Categorical semantics of linear logic and higher-order model-checking

Charles Grellois (joint work with Paul-André Melliès)

PPS & LIAFA — Université Paris 7

April 9th, 2015

Model-checking higher-order programs

Following Igor's talk this morning, we focus on the

model-checking problem

of trees generated by

higher-order recursion schemes (HORS)

using MSO's automata-theoretic counterpart:

alternating parity automata (APT)

In this talk, we discuss how linear logic and its categorical semantics bring to light key elements of this problem, and notably lead to yet another decidability proof.

Model-checking higher-order programs

Following Igor's talk this morning, we focus on the

model-checking problem

of trees generated by

higher-order recursion schemes (HORS)

using MSO's automata-theoretic counterpart:

alternating parity automata (APT)

In this talk, we discuss how linear logic and its categorical semantics bring to light key elements of this problem, and notably lead to yet another decidability proof.

Model-checking higher-order programs

This model-checking problem is decidable:

- Ong 2006 (game semantics)
- Hague-Murawski-Ong-Serre 2008 (game semantics + collapsible higher-order pushdown automata)
- Kobayashi-Ong 2009 (intersection types)
- Salvati-Walukiewicz 2011 (interpretation with Krivine machines)
- Carayol-Serre 2012 (collapsible higher-order pushdown automata)
- Tsukada-Ong 2014 (game semantics)
- Salvati-Walukiewicz 2015 (interpretation in finite models)
- Grellois-Melliès 2015

Our aim was to deepen the semantic understanding we have of this result, using existing relations between alternating automata, intersection types, (linear) logic and its models – game-based as well as denotational.

Prologue: finite automata theory

A very naive model-checking problem

We start by introducing some key ideas of the approach in a simpler framework.

Consider the most naive possible model-checking problem where:

- Actions of the program are modelled by a finite word
- The property to check corresponds to a finite automaton

A very naive model-checking problem

A word of actions :

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

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

Corresponds to an automaton with two states: $Q = \{q_0, q_1\}$.
q_0 is both initial and final.

A very naive model-checking problem

A word of actions :

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

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

Corresponds to an automaton with two states: $Q = \{q_0, q_1\}$.
q_0 is both initial and final.

A very naive model-checking problem

A word of actions :

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

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

Corresponds to an automaton with two states: $Q = \{q_0, q_1\}$.
q_0 is both initial and final.

A type-theoretic intuition

The transition function may be seen as a typing of the letters of the word, seen as function symbols.

For example,

$$\delta(q_0, \mathit{read}) \quad = \quad q_1$$

corresponds to the typing

$$\mathit{read} \quad : \quad q_1 \to q_0$$

Note that the order is reversed.

The idea is that the type of a word is a state from which the word is accepted.

A type-theoretic intuition

The transition function may be seen as a typing of the letters of the word, seen as function symbols.

For example,

$$\delta(q_0, \mathit{read}) \quad = \quad q_1$$

corresponds to the typing

$$\mathit{read} \ : \ q_1 \to q_0$$

Note that the order is reversed.

The idea is that the type of a word is a state from which the word is accepted.

A type-theoretic intuition

The transition function may be seen as a typing of the letters of the word, seen as function symbols.

For example,

$$\delta(q_0, \mathit{read}) \quad = \quad q_1$$

corresponds to the typing

$$\mathit{read} \ : \ q_1 \to q_0$$

Note that the order is reversed.

The idea is that the type of a word is a state from which the word is accepted.

A type-theoretic intuition: a run of the automaton

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

A type-theoretic intuition: a run of the automaton

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

A type-theoretic intuition: a run of the automaton

$$\dfrac{\dfrac{\vdash \mathit{read} \ : \ q_1 \to q_0 \qquad \dfrac{}{\vdash \ \mathit{write} \cdot \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_1}}{\vdash \ (\mathit{read} \cdot \mathit{write})^2 \cdot \mathit{close} \ : \ q_0}}{\vdots}$$

A type-theoretic intuition: a run of the automaton

$$\dfrac{\dfrac{\vdash \mathit{read} \ : \ q_1 \to q_0 \qquad \dfrac{\vdash \mathit{write} \ : \ q_0 \to q_1 \qquad \dfrac{}{\vdash \ \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_0}}{\vdash \ \mathit{write} \cdot \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_1}}{\vdash \ (\mathit{read} \cdot \mathit{write})^2 \cdot \mathit{close} \ : \ q_0}}{\vdots}$$

and so on.

Note that the set of constructors' typings define δ.

And that typing naturally extends to programs computing words.

A type-theoretic intuition: a run of the automaton

$$\dfrac{\dfrac{\vdash \mathit{read} \ : \ q_1 \to q_0 \qquad \dfrac{\vdash \mathit{write} \ : \ q_0 \to q_1 \qquad \dfrac{}{\vdash \ \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_0}}{\vdash \ \mathit{write} \cdot \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_1}}{\vdash \ (\mathit{read} \cdot \mathit{write})^2 \cdot \mathit{close} \ : \ q_0}}{\vdots}$$

and so on.

Note that the set of constructors' typings define δ.

And that typing naturally extends to programs computing words.

A type-theoretic intuition: a run of the automaton

$$\dfrac{\dfrac{\vdash \mathit{read} \ : \ q_1 \to q_0 \qquad \dfrac{\vdash \mathit{write} \ : \ q_0 \to q_1 \qquad \dfrac{}{\vdash \ \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_0}}{\vdash \ \mathit{write} \cdot \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_1}}{\vdash \ (\mathit{read} \cdot \mathit{write})^2 \cdot \mathit{close} \ : \ q_0}}{\vdots}$$

and so on.

Note that the set of constructors' typings define δ.

And that typing naturally extends to programs computing words.

A type-theoretic intuition: a run of the automaton

$$\dfrac{\dfrac{\vdash \mathit{read} \ : \ q_1 \to q_0 \qquad \dfrac{\vdash \mathit{write} \ : \ q_0 \to q_1 \qquad \vdash \ \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_0}{\vdash \ \mathit{write} \cdot \mathit{read} \cdot \mathit{write} \cdot \mathit{close} \ : \ q_1}}{\vdash \ (\mathit{read} \cdot \mathit{write})^2 \cdot \mathit{close} \ : \ q_0}}{\vdots}$$

and so on.

Note that the set of constructors' typings define δ.

And that typing naturally extends to programs computing words.

Automata and recognition

Recall that, 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.

Automata and recognition

Recall that, 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.

A very naive model-checking problem

Now the model-checking problem can be solved by:

- computing the interpretation of a word (its denotation)
- and check whether it belongs to M

This is reminiscent of interpretations in logical models – which would allow to model-check terms as well.

Typings and interpretations. A choice for M is the one of the transition monoid of the automata. Note that it can be computed from the data of all constructors' types.

Somehow, typings compute the denotations.

A very naive model-checking problem

Now the model-checking problem can be solved by:

- computing the interpretation of a word (its denotation)
- and check whether it belongs to M

This is reminiscent of interpretations in logical models – which would allow to model-check terms as well.

Typings and interpretations. A choice for M is the one of the transition monoid of the automata. Note that it can be computed from the data of all constructors' types.

Somehow, typings compute the denotations.

A very naive model-checking problem

A more elaborate problem: what about ultimately periodic words and Büchi automata ?

We would need some model extending the monoid's behaviour with some notion of recursion (for periodicity) which would model the Büchi condition.

Alternatively, we can do this syntactically over type derivations: we get infinite-depth derivations, over which we can check whether a final state occurs infinitely.

Ideas from the typing approach may help to define an appropriate model.

A very naive model-checking problem

A more elaborate problem: what about ultimately periodic words and Büchi automata ?

We would need some model extending the monoid's behaviour with some notion of recursion (for periodicity) which would model the Büchi condition.

Alternatively, we can do this syntactically over type derivations: we get infinite-depth derivations, over which we can check whether a final state occurs infinitely.

Ideas from the typing approach may help to define an appropriate model.

A very naive model-checking problem

A more elaborate problem: what about ultimately periodic words and Büchi automata ?

We would need some model extending the monoid's behaviour with some notion of recursion (for periodicity) which would model the Büchi condition.

Alternatively, we can do this syntactically over type derivations: we get infinite-depth derivations, over which we can check whether a final state occurs infinitely.

Ideas from the typing approach may help to define an appropriate model.

A very naive model-checking problem

A more elaborate problem: what about ultimately periodic words and Büchi automata ?

We would need some model extending the monoid's behaviour with some notion of recursion (for periodicity) which would model the Büchi condition.

Alternatively, we can do this syntactically over type derivations: we get infinite-depth derivations, over which we can check whether a final state occurs infinitely.

Ideas from the typing approach may help to define an appropriate model.

Model-checking higher-order programs

We seek to extend this situation to *recursion schemes* and automata with a parity condition.

We would like to interpret the recursion scheme in an algebraic structure, so that

acceptance by the automata

of the tree of behaviours it generates would reduce to

checking whether some element belongs to the semantics

of the term.

Or, using an associated type system, to

check whether the term has an appropriate type.

