The modal nature of colors in higher-order model-checking

Charles Grellois   Paul-André Melliès

PPS — Université Paris 7

July 18th, 2014

Model-checking higher-order programs

Usual approach in verification: model-checking. Interaction of a program and a property.

To model higher-order functional programs : recursion schemes (HORS), generating the tree of behaviours of the program they model.

Properties expressed in MSO or modal μ-calculus (equi-expressive over trees).

Their automata counterpart is given by alternating parity automata (APT).

Higher-order recursion schemes

Idea: it is a kind of grammar whose parameters may be functions and which generates trees.

Alternatively, it is a formalism equivalent to λY calculus with uninterpreted constants from a ranked alphabet.

A very simple functional program

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

With a recursion scheme we can model this program and produce its tree of behaviours.

Note that constants are not interpreted: in particular, a recursion scheme does not evaluate a `if`.

A very simple functional program

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

With a recursion scheme we can model this program and produce its tree of behaviours.

Note that constants are not interpreted: in particular, a recursion scheme does not evaluate a `if`.

A very simple functional program

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

formulated as a recursion scheme:

$$
\begin{array}{rcl}
\texttt{S} & = & \texttt{L Nil} \\
\texttt{L}\ x & = & \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ ))
\end{array}
$$

or, in λ-calculus style :

$$
\begin{array}{rcl}
\texttt{S} & = & \texttt{L Nil} \\
\texttt{L} & = & \lambda x.\,\texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ ))
\end{array}
$$

(this latter representation is a regular grammar !)

A very simple functional program

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

formulated as a recursion scheme:

$$\begin{array}{rcl} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L}\ x & = & \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ ) \end{array}$$

or, in λ-calculus style :

$$\begin{array}{rcl} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L} & = & \lambda x.\texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ ) \end{array}$$

(this latter representation is a regular grammar !)

A very simple functional program

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

formulated as a recursion scheme:

$$\begin{array}{rcl} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L } x & = & \texttt{if } x \; (\texttt{L} \; (\texttt{data } x \;)) \end{array}$$

or, in λ-calculus style :

$$\begin{array}{rcl} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L} & = & \lambda x.\, \texttt{if } x \; (\texttt{L} \; (\texttt{data } x \;)) \end{array}$$

(this latter representation is a regular grammar !)

Value tree of a recursion scheme

$$\begin{array}{rcl} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L}\ x & = & \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ ) \end{array}$$

generates:

S

Value tree of a recursion scheme

$$\begin{array}{rcl} \mathtt{S} & = & \mathtt{L\ Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )) \end{array}$$

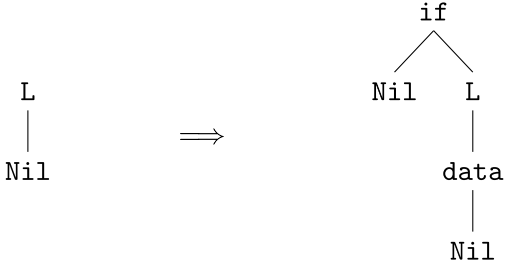
generates:

$$\mathtt{S} \quad \Longrightarrow \quad \begin{array}{c} \mathtt{L} \\ | \\ \mathtt{Nil} \end{array}$$

Value tree of a recursion scheme

$$
\begin{array}{rcl}
\mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\
\mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )
\end{array}
$$

generates:

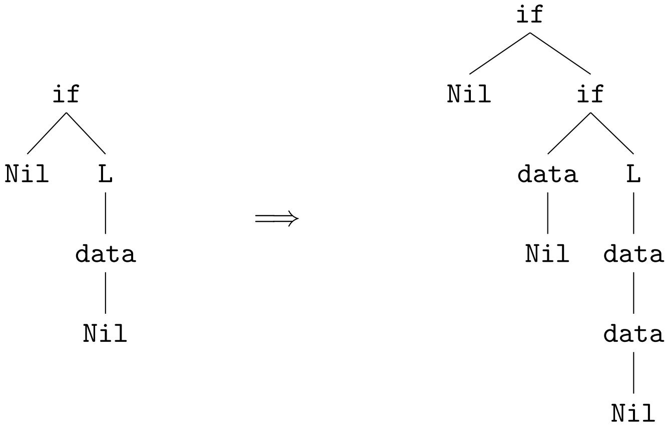


Notice that substitution and expansion occur in one same step.

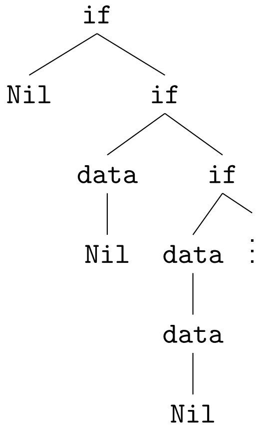
Value tree of a recursion scheme

$$\begin{array}{rcl} \mathtt{S} & = & \mathtt{L\ Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ) \end{array}$$

generates:

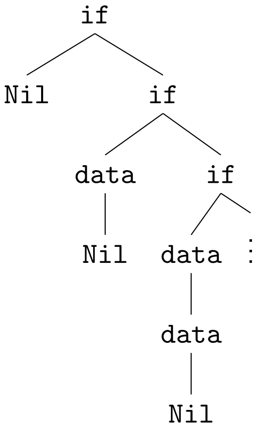


Value tree of a recursion scheme



Very simple program, yet it produces a tree which is not regular. . .

Value tree of a recursion scheme



Very simple program, yet it produces a tree which is not regular...

Alternating parity tree automata

Modal μ-calculus is an extension of boolean logic over a branching structure, with fixpoints and quantifications over the successors of the current position.

It allows to unravel some formula over the structure. This can be encoded into an alternating parity tree automata (APT).

Its states are the subformulas of the encoded formula.

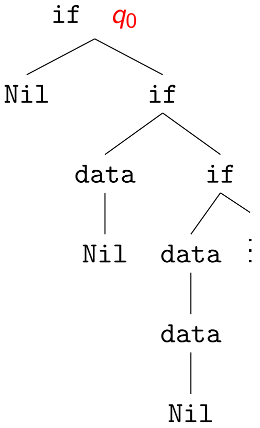
Alternating parity tree automata

APT are non-deterministic tree automata whose transitions may duplicate or drop a subtree.

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

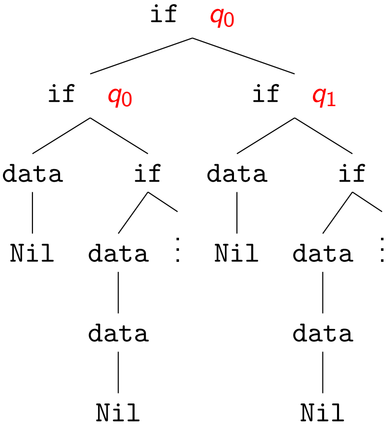
Alternating parity tree automata

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



Alternating parity tree automata

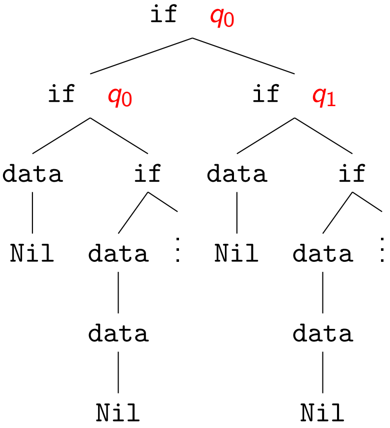
$\delta(q_0, \texttt{if}) \ = \ (2, q_0) \wedge (2, q_1).$



and so on. This gives the notion of run-tree.

