Type systems and models of linear logic for higher-order verification

Charles Grellois (joint work with Paul-André Melliès)

PPS & LIAFA — Université Paris 7

February 19, 2015

Model-checking higher-order programs

A well-known approach in verification: model-checking.

- Construct a model of a program
- Specify a property in an appropriate logic
- Make them interact in order to determine whether the program satisfies the property.

Interaction is often realized by translating the formula into an equivalent automaton, which then runs over the model.

Need to balance expressivity vs. complexity in the choice of the model and of the logic.

A very naive model-checking problem

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,

$$\delta(q_0,\ read) \quad = \quad q_1$$

corresponds to the typing

$$read \ : \ q_1 \rightarrow q_0$$

Note that this order is reversed compared to usual automata. . .

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

$$\mathit{read} \; : \; q_1 \rightarrow q_0$$

Note that this order is reversed compared to usual automata...

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

$$\mathit{read} \; : \; q_1 \rightarrow q_0$$

Note that this order is reversed compared to usual automata. . .

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

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

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

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

A type-theoretic intuition: a run of the automaton

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

and so on.

A type-theoretic intuition: a run of the automaton

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

and so on.

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

Note that the interpretation depends on the choice of $\mathcal{A}$. However, the problem can be reformulated in order to remove this dependency.

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

Note that the interpretation depends on the choice of $\mathcal{A}$. However, the problem can be reformulated in order to remove this dependency.

A very naive model-checking problem

Now the model-checking problem can be solved by:

- computing the interpretation of a word
- and check whether it belongs to M

This is reminiscent of interpretations in logical models.

A very naive model-checking problem

In this talk, we explore these ideas of typing and of interpretation in logical models, in a more general case than the one of finite words and of finite automata.

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.

A useful intuition: such typing derivations relate to the construction of denotations in the 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.

A useful intuition: such typing derivations relate to the construction of denotations in the 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.

A useful intuition: such typing derivations relate to the construction of denotations in the 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.

A useful intuition: such typing derivations relate to the construction of denotations in the model.

Model-checking higher-order programs

This work is concerned with the verification of higher-order functional programs, as Java for instance.

A function may take a function as input.
Example: compose $\phi\ x = \phi(\phi(x))$

A model for such programs is higher-order recursion schemes (HORS), generating trees describing all the potential behaviours of a program.

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

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

Model-checking higher-order programs

This work is concerned with the verification of higher-order functional programs, as Java for instance.

A function may take a function as input.
Example: $\texttt{compose}\ \phi\ x\ =\ \phi(\phi(x))$

A model for such programs is higher-order recursion schemes (HORS), generating trees describing all the potential behaviours of a program.

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

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

Model-checking higher-order programs

This work is concerned with the verification of higher-order functional programs, as Java for instance.

A function may take a function as input.
Example: `compose` $\phi\ x\ =\ \phi(\phi(x))$

A model for such programs is higher-order recursion schemes (HORS), generating trees describing all the potential behaviours of a program.

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

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

Model-checking higher-order programs

This work is concerned with the verification of higher-order functional programs, as Java for instance.

A function may take a function as input.
Example: $\texttt{compose}\ \phi\ x\ =\ \phi(\phi(x))$

A model for such programs is higher-order recursion schemes (HORS), generating trees describing all the potential behaviours of a program.

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

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

Model-checking higher-order programs

This model-checking problem is decidable:

- Ong 2006 (game semantics)
- Hague-Murawski-Ong-Serre 2008 (game semantics, higher-order pushdown automata)
- Kobayashi-Ong 2009 (intersection types)
- Salvati-Walukiewicz (interpretation in finite models)
- ...

Our aim is to deepen the semantic understanding we have of this result, using existing relations between alternating automata, intersection types, (linear) logic and its models.

Model-checking higher-order programs

Is it possible to extend to this situation the setting for finite automata ?

We would like to interpret the tree of behaviours in an algebraic structure, so that

acceptance by the automata

would reduce to

checking whether some element belongs to the semantics

of the tree.

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 Σ.