Higher-order recursion schemes (quick reminder)

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:

$$S$$

Value tree of a recursion scheme

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

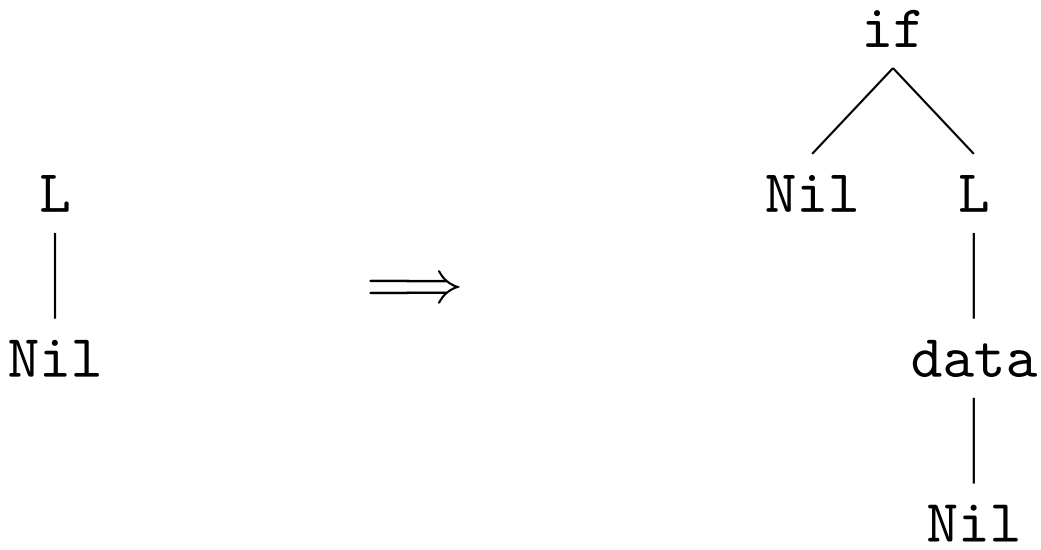
generates:

$$\mathtt{S} \Longrightarrow$$

Value tree of a recursion scheme

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

generates:

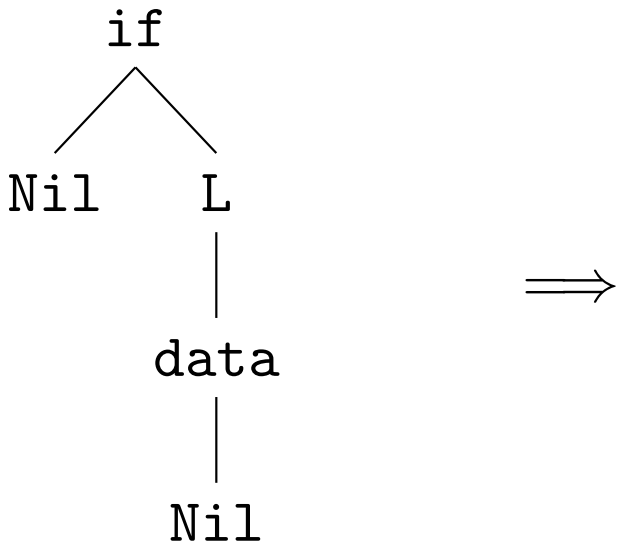


Notice that substitution and expansion occur in one same step.

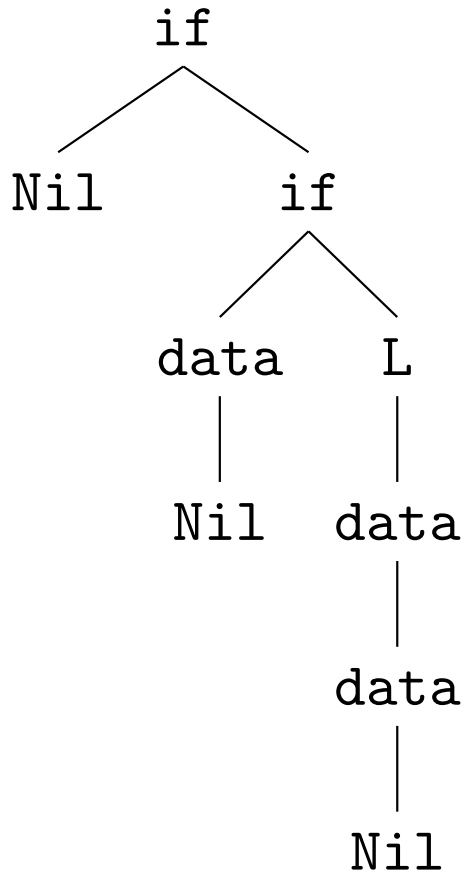
Value tree of a recursion scheme

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

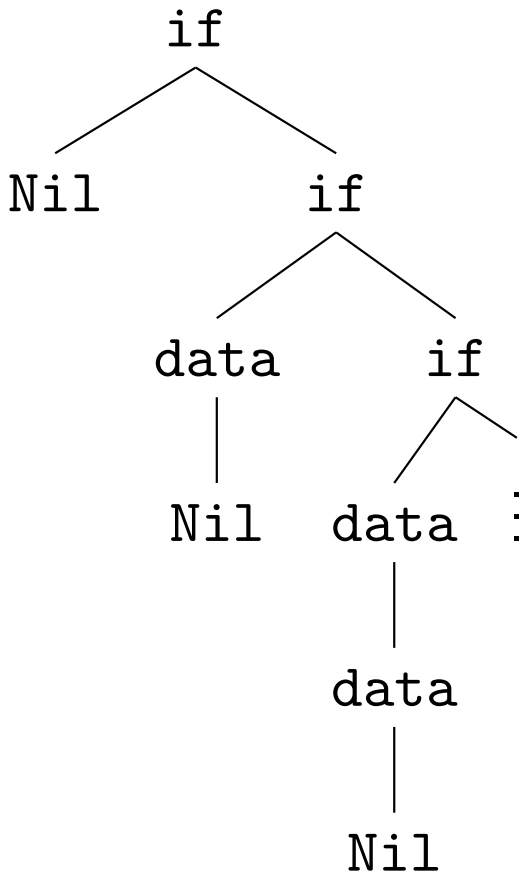
generates:



$\Longrightarrow$

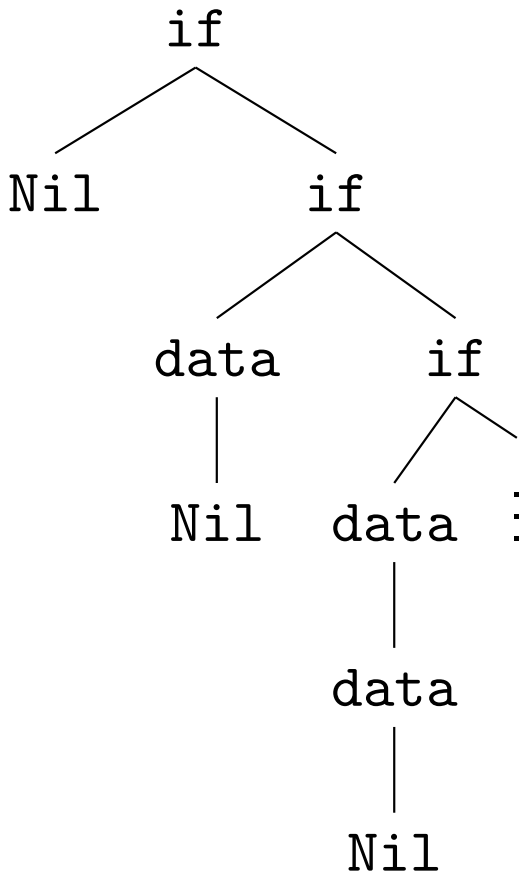


Value tree of a recursion scheme



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...

Representation of recursion schemes

The only finite representation of such a tree is actually the scheme itself — even for this very simple, order-1 recursion scheme.

This suggests that we should interpret the associated λY-term in an algebraic structure suitable for higher-order interpretations: a logical model (a domain).

A quick overview of λY-calculus

We add to the λ-calculus (to the syntax of terms) a family of operators

$$Y_\kappa \quad :: \quad (\kappa \to \kappa) \to \kappa$$

which act as fixpoint. This action is modelled by the relation δ:

$$Y\,M \;\to_\delta\; M\,(Y\,M)$$

Recursion schemes can be translated into λY-terms generating the same tree via

$$F \mapsto Y\,(\lambda F.\,\mathcal{R}(F))$$

Conversely, any λY-term of ground type without free variables can be translated to a recursion scheme.
With this translation, the evaluation of a recursion scheme amounts to the computation of the Böhm tree of the associated λY-term.

Logical specification

Alternating parity tree automata

Over trees we may use several logics: CTL, MSO,...

We focus on MSO, which is equivalent to modal μ-calculus over trees. Its automata companion model is alternating parity tree automata (APT).

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

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

This is reminiscent of the behavior of the exponential modality of linear logic...

Alternating parity tree automata

Over trees we may use several logics: CTL, MSO,...

We focus on MSO, which is equivalent to modal μ-calculus over trees. Its automata companion model is alternating parity tree automata (APT).

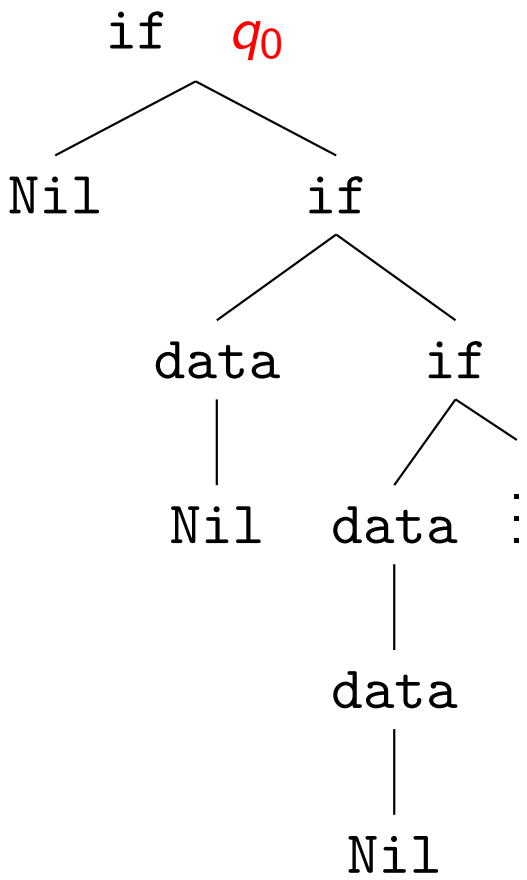
APT are non-deterministic tree automata whose transitions may duplicate or drop a subtree.

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

This is reminiscent of the behavior of the exponential modality of linear logic...