Alternating parity tree automata

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



and so on. This gives the notion of run-tree.

Alternating parity tree automata

In modal μ-calculus, there are two fixpoints: informally, one allows finite looping, and the other infinite looping.

An APT run-tree can violate the finite looping condition. So we distinguish winning and loosing run-trees.

Each state is given a colour (an integer). A branch is winning iff the maximal colour among the ones occuring infinitely often over it is even. Else it is loosing.

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

A MSO formula holds at the root of a tree iff there exists a winning run-tree of its companion automaton.

Alternating parity tree automata

In modal μ-calculus, there are two fixpoints: informally, one allows finite looping, and the other infinite looping.

An APT run-tree can violate the finite looping condition. So we distinguish winning and loosing run-trees.

Each state is given a colour (an integer). A branch is winning iff the maximal colour among the ones occuring infinitely often over it is even. Else it is loosing.

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

A MSO formula holds at the root of a tree iff there exists a winning run-tree of its companion automaton.

Alternating parity tree automata

In modal μ-calculus, there are two fixpoints: informally, one allows finite looping, and the other infinite looping.

An APT run-tree can violate the finite looping condition. So we distinguish winning and loosing run-trees.

Each state is given a colour (an integer). A branch is winning iff the maximal colour among the ones occuring infinitely often over it is even. Else it is loosing.

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

A MSO formula holds at the root of a tree iff there exists a winning run-tree of its companion automaton.

Model-checking higher-order programs: Ong's decidability result

Theorem (Ong 2006)

Given a higher-order recursion scheme $\mathcal{G}$ and a MSO formula ϕ, one can decide whether ϕ holds at the root of the tree generated by $\mathcal{G}$.

Several proofs, among which the original one by Ong (2006) and a proof by Kobayashi and Ong (2009).

Model-checking higher-order programs: Ong's decidability result

Both proofs use the only canonical finitary representation of a scheme: its underlying term. That is, its unfolding *without* substitution.

APT runs over trees: that is, order-1 normal terms. Both proofs extend this behaviour to λ-terms, and thus to schemes.

Ong (2006): use game semantics to analyze the scheme's dynamics, and import this behaviour in an automaton running over λ-terms.

Model-checking higher-order programs: Ong's decidability result

Both proofs use the only canonical finitary representation of a scheme: its underlying term. That is, its unfolding *without* substitution.

APT runs over trees: that is, order-1 normal terms. Both proofs extend this behaviour to λ-terms, and thus to schemes.

Ong (2006): use game semantics to analyze the scheme's dynamics, and import this behaviour in an automaton running over λ-terms.

Alternating parity tree automata and intersection types

A key remark (Kobayashi 2009): if $\delta(q, a) \;=\; (1, q_0) \wedge (1, q_1) \wedge (2, q_2)$

then we may consider that a has a refined intersection type

$$(q_0 \wedge q_1) \Rightarrow q_2 \Rightarrow q$$

and what about colours ?

Consider $(\boxdot_{c_0}\; q_0 \wedge \boxdot_{c_1}\; q_1) \Rightarrow \boxdot_{c_2}\; q_2 \Rightarrow q$

(Kobayashi-Ong 2009, Grellois-Melliès 2014)

Extending this to a full type system refining simply-typed λ-calculus, we get a generalization of APT to λ-terms.

Alternating parity tree automata and intersection types

A key remark (Kobayashi 2009): if $\delta(q, a) \; = \; (1, q_0) \wedge (1, q_1) \wedge (2, q_2)$

then we may consider that a has a refined intersection type

$$(q_0 \wedge q_1) \Rightarrow q_2 \Rightarrow q$$

and what about colours ?

Consider $(\boxdot_{c_0} \; q_0 \wedge \boxdot_{c_1} \; q_1) \Rightarrow \boxdot_{c_2} \; q_2 \Rightarrow q$

(Kobayashi-Ong 2009, Grellois-Melliès 2014)

Extending this to a full type system refining simply-typed λ-calculus, we get a generalization of APT to λ-terms.

Alternating parity tree automata and intersection types

A key remark (Kobayashi 2009): if $\delta(q, a) \; = \; (1, q_0) \wedge (1, q_1) \wedge (2, q_2)$

then we may consider that a has a refined intersection type

$$(q_0 \wedge q_1) \Rightarrow q_2 \Rightarrow q$$

and what about colours ?

Consider $(\boxdot_{c_0} \; q_0 \wedge \boxdot_{c_1} \; q_1) \Rightarrow \boxdot_{c_2} \; q_2 \Rightarrow q$

(Kobayashi-Ong 2009, Grellois-Melliès 2014)

Extending this to a full type system refining simply-typed λ-calculus, we get a generalization of APT to λ-terms.

Alternating parity tree automata and intersection types

A key remark (Kobayashi 2009): if $\delta(q, a) \;=\; (1, q_0) \wedge (1, q_1) \wedge (2, q_2)$

then we may consider that a has a refined intersection type

$$(q_0 \wedge q_1) \Rightarrow q_2 \Rightarrow q$$

and what about colours ?

Consider $(\boxdot_{c_0}\ q_0 \wedge \boxdot_{c_1}\ q_1) \Rightarrow \boxdot_{c_2}\ q_2 \Rightarrow q$

(Kobayashi-Ong 2009, Grellois-Melliès 2014)

Extending this to a full type system refining simply-typed λ-calculus, we get a generalization of APT to λ-terms.

A type-system for verification: KO'09, without colours

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

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

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

λ
$$\frac{\Delta, x : \bigwedge_{i \in I} \ \theta_i :: \kappa \ \vdash \ t : \theta :: \kappa' \qquad I \subseteq J}{\Delta \ \vdash \ \lambda x . t : \left(\bigwedge_{j \in J} \ \theta_j\right) \to \theta :: \kappa \to \kappa'}$$

A type-system for verification: KO'09, without colours

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

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

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

A type-system for verification: KO'09, without colours

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

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

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

λ $$\frac{\Delta, x \,:\, \bigwedge_{i \in I} \;\theta_i \;::\; \kappa \;\vdash\; t \,:\, \theta \;::\; \kappa' \qquad\qquad I \subseteq J}{\Delta \;\vdash\; \lambda x\,.\,t \,:\, \left(\bigwedge_{j \in J} \;\theta_j\right) \to \theta :: \kappa \to \kappa'}$$

