\ +\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

A type-system for verification (Grellois-Melliès 2014)

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

gives a proof of the same sequent:

$$y : \Box_\epsilon\, \sigma_i \ \vdash\ y : \sigma_i \qquad\qquad y : \Box_\epsilon\, \sigma_i \ \vdash\ y : \sigma_i$$



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

A type-system for verification (Grellois-Melliès 2014)

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.

It was linear logic all the way!

Linear logic very naturally handles alternation via

$$A \Rightarrow B \quad = \quad !A \multimap B$$

and we can extend it with a coloring modality $\Box$.

New colored, infinitary semantics:

$$\lightning\, A \quad = \quad \mathcal{M}_{count}(Col \times A)$$

Quantitative colored intersection types $\Leftrightarrow$ elements of this colored, infinitary relational semantics.

Typing derivations $\Leftrightarrow$ computation of denotations.

It was linear logic all the way!

We obtain two kind of semantics:

- a quantitative, infinitary semantics, corresponding to non-idempotent colored types,
- and a qualitative, finitary one, which is decidable (colored extension of the Scott model of linear logic, with a parity fixpoint).

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!