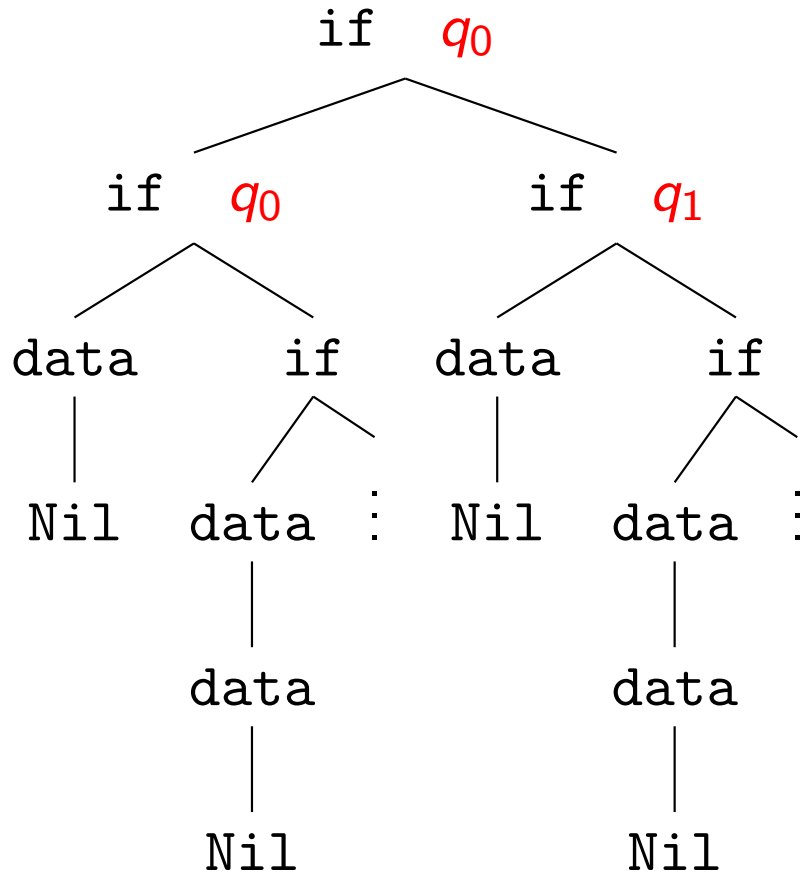
Alternating parity tree automata

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



Alternating parity tree automata

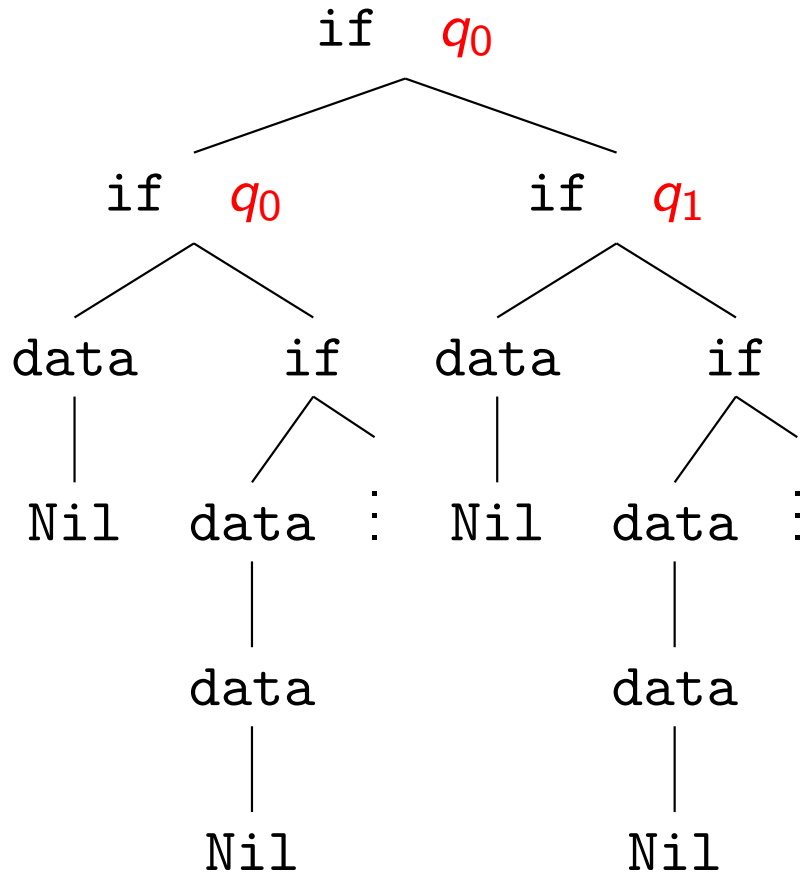
$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1).$



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

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. They are unranked.

Alternation and intersection types

Alternating parity tree automata and intersection types

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

then we may consider that a has a refined intersection type

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

Alternating parity tree automata and intersection types

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

then we may consider that a has a refined intersection type

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

Alternating parity tree automata and intersection types

This remark is very important, because unlike automata, typing lifts to higher-order.

So we may type a recursion scheme with the states of an automaton to verify if the property it expresses is satisfied.

Very important consequence: remember even very simple program models can be not regular. But schemes always are finite — and most of the time rather small.

Typing the rules of the recursion scheme is the key of Kobayashi and Ong's 2009 decidability proof.

Alternating parity tree automata and intersection types

This remark is very important, because unlike automata, typing lifts to higher-order.

So we may type a recursion scheme with the states of an automaton to verify if the property it expresses is satisfied.

Very important consequence: remember even very simple program models can be not regular. But schemes always are finite — and most of the time rather small.

Typing the rules of the recursion scheme is the key of Kobayashi and Ong's 2009 decidability proof.

A type system for verification: without colours

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

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

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

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

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

A type system for verification: without colours

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

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

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

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

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

A type system for verification: without colours

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

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

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

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

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

A type system for verification: without colours

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

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

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

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

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

A type system for verification: without colours

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

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

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

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

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

A type system for verification

In this type system, there is a proof of

$$S : \bigwedge q_0 :: o \vdash S : q_0 :: o$$

if and only if the alternating automaton has a run-tree over $[\![\mathcal{G}]\!]$.

Note that these intersection types are idempotent:

$$q_0 \wedge q_0 \quad = \quad q_0$$

Intersection type systems have been studied a lot in semantics.

Moreover, for some appropriate intersection type systems, derivations may be understood via game semantics as the construction of denotations in associated models of linear logic.

A type system for verification

In this type system, there is a proof of

$$S : \bigwedge q_0 :: o \vdash S : q_0 :: o$$

if and only if the alternating automaton has a run-tree over $[\![\mathcal{G}]\!]$.

Note that these intersection types are idempotent:

$$q_0 \wedge q_0 \quad = \quad q_0$$

Intersection type systems have been studied a lot in semantics.

Moreover, for some appropriate intersection type systems, derivations may be understood via game semantics as the construction of denotations in associated models of linear logic.

A type system for verification

In this type system, there is a proof of

$$S : \bigwedge q_0 :: o \vdash S : q_0 :: o$$

if and only if the alternating automaton has a run-tree over $[\![\mathcal{G}]\!]$.

Note that these intersection types are idempotent:

$$q_0 \wedge q_0 \quad = \quad q_0$$

Intersection type systems have been studied a lot in semantics.

Moreover, for some appropriate intersection type systems, derivations may be understood via game semantics as the construction of denotations in associated models of linear logic.

Linear models of the λ-calculus

Linear decomposition of the intuitionnistic arrow

In linear logic, the intuitionnistic arrow $A \Rightarrow B$ factors as

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

Recall that, given a categorical model of linear logic (with a suitable interpretation of !), considering only morphisms

$$!A \multimap B$$

automatically gives a model of λ-calculus (Kleisli construction).

Linear decomposition of the intuitionnistic arrow

In linear logic, the intuitionnistic arrow $A \Rightarrow B$ factors as

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

Recall that, given a categorical model of linear logic (with a suitable interpretation of !), considering only morphisms

$$!A \multimap B$$

automatically gives a model of λ-calculus (Kleisli construction).

Kleisli construction

With a "suitable interpretation" of ! comes an identity morphism

$$!A \multimap A$$

which uses an element of A once, and outputs it, and a comultiplication morphism used to define compositions:

$$!A \xrightarrow{comult} !!A \xrightarrow{!f} !B \xrightarrow{g} C$$

The resulting category is cartesian closed: it is a model of the simply-typed λ-calculus.

Models of linear logic

We would typically like to understand the refined intersection typing

$$a \ : \ (q_0 \wedge q_1) \Rightarrow q_2 \Rightarrow q \ :: \ o \to o \to o$$

as the fact that

$$(\{ q_0, q_1 \}, \{ q_2 \}, q) \quad \in \quad [\![a]\!]$$

However, set-based interpretations of the exponential lead to complicated models of linear logic.

Some additional ordering on sets is required, as well as a saturation property – roughly speaking, if a morphism can compute b out of X, it can also compute a worse output a out of a better input Y.

Models of linear logic

We would typically like to understand the refined intersection typing

$$a \ : \ (q_0 \wedge q_1) \Rightarrow q_2 \Rightarrow q \ :: \ o \to o \to o$$

as the fact that

$$(\{ q_0, q_1 \}, \{ q_2 \}, q) \quad \in \quad [\![a]\!]$$

However, set-based interpretations of the exponential lead to complicated models of linear logic.

Some additional ordering on sets is required, as well as a saturation property – roughly speaking, if a morphism can compute b out of X, it can also compute a worse output a out of a better input Y.

Models of linear logic

There are indeed two main classes of denotational models of linear logic:

- qualitative models: the exponential modality enumerates the resources used by a program, but not their multiplicity,
- quantitative models, in which the number of occurences of a resource is precisely tracked.

The former use a set-based interpretation of the exponential, the latter a multiset-based one.

Models of linear logic

Typing in Kobayashi's system corresponds to interpretation in a qualitative model of linear logic — due to idempotency of types, multiplicities are not accounted for.

(only works for η-long forms...)

It is interesting to consider quantitative interpretations as well – they are bigger, yet simpler.

They correspond to non-idempotent intersection types.

A first result: we could relate idempotent and non-idempotent typing derivations by a lifting/collapse mechanism.

Models of linear logic

Typing in Kobayashi's system corresponds to interpretation in a qualitative model of linear logic — due to idempotency of types, multiplicities are not accounted for.

(only works for η-long forms...)

It is interesting to consider quantitative interpretations as well – they are bigger, yet simpler.

They correspond to non-idempotent intersection types.

A first result: we could relate idempotent and non-idempotent typing derivations by a lifting/collapse mechanism.

Models of linear logic

Typing in Kobayashi's system corresponds to interpretation in a qualitative model of linear logic — due to idempotency of types, multiplicities are not accounted for.