A type-system for verification: KO'09, without colours

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

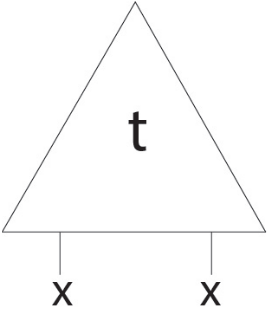
δ $$\frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\,\} \quad \text{satisfies} \quad \delta_A(q, a)}{\emptyset \vdash a \,:\, \bigwedge_{j=1}^{k_1}\ q_{1j} \rightarrow \ \ldots\ \rightarrow \bigwedge_{j=1}^{k_n}\ q_{nj} \rightarrow q \ ::\ \perp \rightarrow \cdots \rightarrow \perp}$$

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

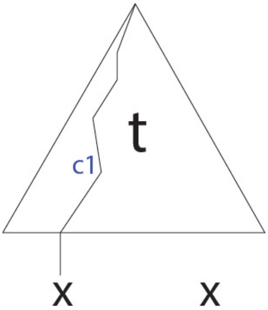
λ $$\frac{\Delta, x \,:\, \bigwedge_{i \in I}\ \theta_i \ ::\ \kappa \ \vdash\ t \,:\, \theta \ ::\ \kappa' \qquad\qquad I \subseteq J}{\Delta \ \vdash\ \lambda x \,.\, t \,:\, \left(\bigwedge_{j \in J}\ \theta_j\right) \rightarrow \theta :: \kappa \rightarrow \kappa'}$$

A type-system for verification: KO'09

In general, we need to add information in the typing about the colour of a term. Consider a term $\lambda x.\, t[x]$:

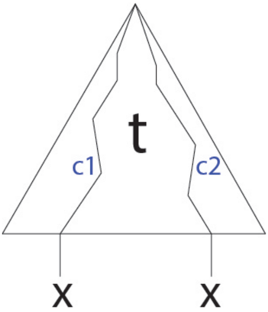


A type-system for verification: KO'09



In this picture, c_1 is the maximal colour seen from the root of the term to its first occurence of x.

A type-system for verification: KO'09



In this picture, c_1 is the maximal colour seen from the root of the term to its first occurence of x, and c_2 plays the same role for its second occurence.

Thus t will have some type $\boxdot_{c_1} \sigma_1 \wedge \boxdot_{c_2} \sigma_2 \to \tau$.

A type-system for verification: KO'09

$$\text{Axiom} \qquad \frac{}{x : \bigwedge_{\{i\}} \boxdot_{\Omega(\theta_i)} \theta_i :: \kappa \vdash x : \theta_i :: \kappa}$$

where $\Omega(\theta_i) = \Omega(\sigma_1 \to \cdots \to \sigma_n \to q) = \Omega(q)$

A type-system for verification: KO'09

Axiom $\quad \dfrac{}{x \,:\, \bigwedge_{\{i\}} \boxdot_{\Omega(\theta_i)}\, \theta_i \,::\, \kappa \;\vdash\; x \,:\, \theta_i \,::\, \kappa}$

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

where $m_{ij} \;=\; \max(\Omega(q_{ij}), \Omega(q))$

A type-system for verification: KO'09

Axiom $\dfrac{}{x \;:\; \bigwedge_{\{i\}} \boxdot_{\Omega(\theta_i)} \; \theta_i \;::\; \kappa \;\vdash\; x \;:\; \theta_i \;::\; \kappa}$

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

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

$\lambda \qquad \dfrac{\Delta\,,\, x \;:\; \bigwedge_{i \in I} \boxdot_{m_i} \; \theta_i \;::\; \kappa \;\vdash\; t \;:\; \theta \;::\; \kappa' \qquad\qquad I \subseteq J}{\Delta \;\vdash\; \lambda x\,.\,t \;:\; \left(\bigwedge_{j \in J} \boxdot_{m_j} \; \theta_j\right) \rightarrow \theta :: \kappa \rightarrow \kappa'}$

A type-system for verification: KO'09

Axiom $$\frac{}{x : \bigwedge_{\{i\}} \boxdot_{\Omega(\theta_i)} \, \theta_i :: \kappa \;\vdash\; x : \theta_i :: \kappa}$$

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

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

λ $$\frac{\Delta, x : \bigwedge_{i \in I} \boxdot_{m_i} \, \theta_i :: \kappa \;\vdash\; t : \theta :: \kappa' \qquad I \subseteq J}{\Delta \;\vdash\; \lambda x . t : \left(\bigwedge_{j \in J} \boxdot_{m_j} \, \theta_j \right) \to \theta :: \kappa \to \kappa'}$$

KO'09: companion game **Adamic**($\mathcal{G}, \mathcal{A}$)

This system allows to type finite terms.

Recall the scheme:

$$\begin{array}{lcl} \mathtt{S} & = & \mathtt{L\ Nil} \\ \mathtt{L} & = & \lambda x.\,\mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ) \end{array}$$

The type system will provide derivations of sequents of the shape:

$$L : \bigwedge_{\{1\}} \boxdot_c \tau :: \bot \to \bot \ \vdash\ \lambda x.\,\mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ) : \sigma :: \bot \to \bot$$

The point is to type the rewrite rules, under certain assumptions (= a context) on the non-terminals occuring in the considered term.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

This system allows to type finite terms.

Recall the scheme:

$$\begin{array}{lcl} \mathtt{S} & = & \mathtt{L\ Nil} \\ \mathtt{L} & = & \lambda x.\,\mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ) \end{array}$$

The type system will provide derivations of sequents of the shape:

$$L : \bigwedge_{\{1\}} \boxdot_c \tau :: \bot \to \bot \ \vdash\ \lambda x.\,\mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ) : \sigma :: \bot \to \bot$$

The point is to type the rewrite rules, under certain assumptions (= a context) on the non-terminals occuring in the considered term.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

This system allows to type finite terms.

Recall the scheme:

$$\begin{array}{lcl} \mathtt{S} & = & \mathtt{L\ Nil} \\ \mathtt{L} & = & \lambda x.\,\mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ) \end{array}$$

The type system will provide derivations of sequents of the shape:

$$L : \bigwedge_{\{1\}} \boxdot_c \tau :: \perp \to \perp \ \vdash \ \lambda x.\,\mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ) : \sigma :: \perp \to \perp$$

The point is to type the rewrite rules, under certain assumptions (= a context) on the non-terminals occuring in the considered term.

KO'09: companion game $\textbf{Adamic}(\mathcal{G}, \mathcal{A})$

Important point: if

$$\Gamma \ \vdash \ t \ : \ \tau \ : \ \kappa$$

holds in the type system, and F is a non-terminal occuring in t, then

$$F \ : \ \bigwedge_{i \in I} \ \boxdot_{c_i} \ \tau_i \ :: \ \kappa'$$

occurs in Γ, and for every occurence of F in t there exists $i \in I$ such that

- this occurence of F is introduced with type τ_i in the associated typing derivation,
- and the maximal colour seen from the introduction of this occurence to the root of the derivation is c_i.

Note that some occurences may correspond to the same index $i \in I$.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

Important point: if

$$\Gamma \ \vdash \ t \ : \ \tau \ : \ \kappa$$

holds in the type system, and F is a non-terminal occuring in t, then

$$F \ : \ \bigwedge_{i \in I} \ \boxdot_{c_i} \ \tau_i \ :: \ \kappa'$$

occurs in Γ, and for every occurence of F in t there exists $i \in I$ such that

- this occurence of F is introduced with type τ_i in the associated typing derivation,
- and the maximal colour seen from the introduction of this occurence to the root of the derivation is c_i.

Note that some occurences may correspond to the same index $i \in I$.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

Recursion is obtained with a parity game **Adamic**$(\mathcal{G}, \mathcal{A})$.
Adam plays typed occurences of non-terminals, and Eve plays typing contexts.

Adam starts by picking $S : q_0 :: \perp$.

Then Eve provides a context Γ such that $\Gamma \vdash \mathcal{R}(S) : q_0 :: \perp$.

Then Adam picks a non-terminal F occuring in $\mathcal{R}(S)$. Recall it has type

$$F : \bigwedge_{i \in I} \boxdot_{c_i} \tau_i :: \kappa'$$

so that Adam also picks an integer $i \in I$.

Now Eve has to find Γ' such that $\Gamma' \vdash \mathcal{R}(F) : \tau_i :: \kappa'$

Adam picks $G \in \Gamma'$ and an integer j, and so on.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

Recursion is obtained with a parity game **Adamic**$(\mathcal{G}, \mathcal{A})$.
Adam plays typed occurences of non-terminals, and Eve plays typing contexts.

Adam starts by picking $S \,:\, q_0 :: \perp$.

Then Eve provides a context Γ such that $\Gamma \vdash \mathcal{R}(S) : q_0 :: \perp$.

Then Adam picks a non-terminal F occuring in $\mathcal{R}(S)$. Recall it has type

$$F : \bigwedge_{i \in I} \boxdot_{c_i} \tau_i :: \kappa'$$

so that Adam also picks an integer $i \in I$.

Now Eve has to find Γ' such that $\Gamma' \vdash \mathcal{R}(F) : \tau_i :: \kappa'$

Adam picks $G \in \Gamma'$ and an integer j, and so on.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

Recursion is obtained with a parity game **Adamic**$(\mathcal{G}, \mathcal{A})$.
Adam plays typed occurences of non-terminals, and Eve plays typing contexts.

Adam starts by picking $S : q_0 :: \perp$.

Then Eve provides a context Γ such that $\Gamma \vdash \mathcal{R}(S) : q_0 :: \perp$.

Then Adam picks a non-terminal F occuring in $\mathcal{R}(S)$. Recall it has type

$$F : \bigwedge_{i \in I} \boxed{}_{c_i} \tau_i :: \kappa'$$

so that Adam also picks an integer $i \in I$.

Now Eve has to find Γ' such that $\Gamma' \vdash \mathcal{R}(F) : \tau_i :: \kappa'$

Adam picks $G \in \Gamma'$ and an integer j, and so on.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

Recursion is obtained with a parity game **Adamic**$(\mathcal{G}, \mathcal{A})$.
Adam plays typed occurences of non-terminals, and Eve plays typing contexts.

Adam starts by picking $S \,:\, q_0 :: \perp$.

Then Eve provides a context Γ such that $\Gamma \vdash \mathcal{R}(S) \,:\, q_0 :: \perp$.

Then Adam picks a non-terminal F occuring in $\mathcal{R}(S)$. Recall it has type

$$F \,:\, \bigwedge_{i \in I} \boxdot_{c_i} \tau_i :: \kappa'$$

so that Adam also picks an integer $i \in I$.

Now Eve has to find Γ' such that $\Gamma' \vdash \mathcal{R}(F) \,:\, \tau_i :: \kappa'$

Adam picks $G \in \Gamma'$ and an integer j, and so on.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

Recursion is obtained with a parity game **Adamic**$(\mathcal{G}, \mathcal{A})$.
Adam plays typed occurences of non-terminals, and Eve plays typing contexts.

Adam starts by picking $S \,:\, q_0 :: \perp$.

Then Eve provides a context Γ such that $\Gamma \vdash \mathcal{R}(S) : q_0 :: \perp$.

Then Adam picks a non-terminal F occuring in $\mathcal{R}(S)$. Recall it has type

$$F \,:\, \bigwedge_{i \in I} \boxdot_{c_i} \tau_i :: \kappa'$$

so that Adam also picks an integer $i \in I$.

Now Eve has to find Γ' such that $\Gamma' \vdash \mathcal{R}(F) : \tau_i :: \kappa'$

Adam picks $G \in \Gamma'$ and an integer j, and so on.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

Recursion is obtained with a parity game **Adamic**$(\mathcal{G}, \mathcal{A})$.
Adam plays typed occurences of non-terminals, and Eve plays typing contexts.

Adam starts by picking $S \,:\, q_0 \,::\, \perp$.

Then Eve provides a context Γ such that $\Gamma \,\vdash\, \mathcal{R}(S) \,:\, q_0 \,::\, \perp$.

Then Adam picks a non-terminal F occuring in $\mathcal{R}(S)$. Recall it has type

$$F \,:\, \bigwedge_{i \in I} \boxdot_{c_i} \, \tau_i \,::\, \kappa'$$

so that Adam also picks an integer $i \in I$.

Now Eve has to find Γ' such that $\Gamma' \,\vdash\, \mathcal{R}(F) \,:\, \tau_i \,::\, \kappa'$

Adam picks $G \in \Gamma'$ and an integer j, and so on.

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

This defines a parity game: whenever Adam picks an integer i, the corresponding colour c_i is played.

Theorem (Kobayashi-Ong 2009)

Eve has a winning strategy starting from $S : q_0 :: \perp$ for the parity game **Adamic**$(\mathcal{G}, \mathcal{A})$ *iff the automaton $\mathcal{A}$ has a winning run-tree over the value tree of the scheme $\mathcal{G}$.*

KO'09: companion game **Adamic**$(\mathcal{G}, \mathcal{A})$

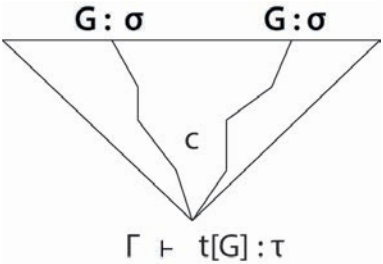
This defines a parity game: whenever Adam picks an integer i, the corresponding colour c_i is played.

Theorem (Kobayashi-Ong 2009)

Eve has a winning strategy starting from $S : q_0 :: \perp$ for the parity game **Adamic**$(\mathcal{G}, \mathcal{A})$ *iff the automaton $\mathcal{A}$ has a winning run-tree over the value tree of the scheme $\mathcal{G}$.*

Typing vs. **Adamic**($\mathcal{G}, \mathcal{A}$): a difference w.r.t. unfolding

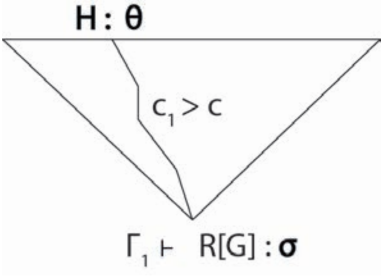
Suppose Adam plays F, which rewrites to a term t where a non-terminal G occurs at least twice. Suppose that Eve answers with Γ built from the following derivation tree:



These two occurences of G correspond in Γ to the same index in the intersection type for G.