(only works for η-long forms...)

It is interesting to consider quantitative interpretations as well – they are bigger, yet simpler.

They correspond to non-idempotent intersection types.

A first result: we could relate idempotent and non-idempotent typing derivations by a lifting/collapse mechanism.

Relational model of linear logic

Consider the relational model, in which

- $[\![o]\!] = Q$
- $[\![A \multimap B]\!] = [\![A]\!] \times [\![B]\!]$
- $[\![!A]\!] = \mathcal{M}_{fin}([\![A]\!])$

where $\mathcal{M}_{fin}(A)$ is the set of finite multisets of elements of $[\![A]\!]$.

We have

$$[\![A \Rightarrow B]\!] = \mathcal{M}_{fin}([\![A]\!]) \times [\![B]\!]$$

It is some collection (with multiplicities) of elements of $[\![A]\!]$ producing an element of $[\![B]\!]$.

Relational model of linear logic

Consider the relational model, in which

- $[\![o]\!] = Q$
- $[\![A \multimap B]\!] = [\![A]\!] \times [\![B]\!]$
- $[\![!A]\!] = \mathcal{M}_{fin}([\![A]\!])$

where $\mathcal{M}_{fin}(A)$ is the set of finite multisets of elements of $[\![A]\!]$.

We have

$$[\![A \Rightarrow B]\!] \quad = \quad \mathcal{M}_{fin}([\![A]\!]) \times [\![B]\!]$$

It is some collection (with multiplicities) of elements of $[\![A]\!]$ producing an element of $[\![B]\!]$.

Intersection types and relational interpretations

Consider again the typing

$$a : (q_0 \wedge q_1) \to q_2 \to q :: o \to o \to o$$

In the relational model:

$$[\![A]\!] \subseteq \mathcal{M}_{fin}(Q) \times \mathcal{M}_{fin}(Q) \times Q$$

and this example translates as

$$([q_0, q_1], [q_2], q) \in [\![a]\!]$$

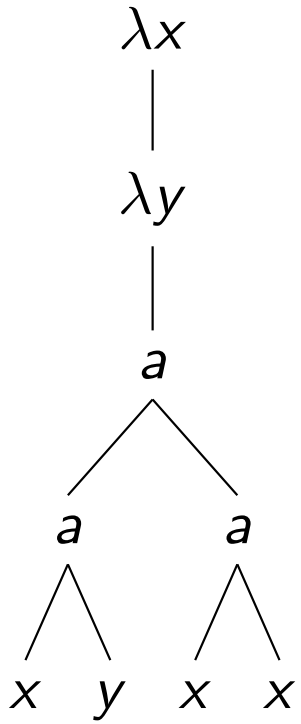
An example of interpretation

Terms are interpreted as subsets of the interpretation of their simple type.

Consider the rule

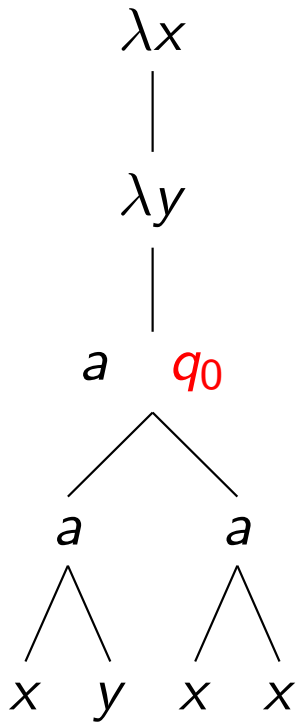
$$F\ x\ y\ =\ a\ (\ a\ x\ y\ )\ (\ a\ x\ x\ )$$

which corresponds to



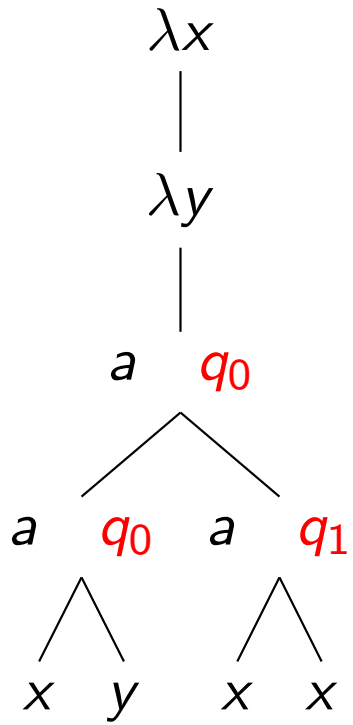
An example of interpretation

and suppose that $\mathcal{A}$ may run as follows on the tree:

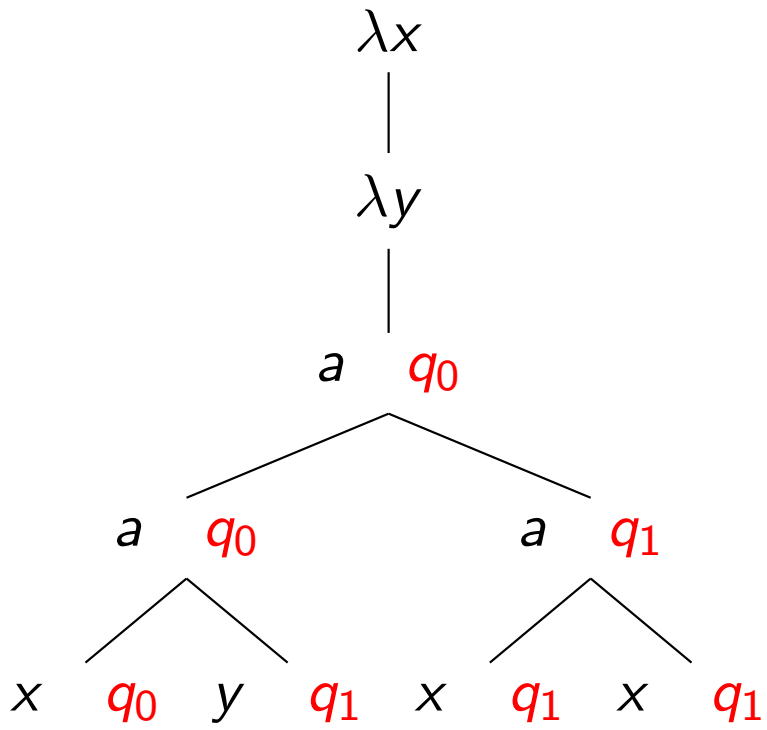


An example of interpretation

and suppose that $\mathcal{A}$ may run as follows on the tree:



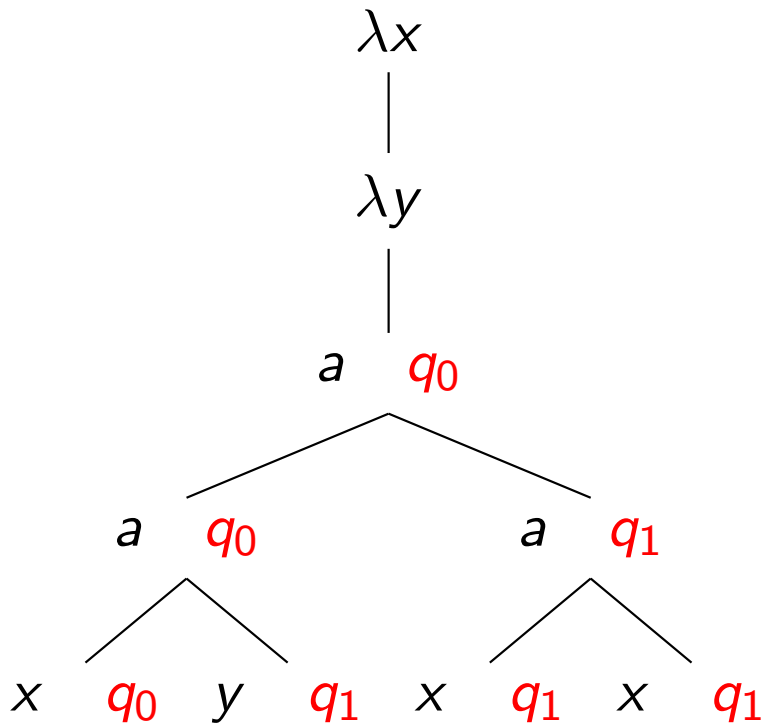
An example of interpretation



Then this rule will be interpreted in the model as

$$([q_0, q_1, q_1], [q_1], q_0)$$

An example of interpretation



Then this rule will be interpreted in the model as

$$([q_0, q_1, q_1], [q_1], q_0)$$

Relational interpretation and automata acceptance

There is an inductive fixed point operator in the model, which allows to generate finite trees.

Theorem (G.-Melliès 2014)

Consider an alternating tree automaton $\mathcal{A}$ and a λY-term t reducing to a tree T.

Then $\mathcal{A}$ has a finite run-tree over T if and only if

$$q_0 \in [\![t]\!]$$

where the interpretation is computed in the relational model.

Elements of proof

The proof relies on

- a theorem, reformulated from Kobayashi and Ong's original approach, giving an equivalence between the existence of a run-tree and the existence of a typing in an intersection type system,
- on a translation theorem stating the equivalence of this type system with a type system derived from the intuitionnistic fragment of Bucciarelli and Ehrhard's indexed linear logic
- and on a correspondence between the typing proofs of the latter system and the relational denotations of terms.

Hidden relation between qualitative and quantitative semantics. . .