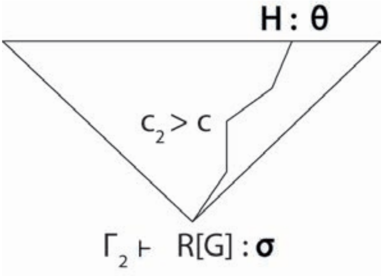
Typing vs. **Adamic**$(\mathcal{G}, \mathcal{A})$: a difference w.r.t. unfolding

If Adam plays it next, Eve answers with a new context Γ_1 such that $\Gamma_1 \vdash \mathcal{R}(G) : \sigma$. Suppose it comes from



Typing vs. **Adamic**($\mathcal{G}, \mathcal{A}$): a difference w.r.t. unfolding

Suppose Eve could have played Γ_2 as well:



Typing vs. **Adamic**$(\mathcal{G}, \mathcal{A})$: a difference w.r.t. unfolding

In this game, Adam played the colour c and asked Eve a typing context for $\mathcal{R}(G)$.

- If Eve plays Γ_1, then Adam plays H with colour c_1. Overall, he played $c_1 = \max(c, c_1)$ during these two turns.
- If Eve plays Γ_2, then Adam plays H with colour c_2. Overall, he played $c_2 = \max(c, c_2)$ during these two turns.

Eve could choose to remove c_1 or c_2 from Adam's possibilities.

Typing vs. **Adamic**$(\mathcal{G}, \mathcal{A})$: a difference w.r.t. unfolding

In this game, Adam played the colour c and asked Eve a typing context for $\mathcal{R}(G)$.

- If Eve plays Γ_1, then Adam plays H with colour c_1. Overall, he played $c_1 = \max(c, c_1)$ during these two turns.
- If Eve plays Γ_2, then Adam plays H with colour c_2. Overall, he played $c_2 = \max(c, c_2)$ during these two turns.

Eve could choose to remove c_1 or c_2 from Adam's possibilities.

Typing vs. **Adamic**$(\mathcal{G}, \mathcal{A})$: a difference w.r.t. unfolding

In this game, Adam played the colour c and asked Eve a typing context for $\mathcal{R}(G)$.

- If Eve plays Γ_1, then Adam plays H with colour c_1. Overall, he played $c_1 = \max(c, c_1)$ during these two turns.
- If Eve plays Γ_2, then Adam plays H with colour c_2. Overall, he played $c_2 = \max(c, c_2)$ during these two turns.

Eve could choose to remove c_1 or c_2 from Adam's possibilities.

Typing vs. **Adamic**$(\mathcal{G}, \mathcal{A})$: a difference w.r.t. unfolding

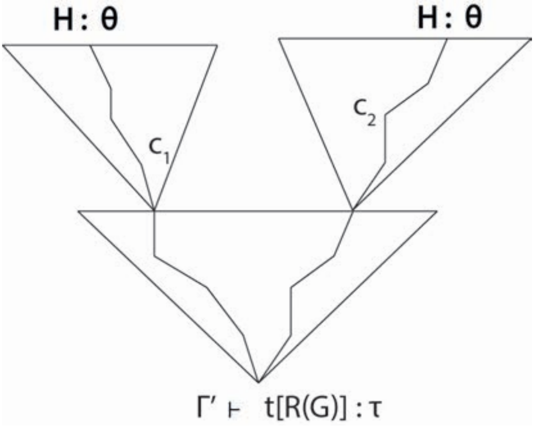
In this game, Adam played the colour c and asked Eve a typing context for $\mathcal{R}(G)$.

- If Eve plays Γ_1, then Adam plays H with colour c_1. Overall, he played $c_1 = \max(c, c_1)$ during these two turns.
- If Eve plays Γ_2, then Adam plays H with colour c_2. Overall, he played $c_2 = \max(c, c_2)$ during these two turns.

Eve could choose to remove c_1 or c_2 from Adam's possibilities.

Typing vs. **Adamic**($\mathcal{G}, \mathcal{A}$): a difference w.r.t. unfolding

But suppose we expanded syntactically $t[G]$ to $t[\mathcal{R}(G)]$. Then Eve's typing may have been



In which case Adam can choose to play either c_1 or c_2.

Typing vs. **Adamic**$(\mathcal{G}, \mathcal{A})$: a difference w.r.t. unfolding

This difference comes from the fact that the game **Adamic**$(\mathcal{G}, \mathcal{A})$ does not reflect typing derivations: Adam does not explore branches of the derivation tree. He cannot distinguish between occurences of a non-terminal occuring in a context with the same typing.

This game **Adamic**$(\mathcal{G}, \mathcal{A})$ actually underlies a property of uniformity of typing, to which we will get back later.

Reflecting typings: the game **Edenic**$(\mathcal{G}, \mathcal{A})$

In order to reflect typing derivations, it suffices to modify slightly the game **Adamic**$(\mathcal{G}, \mathcal{A})$:

- Eve now provides not only typing contexts, but also the proof-tree she built.
- Adam plays any occurence of a non-terminal occuring as a leaf of the proof-tree given by Eve.

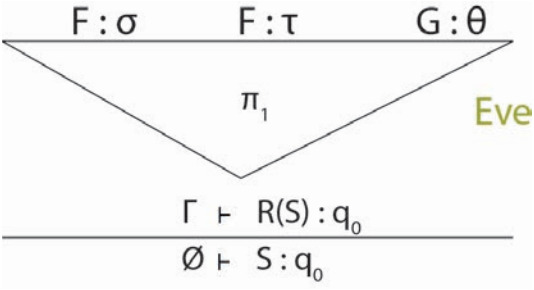
Define the resulting game as **Edenic**$(\mathcal{G}, \mathcal{A})$.

$\mathbf{Edenic}(\mathcal{G}, \mathcal{A})$ and type systems

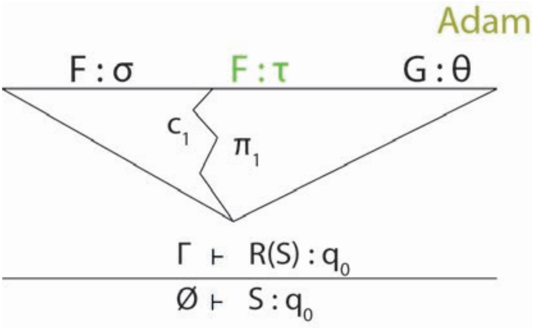
In general, a game in Edenic is as follows:

$$\emptyset \vdash S : q_0 \qquad \text{Adam}$$

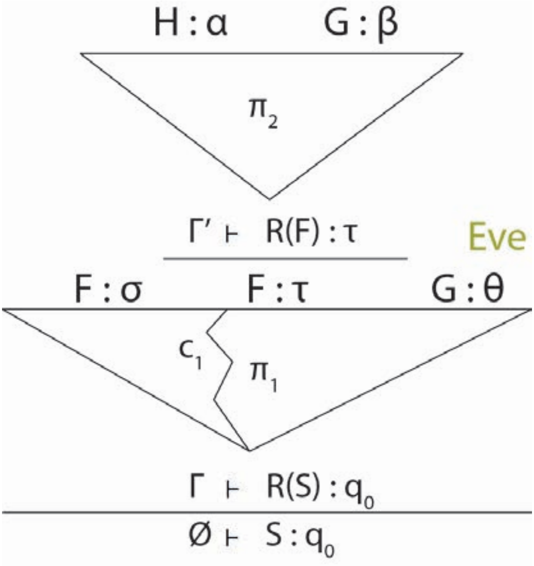
Edenic($\mathcal{G}, \mathcal{A}$) and type systems



$\mathbf{Edenic}(\mathcal{G}, \mathcal{A})$ and type systems



Edenic($\mathcal{G}, \mathcal{A}$) and type systems



Edenic$(\mathcal{G}, \mathcal{A})$ and type systems

In this game, a strategy for Eve corresponds to a typing derivation of the infinite term underlying the recursion scheme.

Adam plays branches of this derivation tree.

The parity condition is then understood as a condition over branches of the derivation tree.

Edenic$(\mathcal{G}, \mathcal{A})$ and type systems

In this game, a strategy for Eve corresponds to a typing derivation of the infinite term underlying the recursion scheme.

Adam plays branches of this derivation tree.

The parity condition is then understood as a condition over branches of the derivation tree.

A type-system for verification: KO_{fix}

Consider the type system of Kobayashi and Ong, enriched with the *fix* rule:

$$\textit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{F \,:\, \boxdot_{\Omega(\theta)}\, \theta \,::\, \kappa \;\vdash\; F \,:\, \theta \,::\, \kappa}$$

and allowing infinite-depth derivations.

To reflect the parity condition in $\mathbf{Edenic}(\mathcal{G}, \mathcal{A})$, we colour the instances of the *fix* rule (as in the game).

Then the usual parity condition over trees discriminates winning from loosing derivation trees.

We call this type system KO_{fix}.

A type-system for verification: KO_{fix}

Consider the type system of Kobayashi and Ong, enriched with the *fix* rule:

$$\textit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{F \,:\, \boxdot_{\Omega(\theta)}\, \theta \,::\, \kappa \;\vdash\; F \,:\, \theta \,::\, \kappa}$$

and allowing infinite-depth derivations.

To reflect the parity condition in **Edenic**$(\mathcal{G}, \mathcal{A})$, we colour the instances of the *fix* rule (as in the game).

Then the usual parity condition over trees discriminates winning from loosing derivation trees.

We call this type system KO_{fix}.

A type-system for verification: KO_{fix}

Theorem

There is a winning run-tree over the tree produced by a scheme if and only if there exists a winning derivation tree of the sequent

$$S : \boxdot_{\Omega(q_0)} q_0 :: \perp \vdash S : q_0 :: \perp$$

in the type system KO_{fix}.

A connection to game semantics

In game semantics, two players interact on an arena, which models a type.

The two players can be thought of as a program and a counter-program.

A key point of our approach is to see the automaton as a counter-program interacting with a λ-term.

A connection to game semantics

In game semantics, two players interact on an arena, which models a type.

The two players can be thought of as a program and a counter-program.

A key point of our approach is to see the automaton as a counter-program interacting with a λ-term.

A connection to game semantics

There is a correspondence between

- Innocent strategies
- and derivation trees of the simply-typed λ-calculus

Note that innocent strategies correspond to β-normal η-long terms.

A connection to game semantics

The idea is that the counter-program explores branches of the term, while the program reveals the term on request.

The game starts on its initial node, indexed with the type A of the term played by the program:

$$\Omega_A$$

A connection to game semantics

The counter-program plays first, declaring variables according to the type A, and asking the head variable of the term:

$$\Omega_A \longrightarrow \lambda x. \lambda y. \mho$$

if $A = A_1 \Rightarrow A_2 \Rightarrow \bot$.

A connection to game semantics

Then Eve answers with the head symbol, and the types of its arguments.

$$\lambda x.\,\lambda y.\,\mho \;\longrightarrow\; \lambda x.\,\lambda y.\,y\;\Omega_{A_3}\;\Omega_{A_4}$$

if $A_2 \;=\; A_3 \Rightarrow A_4 \Rightarrow \perp$.

Then the counter-program chooses which of y's arguments to explore, and so on...

From the whole set of interactions of the program and the counter-program, one can extract a typing derivation of the term.

A connection to game semantics

Then Eve answers with the head symbol, and the types of its arguments.

$$\lambda x.\, \lambda y.\, \mho \;\longrightarrow\; \lambda x.\, \lambda y.\, y\ \Omega_{A_3}\ \Omega_{A_4}$$

if $A_2 \;=\; A_3 \Rightarrow A_4 \Rightarrow \perp$.

Then the counter-program chooses which of y's arguments to explore, and so on...

From the whole set of interactions of the program and the counter-program, one can extract a typing derivation of the term.

A connection to game semantics

Then Eve answers with the head symbol, and the types of its arguments.

$$\lambda x.\, \lambda y.\, \mho \;\longrightarrow\; \lambda x.\, \lambda y.\, y\ \Omega_{A_3}\ \Omega_{A_4}$$

if $A_2 \;=\; A_3 \Rightarrow A_4 \Rightarrow \perp$.

Then the counter-program chooses which of y's arguments to explore, and so on. . .

From the whole set of interactions of the program and the counter-program, one can extract a typing derivation of the term.

A connection to game semantics

Edenic$(\mathcal{G}, \mathcal{A})$ and its type-theoretic counterpart KO_{fix} are close in spirit to this connection.

However, there are some obstructions:

- We need terms in η-long β-normal form.
- How do we understand the *fix* rule ? So far it is not a counter-program question, nor a program answer. . .
- We need a game semantics for refined types.

A connection to game semantics

Edenic$(\mathcal{G}, \mathcal{A})$ and its type-theoretic counterpart KO_{fix} are close in spirit to this connection.

However, there are some obstructions:

- We need terms in η-long β-normal form.
- How do we understand the *fix* rule ? So far it is not a counter-program question, nor a program answer. . .
- We need a game semantics for refined types.

A connection to game semantics

Edenic$(\mathcal{G}, \mathcal{A})$ and its type-theoretic counterpart KO_{fix} are close in spirit to this connection.

However, there are some obstructions:

- We need terms in η-long β-normal form.
- How do we understand the *fix* rule ? So far it is not a counter-program question, nor a program answer. . .
- We need a game semantics for refined types.

A connection to game semantics

Edenic$(\mathcal{G}, \mathcal{A})$ and its type-theoretic counterpart KO_{fix} are close in spirit to this connection.