A duality between terms and alternating automata

Linear typing of tree-producing terms

Consider a λ-term $t :: o$ reducing to a tree T over the signature

$$\Sigma \quad = \quad \{ a : 2, b : 1, c : 0 \}$$

Treating a, b and c as free variables, we obtain by Church encoding the λ-term

$$\lambda a.\, \lambda b.\, \lambda c.\, t \quad : \quad (o \Rightarrow o \Rightarrow o) \Rightarrow (o \Rightarrow o) \Rightarrow o \Rightarrow o$$

which can be typed by the following formula of linear logic:

$$A \quad = \quad !\,(!\,o \multimap !\,o \multimap o) \multimap !\,(!\,o \multimap o) \multimap !\,o \multimap o$$

Linear typing of tree-producing terms

Consider a λ-term $t :: o$ reducing to a tree T over the signature

$$\Sigma \quad = \quad \{ a : 2, b : 1, c : 0 \}$$

Treating a, b and c as free variables, we obtain by Church encoding the λ-term

$$\lambda a.\, \lambda b.\, \lambda c.\, t \quad : \quad (o \Rightarrow o \Rightarrow o) \Rightarrow (o \Rightarrow o) \Rightarrow o \Rightarrow o$$

which can be typed by the following formula of linear logic:

$$A \quad = \quad !\,(\,!\,o \multimap !\,o \multimap o\,) \multimap !\,(\,!\,o \multimap o\,) \multimap !\,o \multimap o$$

Linear typing of tree-producing terms

Consider a λ-term $t :: o$ reducing to a tree T over the signature

$$\Sigma \quad = \quad \{\, a : 2,\, b : 1,\, c : 0 \,\}$$

Treating a, b and c as free variables, we obtain by Church encoding the λ-term

$$\lambda a.\, \lambda b.\, \lambda c.\, t \quad : \quad (o \Rightarrow o \Rightarrow o) \Rightarrow (o \Rightarrow o) \Rightarrow o \Rightarrow o$$

which can be typed by the following formula of linear logic:

$$A \quad = \quad !\,(\,!\,o \multimap !\,o \multimap\, o\,) \;\multimap !\,(\,!\,o \multimap\, o\,) \;\multimap !\,o \;\multimap\; o$$

Linear typing of tree-producing terms

Its dual $A^\perp$ is

$$A^\perp \;=\; !\,(\,!\,o \multimap !\,o \multimap o\,) \otimes !\,(\,!\,o \multimap o\,) \otimes !\,o \otimes (o)^\perp$$

The logic lacks non-determinism, but in relational semantics, this is precisely the type of (the encoding of) alternating parity automata. Indeed, interpreting o as Q:

$$A^\perp = !\,(\,!\,Q \multimap !\,Q \multimap Q) \otimes !\,(!\,Q \multimap Q) \otimes !\,Q \otimes Q^\perp.$$

The element of $o^\perp$ is the initial state, and the remaining encodes the transition function of the automaton.

Linear typing of tree-producing terms

Its dual $A^\perp$ is

$$A^\perp \;=\; !\,(\,!\,o \multimap !\,o \multimap o\,) \otimes !\,(\,!\,o \multimap o\,) \otimes !\,o \otimes (o)^\perp$$

The logic lacks non-determinism, but in relational semantics, this is precisely the type of (the encoding of) alternating parity automata. Indeed, interpreting o as Q:

$$A^\perp = !\,(\,!\,Q \multimap !\,Q \multimap Q) \otimes !\,(!\,Q \multimap Q) \otimes !\,Q \otimes Q^\perp.$$

The element of $o^\perp$ is the initial state, and the remaining encodes the transition function of the automaton.

Relational interpretation and automata acceptance

This duality leads to a more general version of the previous theorem:

Theorem (G.-Melliès 2014)

Consider an alternating tree automaton $\mathcal{A}$ and a λY-term t reducing to (the Church encoding of) a tree T.

Then $\mathcal{A}$ has a finite run-tree over T if and only if

$$q_0 \in [\![t]\!] \circ [\![\delta]\!]$$

where the interpretation is computed in the relational model.

In other words: the dual interpretations of a term and of an automaton interact to compute the set of accepting states of the automaton over the tree generated by the term.

Relational interpretation and automata acceptance

This duality leads to a more general version of the previous theorem:

Theorem (G.-Melliès 2014)

Consider an alternating tree automaton $\mathcal{A}$ and a λY-term t reducing to (the Church encoding of) a tree T.

Then $\mathcal{A}$ has a finite run-tree over T if and only if

$$q_0 \in [\![t]\!] \circ [\![\delta]\!]$$

where the interpretation is computed in the relational model.

In other words: the dual interpretations of a term and of an automaton interact to compute the set of accepting states of the automaton over the tree generated by the term.

An infinitary model of linear logic

An infinitary relation semantics

An infinite run-tree uses countably some elements of the signature.

We therefore need to introduce a variant of the relational semantics of linear logic, in which objects are set of cardinality at most the reals, and we introduce a new exponential modality $\lightning$:

$$[\![\lightning A]\!] \quad = \quad \mathcal{M}_{count}([\![A]\!])$$

(finite-or-countable multisets)

This exponential $\lightning$ satisfies the axioms of an exponential, and thus gives immediately an infinitary model of the λ-calculus by the Kleisli construction.

An infinitary relation semantics

An infinite run-tree uses countably some elements of the signature.

We therefore need to introduce a variant of the relational semantics of linear logic, in which objects are set of cardinality at most the reals, and we introduce a new exponential modality $\lightning$:

$$[\![\lightning A]\!] \quad = \quad \mathcal{M}_{count}([\![A]\!])$$

(finite-or-countable multisets)

This exponential $\lightning$ satisfies the axioms of an exponential, and thus gives immediately an infinitary model of the λ-calculus by the Kleisli construction.

An infinitary relation semantics

This model has a coinductive fixpoint, which performs a potentially infinite composition of the elements of the denotation of a morphism.
The Theorem then extends:

Theorem (G.-Melliès 2014)

Consider an alternating tree automaton $\mathcal{A}$ and a λY-term t producing (the Church encoding of) a tree T.

Then $\mathcal{A}$ has a possibly infinite run-tree over T if and only if

$$q_0 \in [\![t]\!] \circ [\![\delta]\!]$$

where the recursion operator of the λY-calculus is computed using the coinductive fixed point operator of the infinitary relational model.

An infinitary relation semantics

This model has a coinductive fixpoint, which performs a potentially infinite composition of the elements of the denotation of a morphism.
The Theorem then extends:

Theorem (G.-Melliès 2014)

Consider an alternating tree automaton $\mathcal{A}$ and a λY-term t producing (the Church encoding of) a tree T.

Then $\mathcal{A}$ has a possibly infinite run-tree over T if and only if

$$q_0 \in [\![t]\!] \circ [\![\delta]\!]$$

where the recursion operator of the λY-calculus is computed using the coinductive fixed point operator of the infinitary relational model.

Specifying inductive and coinductive behaviours: 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

In the APT, this inductive-coinductive policy is encoded using parity conditions. Every state receives a colour

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

Say that an infinite branch of a run-tree is winning iff the maximal colour among the ones occuring infinitely often along it is even.

Say that a run-tree is winning iff all of its infinite branches are.

Then an APT has a winning run-tree over a tree T iff the root of T satisfies the corresponding MSO formula ϕ.

Alternating parity tree automata

In the APT, this inductive-coinductive policy is encoded using parity conditions. Every state receives a colour

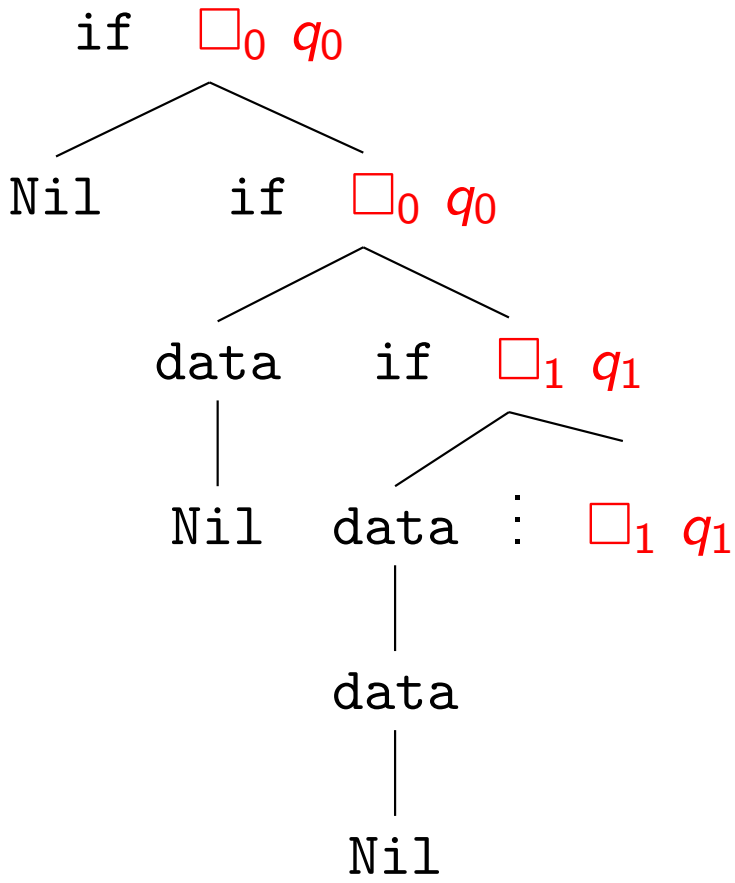
$$\Omega(q) \in Col \subseteq \mathbb{N}$$

Say that an infinite branch of a run-tree is winning iff the maximal colour among the ones occuring infinitely often along it is even.

Say that a run-tree is winning iff all of its infinite branches are.

Then an APT has a winning run-tree over a tree T iff the root of T satisfies the corresponding MSO formula ϕ.

Parity condition on an example



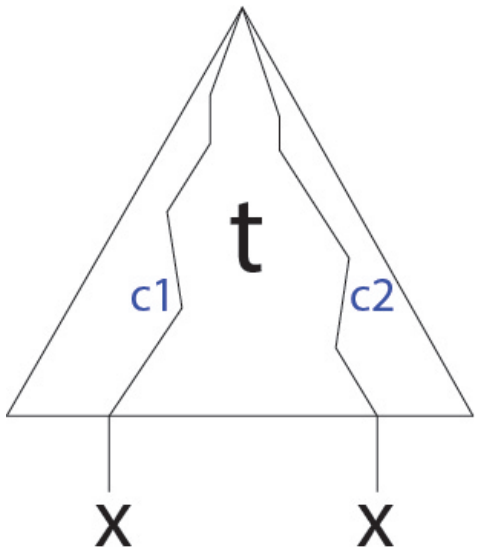
would not be a winning run-tree.

Parity conditions

Kobayashi and Ong extend the typings with colouring annotations:

$$a \ : \ (\emptyset \to \Box_{c_2} \ q_2 \to q_0) \wedge ((\Box_{c_1} \ q_1 \wedge \Box_{c_2} \ q_2) \to \Box_{c_0} \ q_0 \to q_0)$$

This operation lifts to higher-order.



In this setting, t will have some type $\Box_{c_1} \ \sigma_1 \wedge \Box_{c_2} \ \sigma_2 \to \tau$.

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

$$\text{Axiom} \quad \frac{}{x : \bigwedge_{\{i\}} \Box_\epsilon \theta_i :: \kappa \vdash x : \theta_i :: \kappa}$$

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

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

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

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

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

$$\text{Axiom} \quad \frac{}{x : \bigwedge_{\{i\}} \Box_\epsilon \theta_i :: \kappa \vdash x : \theta_i :: \kappa}$$

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

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

$$\mathit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \Box_\epsilon \theta :: \kappa \vdash F : \theta :: \kappa}$$

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

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

$$\text{Axiom} \quad \frac{}{x : \bigwedge_{\{i\}} \Box_\epsilon\, \theta_i \;::\; \kappa \;\vdash\; x : \theta_i \;::\; \kappa}$$

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

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

$$\mathit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) : \theta \;::\; \kappa}{F : \Box_\epsilon\, \theta \;::\; \kappa \;\vdash\; F : \theta \;::\; \kappa}$$

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

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

$$\text{Axiom} \quad \frac{}{x : \bigwedge_{\{i\}} \Box_\epsilon\, \theta_i \;::\; \kappa \;\vdash\; x : \theta_i \;::\; \kappa}$$

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

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

$$\mathit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) : \theta \;::\; \kappa}{F : \Box_\epsilon\, \theta \;::\; \kappa \;\vdash\; F : \theta \;::\; \kappa}$$

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

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

$$\text{Axiom} \quad \frac{}{x : \bigwedge_{\{i\}} \Box_\epsilon\, \theta_i \;::\; \kappa \;\vdash\; x : \theta_i \;::\; \kappa}$$

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

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

$$\mathit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) : \theta \;::\; \kappa}{F : \Box_\epsilon\, \theta \;::\; \kappa \;\vdash\; F : \theta \;::\; \kappa}$$

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

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

This type system can have infinite-depth derivations.

The parity condition over branches of run-trees may be reformulated as a condition over infinite branches of a derivation tree.

On a rule

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

the node $\Delta_i \vdash u\ :\ \theta_i\ ::\ \kappa$ is attributed color m_i.

Other nodes receive the neutral color ϵ, for uniformity.
But it actually means that they are uncolored.

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

This type system can have infinite-depth derivations.

The parity condition over branches of run-trees may be reformulated as a condition over infinite branches of a derivation tree.

On a rule

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

the node $\Delta_i \vdash u\ :\ \theta_i\ ::\ \kappa$ is attributed color m_i.

Other nodes receive the neutral color ϵ, for uniformity.
But it actually means that they are uncolored.

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

This type system can have infinite-depth derivations.

The parity condition over branches of run-trees may be reformulated as a condition over infinite branches of a derivation tree.

On a rule

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

the node $\Delta_i \vdash u\ :\ \theta_i\ ::\ \kappa$ is attributed color m_i.

Other nodes receive the neutral color ϵ, for uniformity.
But it actually means that they are uncolored.

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

This reformulation of the Kobayashi-Ong type system is important, as it discloses a key point of higher-order model-checking:

The coloring operation acts as a system of boxes in the type system.

Tree constructors are the only symbols creating boxes, and the rewriting of the recursion scheme preserves coherently this coloration.

But note that the finitary (β)-reduction does not suffice to evaluate the HORS with respect to the parity condition, so as to test whether a tree is winning.

Over typing trees, we need an external discrimination of run-trees. In models, we will perform this using a suitable fixed point operator.

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

This reformulation of the Kobayashi-Ong type system is important, as it discloses a key point of higher-order model-checking:

The coloring operation acts as a system of boxes in the type system.

Tree constructors are the only symbols creating boxes, and the rewriting of the recursion scheme preserves coherently this coloration.

But note that the finitary (β)-reduction does not suffice to evaluate the HORS with respect to the parity condition, so as to test whether a tree is winning.

Over typing trees, we need an external discrimination of run-trees. In models, we will perform this using a suitable fixed point operator.

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

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

Consider an alternating parity tree automaton $\mathcal{A}$ and a scheme $\mathcal{G}$ producing a tree T.

Then $\mathcal{A}$ has a winning run-tree over T if and only if there exists a winning typing tree of

$$\Gamma \vdash t(\mathcal{G}) : q_0 :: \bot$$

where $t(\mathcal{G})$ is the λ-term corresponding to $\mathcal{G}$.

Parity conditions

As in the prologue, we can take advantage of this type-theoretic approach to design an associated model.

Semantically, the previous remark about the system of boxes induced by coloring means that it defines a parametric comonad.

On objects:

$$\Box A \quad = \quad Col \times A$$

where $Col = \Omega(Q) \uplus \{\epsilon\}$ is the set of colors.

The structural morphisms act as

$$\Box_{\max(m_1,m_2)}\ a \ \multimap\ \Box_{m_1}\ \Box_{m_2}\ a$$

$$\Box_{\epsilon}\ a \ \multimap\ a$$

Parity conditions

As in the prologue, we can take advantage of this type-theoretic approach to design an associated model.

Semantically, the previous remark about the system of boxes induced by coloring means that it defines a parametric comonad.

On objects:

$$\Box A \quad = \quad Col \times A$$

where $Col = \Omega(Q) \uplus \{\epsilon\}$ is the set of colors.

The structural morphisms act as

$$\Box_{\max(m_1,m_2)}\ a \multimap \Box_{m_1}\ \Box_{m_2}\ a$$

$$\Box_{\epsilon}\ a \multimap a$$

Parity conditions

As in the prologue, we can take advantage of this type-theoretic approach to design an associated model.

Semantically, the previous remark about the system of boxes induced by coloring means that it defines a parametric comonad.

On objects:

$$\Box A \quad = \quad Col \times A$$

where $Col = \Omega(Q) \uplus \{\epsilon\}$ is the set of colors.

The structural morphisms act as

$$\Box_{\max(m_1,m_2)}\ a \ \multimap\ \Box_{m_1}\ \Box_{m_2}\ a$$

$$\Box_{\epsilon}\ a \ \multimap\ a$$

Parity conditions

The modality $\Box$ distributes over the exponential $?$: there is a natural transformation

$$?\Box \to \Box?$$

satisfying some coherence diagram.

It follows that the composite

$$!? = ?\Box$$

is an exponential, so that we automatically obtain a model of the λ-calculus associated to the coloured typings.

Parity conditions

The modality $\square$ distributes over the exponential $\lightning$: there is a natural transformation

$$\lightning \square \quad \to \square \lightning$$

satisfying some coherence diagram.

It follows that the composite

$$\lightning = \lightning \square$$

is an exponential, so that we automatically obtain a model of the λ-calculus associated to the coloured typings.

Linear decomposition of the intuitionnistic arrow

Kleisli composition: consider

$$f \ : \ \lightning \Box A \to B$$

and

$$g \ : \ \lightning \Box B \to C$$

Their composite is defined as

$$\lightning \Box B \xrightarrow{g} C$$

where λ is the distributivity law between ! and $\Box$.

Linear decomposition of the intuitionnistic arrow

Kleisli composition: consider

$$f \ : \ \lightning \Box A \to B$$

and

$$g \ : \ \lightning \Box B \to C$$

Their composite is defined as

$$\lightning \Box \lightning \Box A \xrightarrow{\lightning \Box f} \lightning \Box B \xrightarrow{g} C$$

where λ is the distributivity law between ! and $\Box$.

Linear decomposition of the intuitionnistic arrow

Kleisli composition: consider

$$f \ : \ !\Box A \to B$$

and

$$g \ : \ !\Box B \to C$$

Their composite is defined as

$$!!\Box\Box A \xrightarrow{\lambda} !\Box!\Box A \xrightarrow{!\Box f} !\Box B \xrightarrow{g} C$$

where λ is the distributivity law between ! and $\Box$.

Linear decomposition of the intuitionnistic arrow

Kleisli composition: consider

$$f : !\Box A \to B$$

and

$$g : !\Box B \to C$$

Their composite is defined as

$$!\Box\Box A \to !!\Box\Box A \xrightarrow{\lambda} !\Box!\Box A \xrightarrow{!\Box f} !\Box B \xrightarrow{g} C$$

where λ is the distributivity law between ! and $\Box$.

Linear decomposition of the intuitionnistic arrow

Kleisli composition: consider

$$f \ : \ !\Box A \to B$$

and

$$g \ : \ !\Box B \to C$$

Their composite is defined as

$$!\Box A \ \to \ !\Box\Box A \ \to \ !!\Box\Box A \ \xrightarrow{\lambda} \ !\Box !\Box A \ \xrightarrow{!\Box f} \ !\Box B \xrightarrow{g} \ C$$

where λ is the distributivity law between ! and $\Box$.

Parity conditions

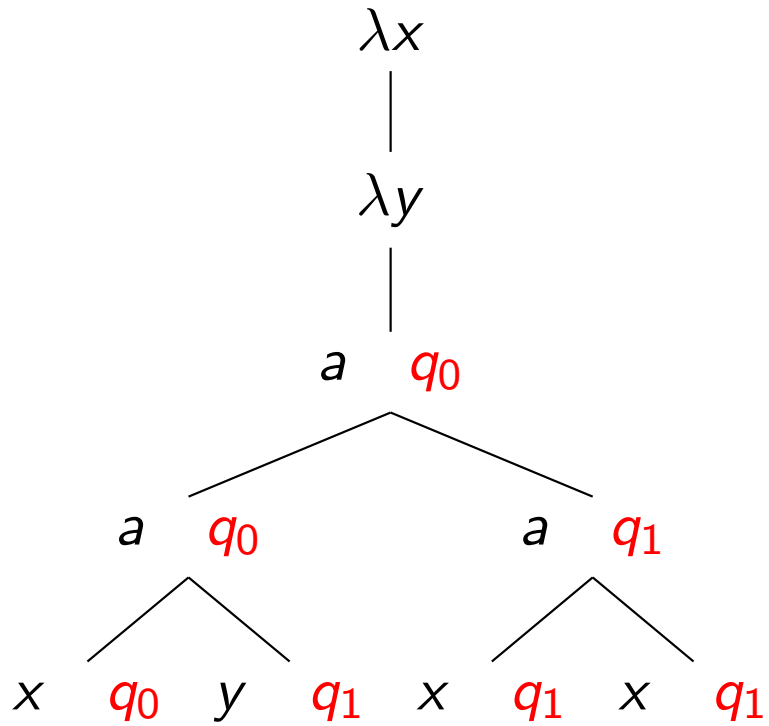
We obtain a very natural colored interpretation of types:

$$[\![A \Rightarrow B]\!] = \mathcal{M}_{count}(Col \times [\![A]\!]) \times [\![B]\!]$$

and we can relate the typing derivations in the colored intersection type system with the construction of denotations in the resulting model.