However, there are some obstructions:

- We need terms in η-long β-normal form.
- How do we understand the *fix* rule ? So far it is not a counter-program question, nor a program answer. . .
- We need a game semantics for refined types.

Frozen terms

The two first obstructions can be adressed at the same time.

For β-normal form, there is a subtle trick to freeze the term: for every non-terminal F of simple type κ, we introduce a new variable f of simple type $\kappa \to \kappa$.

Then we modify the rewriting rules of the considered scheme:

- Every instance of a non-terminal F is replaced with $f\ F$
- Then every term occuring in the right-hand side of a rewriting rule is fully η-expanded

The infinite unfolding of this scheme $\mathcal{G}$ gives an infinite term in $\beta\eta$-normal form, denoted $[\mathcal{G}]$.

Computation may be "restarted" by substituting every newly introduced variable with identity.

Frozen terms

The two first obstructions can be adressed at the same time.

For β-normal form, there is a subtle trick to freeze the term: for every non-terminal F of simple type κ, we introduce a new variable f of simple type $\kappa \to \kappa$.

Then we modify the rewriting rules of the considered scheme:

- Every instance of a non-terminal F is replaced with $f\ F$
- Then every term occuring in the right-hand side of a rewriting rule is fully η-expanded

The infinite unfolding of this scheme $\mathcal{G}$ gives an infinite term in $\beta\eta$-normal form, denoted $[\mathcal{G}]$.

Computation may be "restarted" by substituting every newly introduced variable with identity.

Frozen terms

The two first obstructions can be adressed at the same time.

For β-normal form, there is a subtle trick to freeze the term: for every non-terminal F of simple type κ, we introduce a new variable f of simple type $\kappa \rightarrow \kappa$.

Then we modify the rewriting rules of the considered scheme:

- Every instance of a non-terminal F is replaced with $f\ F$
- Then every term occuring in the right-hand side of a rewriting rule is fully η-expanded

The infinite unfolding of this scheme $\mathcal{G}$ gives an infinite term in $\beta\eta$-normal form, denoted $[\mathcal{G}]$.

Computation may be "restarted" by substituting every newly introduced variable with identity.

Frozen terms

The two first obstructions can be adressed at the same time.

For β-normal form, there is a subtle trick to freeze the term: for every non-terminal F of simple type κ, we introduce a new variable f of simple type $\kappa \to \kappa$.

Then we modify the rewriting rules of the considered scheme:

- Every instance of a non-terminal F is replaced with $f\ F$
- Then every term occuring in the right-hand side of a rewriting rule is fully η-expanded

The infinite unfolding of this scheme $\mathcal{G}$ gives an infinite term in $\beta\eta$-normal form, denoted $[\mathcal{G}]$.

Computation may be "restarted" by substituting every newly introduced variable with identity.

Preservation of typability

These transformations preserve typing, in particular every instance of the *fix* rule

$$fix \quad \frac{\Gamma \ \vdash \ \mathcal{R}(F) \ : \ \theta \ :: \ \kappa}{F \ : \ \boxdot_{\Omega(\theta)} \theta \ :: \ \kappa \ \vdash \ F \ : \ \theta \ :: \ \kappa} \quad \mathrm{dom}(\Gamma) \subseteq \mathcal{N}$$

is replaced with

$$\frac{Ax \dfrac{}{f \ \vdash \ f \ : \ \boxdot_{\Omega(\theta)} \theta \ \rightarrow \ \theta} \qquad \dfrac{\Gamma \ \vdash \ \mathcal{R}(F) \ : \ \theta}{F \ : \ \boxdot_{\Omega(\theta)} \ \theta \ \vdash \ F \ : \ \theta} fix}{f \ : \ \boxdot_{\Omega(\theta)}(\boxdot_{\Omega(\theta)} \theta \ \rightarrow \ \theta), \ F \ : \ \boxdot_{\Omega(\theta)} \ \theta \ \vdash \ f(F) \ : \ \theta} App$$

to build a derivation tree of the frozen term from the original one.

Identity typing

A connection to game semantics

Now the situation appears as a generalization of the correspondence for finite η-long β-normal terms.

The *fix* rule corresponds to a counter-program request: where the *fix* rule for a non-terminal F was played by Adam, and Eve had to answer with a typing for its expansion $\mathcal{F}(F)$, we obtain a situation where the counter-program requires the exploration of f and the typing of its arguments.

There only remains one difference between $\textbf{Edenic}(\mathcal{G}, \mathcal{A})$ and the game semantics of KO_{fix}: the latter plays "smaller steps", every variable can be played, and the answer is only about its immediate successors.

In Edenic, Adam plays non-terminals only, and Eve answers with a derivation tree.

A connection to game semantics

Now the situation appears as a generalization of the correspondence for finite η-long β-normal terms.

The *fix* rule corresponds to a counter-program request: where the *fix* rule for a non-terminal F was played by Adam, and Eve had to answer with a typing for its expansion $\mathcal{F}(F)$, we obtain a situation where the counter-program requires the exploration of f and the typing of its arguments.

There only remains one difference between $\mathbf{Edenic}(\mathcal{G}, \mathcal{A})$ and the game semantics of KO_{fix}: the latter plays "smaller steps", every variable can be played, and the answer is only about its immediate successors.

In Edenic, Adam plays non-terminals only, and Eve answers with a derivation tree.

A connection to game semantics

Now the situation appears as a generalization of the correspondence for finite η-long β-normal terms.

The *fix* rule corresponds to a counter-program request: where the *fix* rule for a non-terminal F was played by Adam, and Eve had to answer with a typing for its expansion $\mathcal{F}(F)$, we obtain a situation where the counter-program requires the exploration of f and the typing of its arguments.

There only remains one difference between **Edenic**$(\mathcal{G}, \mathcal{A})$ and the game semantics of KO_{fix}: the latter plays "smaller steps", every variable can be played, and the answer is only about its immediate successors.

In Edenic, Adam plays non-terminals only, and Eve answers with a derivation tree.

An unexpected discovery

Assume that colours used in $\mathcal{A}$ are at least worth 2. Then, for a scheme in frozen form,

$$S : \boxdot_{\Omega(q_0)} q_0 :: \perp \vdash S : q_0 :: \perp$$

admits a winning derivation in KO_{fix} iff it admits a winning derivation in a variant of this system where Ax, δ and *fix* are replaced with

Axiom-new
$$\frac{}{x : \bigwedge_{\{i\}} \boxdot_1 \theta_i :: \kappa \;\vdash\; x : \theta_i :: \kappa}$$

δ-new ... for $a \in \Sigma$ and $m_{ij} = \Omega(q_{ij})$

fix
$$\frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \boxdot_1 \theta :: \kappa \vdash F : \theta :: \kappa}$$

An unexpected discovery

The idea is that 1 is a "neutral" colour — which cannot be used infinitely often in a row, this excluding problems like $F = \lambda x.\, Fx$.

In fact, variables and non-terminals are neutral from the point of view of colouration.

Symbols from the tree signature are the only creators of colouring information, other symbols only convey it.