An example of coloured interpretation

Suppose $\Omega(q_0) = 0$ and $\Omega(q_1) = 1$.

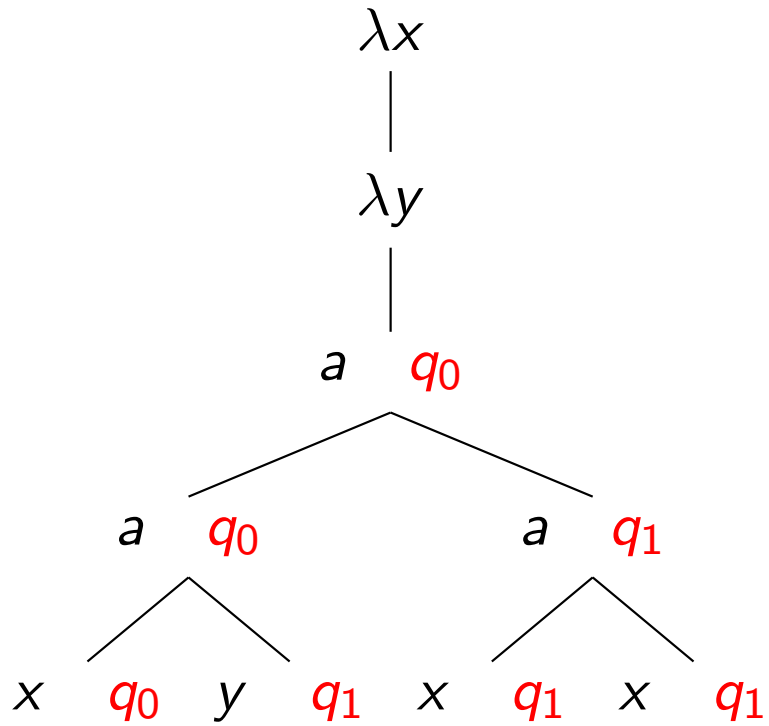


This rule will be interpreted in the model as

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

An example of coloured interpretation

Suppose $\Omega(q_0) = 0$ and $\Omega(q_1) = 1$.



This rule will be interpreted in the model as

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

Connection with the coloured relational model

To obtain the acceptance theorem for alternating parity automata, we need a fixpoint which reflects the parity condition.

This operator composes denotations infinitely, and only keeps the result if it comes from a winning composition tree.

Current work: define this fixpoint by combining the inductive and coinductive ones ?

Connection with the coloured relational model

To obtain the acceptance theorem for alternating parity automata, we need a fixpoint which reflects the parity condition.

This operator composes denotations infinitely, and only keeps the result if it comes from a winning composition tree.

Current work: define this fixpoint by combining the inductive and coinductive ones ?

Connection with the coloured relational model

Theorem (G.-Melliès 2015)

Consider an alternating parity tree automaton $\mathcal{A}$ and a λY-term t producing (the Church encoding of) a tree T.

Then $\mathcal{A}$ has a winning run-tree over T if and only if

$$q_0 \in [\![t]\!] \circ [\![\delta]\!]$$

A finitary coloured model of the λY-calculus

Extensional collapses

In order to get a decidability proof and an estimation of complexity, we need to recast our work in a finitary setting.

If the exponential modality ! is interpreted with finite sets, we obtain the poset-based model of linear logic (a.k.a. its Scott model).

Ehrhard proved in 2012 that it is the extensional collapse of the relational model.

Extensional collapses

In order to get a decidability proof and an estimation of complexity, we need to recast our work in a finitary setting.

If the exponential modality ! is interpreted with finite sets, we obtain the poset-based model of linear logic (a.k.a. its Scott model).

Ehrhard proved in 2012 that it is the extensional collapse of the relational model.

Extensional collapses

Basically, the Scott model of linear logic is a qualitative model in which

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

But it requires to carry an ordering information.

It gives a model of the λ-calculus in which

1. Types are interpreted as preorders
2. Terms are interpreted as initial segments of the preorder: if $(X, a) \in [\![t]\!]$ then for every $Y \geq X$ and $b \leq a$ we have that $(Y, b) \in [\![t]\!]$.
   In other words, if a function can produce a out of X, it can also produce a worse output b out of a better input Y.

Extensional collapses

Basically, the Scott model of linear logic is a qualitative model in which

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

But it requires to carry an ordering information.

It gives a model of the λ-calculus in which

1. Types are interpreted as preorders
2. Terms are interpreted as initial segments of the preorder: if $(X, a) \in [\![t]\!]$ then for every $Y \geq X$ and $b \leq a$ we have that $(Y, b) \in [\![t]\!]$.
   In other words, if a function can produce a out of X, it can also produce a worse output b out of a better input Y.

Extensional collapses

Basically, the Scott model of linear logic is a qualitative model in which

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

But it requires to carry an ordering information.

It gives a model of the λ-calculus in which

1. Types are interpreted as preorders
2. Terms are interpreted as initial segments of the preorder: if $(X, a) \in [\![t]\!]$ then for every $Y \geq X$ and $b \leq a$ we have that $(Y, b) \in [\![t]\!]$.
   In other words, if a function can produce a out of X, it can also produce a worse output b out of a better input Y.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $\lightning A$ building multisets with finite-or-countable multiplicities.

Not needed in the finitary approach, we can use the finitary exponential !.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $\lightning A$ building multisets with finite-or-countable multiplicities.

Not needed in the finitary approach, we can use the finitary exponential !.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $\lightning A$ building multisets with finite-or-countable multiplicities,
2. It features a parametric comonad $\square$, which propagates the colouring information of the APT in the denotations.

We can define it in the same way, we just need to take care of the saturation requirements.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $!A$ building multisets with finite-or-countable multiplicities,
2. It features a parametric comonad $\Box$, which propagates the colouring information of the APT in the denotations.

We can define it in the same way, we just need to take care of the saturation requirements.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $\lightning A$ building multisets with finite-or-countable multiplicities,
2. It features a parametric comonad $\Box$, which propagates the colouring information of the APT in the denotations,
3. There is a distributive law $\lambda : \lightning \Box \to \Box \lightning$, so that these two modalities can be composed to obtain a coloured exponential $\mathbf{\lightning}$, giving by the Kleisli construction a coloured model of the λ-calculus.

Again, we define

$$\lambda : ! \Box \to \Box !$$

it in the same way, we just need to take care of the saturation requirements.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $\lightning A$ building multisets with finite-or-countable multiplicities,
2. It features a parametric comonad $\Box$, which propagates the colouring information of the APT in the denotations,
3. There is a distributive law $\lambda : \lightning \Box \to \Box \lightning$, so that these two modalities can be composed to obtain a coloured exponential $\lightning$, giving by the Kleisli construction a coloured model of the λ-calculus.

Again, we define

$$\lambda : ! \Box \to \Box !$$

it in the same way, we just need to take care of the saturation requirements.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $\lightning A$ building multisets with finite-or-countable multiplicities,
2. It features a parametric comonad $\Box$, which propagates the colouring information of the APT in the denotations,
3. There is a distributive law $\lambda : \lightning \Box \to \Box \lightning$, so that these two modalities can be composed to obtain a coloured exponential $\mathbf{!}$, giving by the Kleisli construction a coloured model of the λ-calculus,
4. There is a coloured parameterized fixed point operator Y which extends this cartesian closed category to a model of the λY-calculus.

One more time: in the same way, we just need to take care of the saturation requirements.

Recipes to obtain a coloured model

1. The model is infinitary: there is an exponential $\lightning A$ building multisets with finite-or-countable multiplicities,
2. It features a parametric comonad $\Box$, which propagates the colouring information of the APT in the denotations,
3. There is a distributive law $\lambda : \lightning \Box \to \Box \lightning$, so that these two modalities can be composed to obtain a coloured exponential $\mathbf{!}$, giving by the Kleisli construction a coloured model of the λ-calculus,
4. There is a coloured parameterized fixed point operator Y which extends this cartesian closed category to a model of the λY-calculus.

One more time: in the same way, we just need to take care of the saturation requirements.

Denotations, type-theoretically

Note that there is, again, a nice way to present the

computation of denotations

using the

typings of terms

in an associated type system.

In fact, the denotation of a closed term t is the set of elements $\alpha \in [\![type(t)]\!]$ such that

$$\emptyset \vdash t : \alpha :: type(t)$$

has a winning derivation tree in the associated type system.

Denotations, type-theoretically

Note that there is, again, a nice way to present the

computation of denotations

using the

typings of terms

in an associated type system.

In fact, the denotation of a closed term t is the set of elements $\alpha \in [\![type(t)]\!]$ such that

$$\emptyset \quad \vdash \quad t \ : \ \alpha \ :: \ type(t)$$

has a winning derivation tree in the associated type system.