In $\delta - new$, the typing now corresponds to an extension of the game semantics moves introduced earlier.

Eve types (and colours) only the *arguments* of the current symbol; its return type/colour having been set previously, when it was applied.

An unexpected discovery

The idea is that 1 is a "neutral" colour — which cannot be used infinitely often in a row, this excluding problems like $F = \lambda x. Fx$.

In fact, variables and non-terminals are neutral from the point of view of colouration.

Symbols from the tree signature are the only creators of colouring information, other symbols only convey it.

In $\delta - new$, the typing now corresponds to an extension of the game semantics moves introduced earlier.

Eve types (and colours) only the *arguments* of the current symbol; its return type/colour having been set previously, when it was applied.

An unexpected discovery

The idea is that 1 is a "neutral" colour — which cannot be used infinitely often in a row, this excluding problems like $F = \lambda x. Fx$.

In fact, variables and non-terminals are neutral from the point of view of colouration.

Symbols from the tree signature are the only creators of colouring information, other symbols only convey it.

In $\delta - \mathit{new}$, the typing now corresponds to an extension of the game semantics moves introduced earlier.

Eve types (and colours) only the *arguments* of the current symbol; its return type/colour having been set previously, when it was applied.

Semantic consequences

From this follows that

The colouring modality is a parameterized comonoad.

It has the same canonical properties as the exponential of linear logic.

Semantic consequences

From this follows that

The colouring modality is a parameterized comonoad.

It has the same canonical properties as the exponential of linear logic.

Semantic consequences

We can extend linear logic, or Melliès' tensorial logic with this colouring modality, in such a way that the following sequents are canonically provable in the resulting logic:

$$\Box_1 \, A \vdash \; A$$

$$\Box_{\max(m_1,m_2)} \, A \vdash \; \Box_{m_1} \, \Box_{m_2} \, A$$

$$\Box_m \, (A \otimes B) \; \vdash \; (\Box_m \, A) \otimes (\Box_m \, B)$$

and use the resulting logic to type terms.

Semantic consequences

The introduction rules are

$$\text{Left } \Box \qquad \frac{\Gamma,\, x \,:\, \tau \,:\, \kappa' \vdash M : \sigma :: \kappa}{\Gamma,\, x \,:\, \Box_1\, \tau \,:\, \kappa' \vdash M : \sigma :: \kappa}$$

$$\text{Right } \Box_m \qquad \frac{\Gamma \vdash M : \sigma :: \kappa}{\Box_m\, \Gamma \vdash M : \Box_m\, \sigma :: \kappa}$$

Semantic consequences

In coloured tensorial calculus, derivations of infinite terms are allowed. There is a notion of winning derivation: the rule Right $\Box_m$ is given the colour m, the other rules have no colour.

Then the usual parity condition over trees discriminates winning from loosing trees.

Semantic consequences

Theorem

Every infinite $KO_{fix}(\mathcal{G}, \mathcal{A})$ derivation tree π proving the sequent

$$S \,:\, \boxdot_{\Omega(q_0)} q_0 \,::\, \perp \quad \vdash \quad S \,:\, q_0 \,::\, \perp$$

can be translated into an infinite derivation tree $[\pi]$ of tensorial logic with colours with conclusion

$$\Gamma \vdash [\mathcal{G}] \,:\, \perp_{q_0} \,::\, \perp$$

where Γ is a refinement of the context of constructors of the signature Σ.

Moreover, π is winning in $KO_{fix}(\mathcal{G}, \mathcal{A})$ iff its translation $[\pi]$ is winning in the coloured tensorial calculus derivation tree.

Note that the context Γ occuring in the coloured tensorial calculus corresponds to the Girard-Reynolds interpretation of trees.

Semantic consequences

Theorem

Every infinite $KO_{fix}(\mathcal{G}, \mathcal{A})$ derivation tree π proving the sequent

$$S \,:\, \boxdot_{\Omega(q_0)} q_0 \,::\, \perp \quad \vdash \quad S \,:\, q_0 \,::\, \perp$$

can be translated into an infinite derivation tree $[\pi]$ of tensorial logic with colours with conclusion

$$\Gamma \vdash [\mathcal{G}] \,:\, \perp_{q_0} \,::\, \perp$$

where Γ is a refinement of the context of constructors of the signature Σ.

Moreover, π is winning in $KO_{fix}(\mathcal{G}, \mathcal{A})$ iff its translation $[\pi]$ is winning in the coloured tensorial calculus derivation tree.

Note that the context Γ occuring in the coloured tensorial calculus corresponds to the Girard-Reynolds interpretation of trees.

Perspectives

- Melliès tensorial logic canonically reflects his asynchronous game semantics: from this extension with a colouring modality follows a coloured game semantics.
- Linear logic provides several models for interpreting calculus. The modal nature of $\Box$ allows to extend them to coloured models where one can interpret the interaction of a term/a scheme with an APT.
- We can give an indexed notion of the coloured linear calculus, reflecting the coloured infinitary relational semantics of this interaction.
- We can also see the decidability result of Kobayashi and Ong as the fact that every typable sequent in KO_{fix} admits a uniform typing -derivation. This relies on the existence of memoryless strategies for parity games, imported to the case of derivation trees.

Perspectives

- Melliès tensorial logic canonically reflects his asynchronous game semantics: from this extension with a colouring modality follows a coloured game semantics.
- Linear logic provides several models for interpreting calculus. The modal nature of $\Box$ allows to extend them to coloured models where one can interpret the interaction of a term/a scheme with an APT.
- We can give an indexed notion of the coloured linear calculus, reflecting the coloured infinitary relational semantics of this interaction.
- We can also see the decidability result of Kobayashi and Ong as the fact that every typable sequent in KO_{fix} admits a uniform typing -derivation. This relies on the existence of memoryless strategies for parity games, imported to the case of derivation trees.

Perspectives

- Melliès tensorial logic canonically reflects his asynchronous game semantics: from this extension with a colouring modality follows a coloured game semantics.
- Linear logic provides several models for interpreting calculus. The modal nature of $\Box$ allows to extend them to coloured models where one can interpret the interaction of a term/a scheme with an APT.
- We can give an indexed notion of the coloured linear calculus, reflecting the coloured infinitary relational semantics of this interaction.
- We can also see the decidability result of Kobayashi and Ong as the fact that every typable sequent in KO_{fix} admits a uniform typing -derivation. This relies on the existence of memoryless strategies for parity games, imported to the case of derivation trees.

Perspectives

- Melliès tensorial logic canonically reflects his asynchronous game semantics: from this extension with a colouring modality follows a coloured game semantics.
- Linear logic provides several models for interpreting calculus. The modal nature of $\Box$ allows to extend them to coloured models where one can interpret the interaction of a term/a scheme with an APT.
- We can give an indexed notion of the coloured linear calculus, reflecting the coloured infinitary relational semantics of this interaction.
- We can also see the decidability result of Kobayashi and Ong as the fact that every typable sequent in KO_{fix} admits a uniform typing -derivation. This relies on the existence of memoryless strategies for parity games, imported to the case of derivation trees.