Denotations, type-theoretically

Following Terui, we can present type-theoretically the computation of derivations in the resulting model of the λY-calculus:

$$\mathrm{Ax}\quad \frac{\exists\, \alpha' \in X \quad \alpha \leq_{[\![\sigma]\!]_{fin}} \alpha'}{x : X :: \sigma \vdash x : \alpha :: \sigma}$$

$$\delta\quad \frac{\alpha \text{ refines } \delta \text{ from } a}{\emptyset \vdash a : \alpha :: \sigma}$$

$$\lambda\quad \frac{\Gamma, x : X :: \sigma \vdash M : \alpha :: \tau}{\Gamma \vdash \lambda x.M : X \to \alpha :: \sigma \to \tau}$$

$$\frac{\Gamma_0 \vdash M : \{\Box_{c_1} \beta_1, \ldots, \Box_{c_n} \beta_n\} \to \alpha :: \sigma \to \tau \qquad \Gamma_i \vdash N : \beta_i :: \sigma \quad (\forall i)}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \vdash MN : \alpha :: \tau}$$

Denotations, type-theoretically

Following Terui, we can present type-theoretically the computation of derivations in the resulting model of the λY-calculus:

$$\mathrm{Ax}\quad \frac{\exists\, \alpha' \in X \quad \alpha \leq_{[\![\sigma]\!]_{fin}} \alpha'}{x : X :: \sigma \vdash x : \alpha :: \sigma}$$

$$\delta\quad \frac{\alpha \text{ refines } \delta \text{ from } a}{\emptyset \vdash a : \alpha :: \sigma}$$

$$\lambda\quad \frac{\Gamma, x : X :: \sigma \vdash M : \alpha :: \tau}{\Gamma \vdash \lambda x.M : X \to \alpha :: \sigma \to \tau}$$

$$\frac{\Gamma_0 \vdash M : \{\Box_{c_1} \beta_1, \ldots, \Box_{c_n} \beta_n\} \to \alpha :: \sigma \to \tau \qquad \Gamma_i \vdash N : \beta_i :: \sigma \quad (\forall i)}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \vdash MN : \alpha :: \tau}$$

Denotations, type-theoretically

Following Terui, we can present type-theoretically the computation of derivations in the resulting model of the λY-calculus:

$$\mathrm{Ax}\quad \frac{\exists\, \alpha' \in X \quad \alpha \ \leq_{[\![\sigma]\!]_{fin}}\ \alpha'}{x : X :: \sigma \ \vdash\ x : \alpha :: \sigma}$$

$$\delta\quad \frac{\alpha \text{ refines } \delta \text{ from } a}{\emptyset \ \vdash\ a : \alpha :: \sigma}$$

$$\lambda\quad \frac{\Gamma, x : X :: \sigma \ \vdash\ M : \alpha :: \tau}{\Gamma \ \vdash\ \lambda x.M : X \to \alpha :: \sigma \to \tau}$$

$$\frac{\Gamma_0 \ \vdash\ M : \{\Box_{c_1} \beta_1, \ldots, \Box_{c_n} \beta_n\} \to \alpha :: \sigma \to \tau \qquad \Gamma_i \ \vdash\ N : \beta_i :: \sigma \quad (\forall i)}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \ \vdash\ M\,N : \alpha :: \tau}$$

Denotations, type-theoretically

Following Terui, we can present type-theoretically the computation of derivations in the resulting model of the λY-calculus:

$$\mathrm{Ax}\quad \frac{\exists\, \alpha' \in X \quad \alpha \ \leq_{[\![\sigma]\!]_{fin}}\ \alpha'}{x : X :: \sigma \ \vdash\ x : \alpha :: \sigma}$$

$$\delta\quad \frac{\alpha \text{ refines } \delta \text{ from } a}{\emptyset \ \vdash\ a : \alpha :: \sigma}$$

$$\lambda\quad \frac{\Gamma, x : X :: \sigma \ \vdash\ M : \alpha :: \tau}{\Gamma \ \vdash\ \lambda x.M : X \to \alpha :: \sigma \to \tau}$$

$$\frac{\Gamma_0 \ \vdash\ M : \{\Box_{c_1} \beta_1, \ldots, \Box_{c_n} \beta_n\} \to \alpha :: \sigma \to \tau \qquad \Gamma_i \ \vdash\ N : \beta_i :: \sigma \quad (\forall i)}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \ \vdash\ M\,N : \alpha :: \tau}$$

Denotations, type-theoretically

The fixpoint rule

$$\frac{\Gamma_0 \vdash M : \{\Box_{c_1} \beta_1, \ldots, \Box_{c_n} \beta_n\} \to \alpha :: \sigma \to \sigma \qquad \Gamma_i \vdash Y_\sigma M : \beta_i :: \sigma}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \vdash Y_\sigma M : \alpha :: \sigma}$$

can be translated to type recursion schemes

$$\frac{\Gamma_0, F : \{\Box_{c_1} \beta_1, \ldots, \Box_{c_n} \beta_n\} :: \sigma \vdash \mathcal{R}(F) : \alpha :: \sigma \qquad \Gamma_i \vdash F : \beta_i :: \sigma}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \vdash F : \alpha :: \sigma}$$

The ordering relation

$$\frac{}{q \leq_{\perp\!\!\!\perp} q}$$

$$\frac{\forall (c, \alpha) \in X \quad \exists (c, \beta) \in Y \quad \alpha \leq_A \beta}{X \leq_{\lightning A} Y}$$

$$\frac{Y \leq_{\lightning A} X \qquad \alpha \leq_B \beta}{X \to \alpha \leq_{\lightning A \multimap B} Y \to \beta}$$

Denotations and typing derivations

We recast the parity condition over derivation trees, and obtain

Theorem

Given a λY-term t, the sequent

$$\Gamma \;=\; x_1 : X_1 :: \sigma_1, \ldots, x_n : X_n :: \sigma_n \;\vdash\; t : \alpha :: \tau$$

has a winning derivation tree in the type system with recursion iff

$$(X_1, \ldots, X_n, \alpha) \in [\![\Gamma \vdash t :: \tau]\!]_{fin} \subseteq (\, !\,[\![\sigma_1]\!]_{fin} \otimes \cdots \otimes !\,[\![\sigma_n]\!]_{fin}\,) \multimap [\![\tau$$

where the denotation is computed in the finitary coloured model enriched with a coloured parameterized fixed point operator.

Connection with higher-order model-checking

Theorem (G.-Melliès 2015)

Consider an alternating parity tree automaton $\mathcal{A}$ and a higher-order recursion scheme $\mathcal{G}$ producing a tree T.

Then $\mathcal{A}$ has a winning run-tree over T if and only if

$$q_0 \in [\![\mathcal{G}]\!]$$

where the interpretation is taken in this finitary, coloured model.

Decidability of higher-order model-checking

Note that the finiteness of the model implies, together with the memoryless decidability of parity games, that every element of the denotation of a term can be extracted from a finitary typing: a finite typing derivation with backtracking pointers, which unravels to the original one.

This implies:

Theorem

The higher-order model-checking problem is decidable.

Decidability of higher-order model-checking

The order of a scheme/of a term can be understood as a measure of its complexity.

It somehow characterizes the size of its set of refined intersection types/of its finitary denotation.

Proposition

If $\mathcal{G}$ has order n, then the complexity of the problem is $O(n\text{-}EXPTIME)$.

Decidability of selection

Given an APT $\mathcal{A}$ and a recursion scheme $\mathcal{G}$, the selection problem is to compute $\mathcal{G}'$ whose value tree is a winning run-tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$.

From a finitary typing, we can build such a scheme, leading to

Theorem

The APT selection problem is decidable.

In fact, this comes from an even stronger result: we can design a new scheme $\mathcal{G}'$ evaluating to a representation of a typing derivation for $\mathcal{G}$ – that is, to a valid computation of a denotation of $\mathcal{G}$ in the finitary model.

Decidability of selection

Given an APT $\mathcal{A}$ and a recursion scheme $\mathcal{G}$, the selection problem is to compute $\mathcal{G}'$ whose value tree is a winning run-tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$.

From a finitary typing, we can build such a scheme, leading to

Theorem

The APT selection problem is decidable.

In fact, this comes from an even stronger result: we can design a new scheme $\mathcal{G}'$ evaluating to a representation of a typing derivation for $\mathcal{G}$ – that is, to a valid computation of a denotation of $\mathcal{G}$ in the finitary model.

Conclusions and perspectives

- We studied linear models of the λY-calculus, designed to reflect the behaviour of alternating parity tree automata, as well as deeply related intersection type systems.
- In spite of its infinitary nature, the relational model is convenient for the theoretical study of the problem.
- From the recipes of the relational approach, we define a finitary model, which gives a new decidability proof of the problem.
- Models are independent of the formula of interest (there is a dependency in the set of states and of colours, though).
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", define properly game semantics with parity, extend our approach to other models of automata ...

Conclusions and perspectives

- We studied linear models of the λY-calculus, designed to reflect the behaviour of alternating parity tree automata, as well as deeply related intersection type systems.
- In spite of its infinitary nature, the relational model is convenient for the theoretical study of the problem.
- From the recipes of the relational approach, we define a finitary model, which gives a new decidability proof of the problem.
- Models are independent of the formula of interest (there is a dependency in the set of states and of colours, though).
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", define properly game semantics with parity, extend our approach to other models of automata . . .

Conclusions and perspectives

- We studied linear models of the λY-calculus, designed to reflect the behaviour of alternating parity tree automata, as well as deeply related intersection type systems.
- In spite of its infinitary nature, the relational model is convenient for the theoretical study of the problem.
- From the recipes of the relational approach, we define a finitary model, which gives a new decidability proof of the problem.
- Models are independent of the formula of interest (there is a dependency in the set of states and of colours, though).
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", define properly game semantics with parity, extend our approach to other models of automata . . .

Conclusions and perspectives

- We studied linear models of the λY-calculus, designed to reflect the behaviour of alternating parity tree automata, as well as deeply related intersection type systems.
- In spite of its infinitary nature, the relational model is convenient for the theoretical study of the problem.
- From the recipes of the relational approach, we define a finitary model, which gives a new decidability proof of the problem.
- Models are independent of the formula of interest (there is a dependency in the set of states and of colours, though).
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", define properly game semantics with parity, extend our approach to other models of automata . . .

Conclusions and perspectives

- We studied linear models of the λY-calculus, designed to reflect the behaviour of alternating parity tree automata, as well as deeply related intersection type systems.
- In spite of its infinitary nature, the relational model is convenient for the theoretical study of the problem.
- From the recipes of the relational approach, we define a finitary model, which gives a new decidability proof of the problem.
- Models are independent of the formula of interest (there is a dependency in the set of states and of colours, though).
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", define properly game semantics with parity, extend our approach to other models of automata ...