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 boolean conditional `if ... then ... else ...`

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 boolean conditional `if ... then ... else ...`

A very simple functional program

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

formulated as a recursion scheme:

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

or, in λ-calculus style :

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

(this latter representation is a regular grammar – equivalently, a λY-term)

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:

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

or, in λ-calculus style :

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

(this latter representation is a regular grammar – equivalently, a λY-term)

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 – equivalently, a λY-term)

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:

S

Value tree of a recursion scheme

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

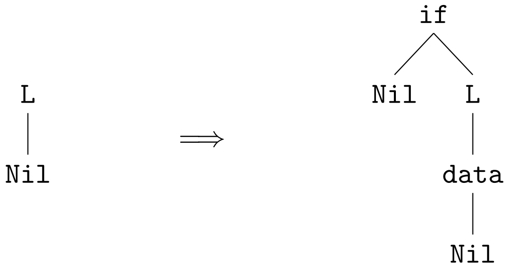
generates:

$\texttt{S} \quad \Longrightarrow$

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:

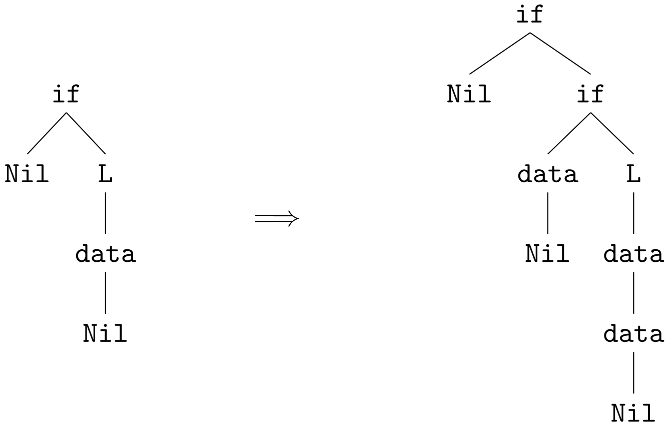


Notice that substitution and expansion occur in one same step.

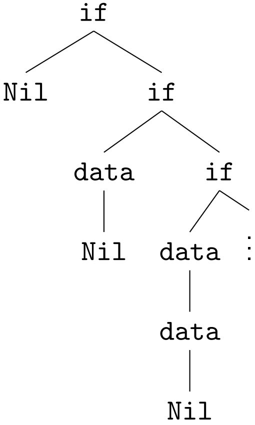
Value tree of a recursion scheme

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

generates:

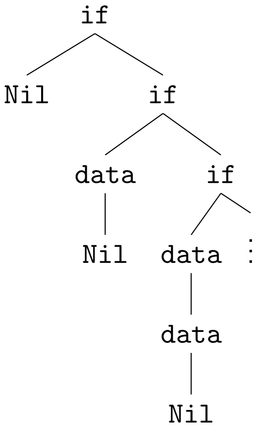


Value tree of a recursion scheme



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. . .

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).

Another recursion scheme

An order 2 example from Serre et al.:

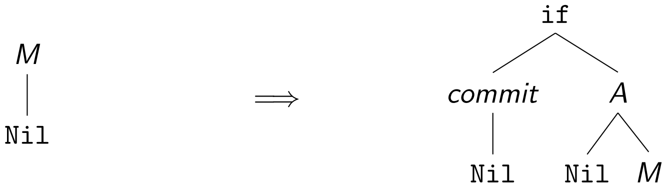
$$\begin{array}{rcl} S & = & M\ \mathtt{Nil} \\ M & = & \lambda x.\ \mathtt{if}\ (\ commit\ x\ )\ (\ A\ x\ M\ ) \\ A & = & \lambda y.\ \lambda \phi.\ \mathtt{if}\ (\ \phi\ (\ error\ end\ )\ )\ (\ \phi\ (\ cons\ y\ )\ ) \end{array}$$

with

$$\Sigma \;=\; \{\mathtt{Nil} : 0,\ \mathtt{if} : 2,\ commit : 1,\ error : 1,\ end : 0,\ cons : 1\}$$

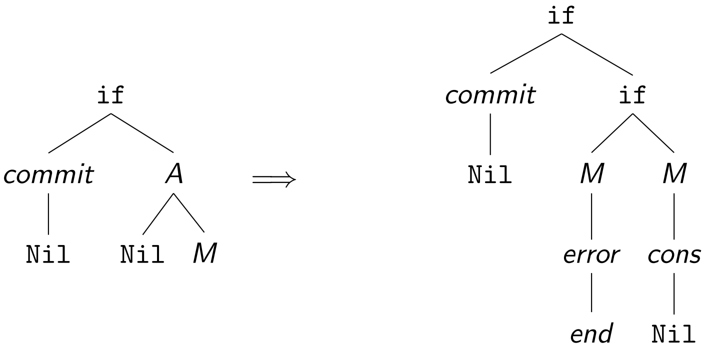
Value tree of a recursion scheme

$$
\begin{array}{rcl}
S & = & M\ \mathtt{Nil} \\
M\ x & = & \mathtt{if}\ (\ commit\ x\ )\ (\ A\ x\ M\ ) \\
A\ y\ \phi & = & \mathtt{if}\ (\ \phi\ (\ error\ end\ )\ )\ (\ \phi\ (\ cons\ y\ )\ )
\end{array}
$$



Value tree of a recursion scheme

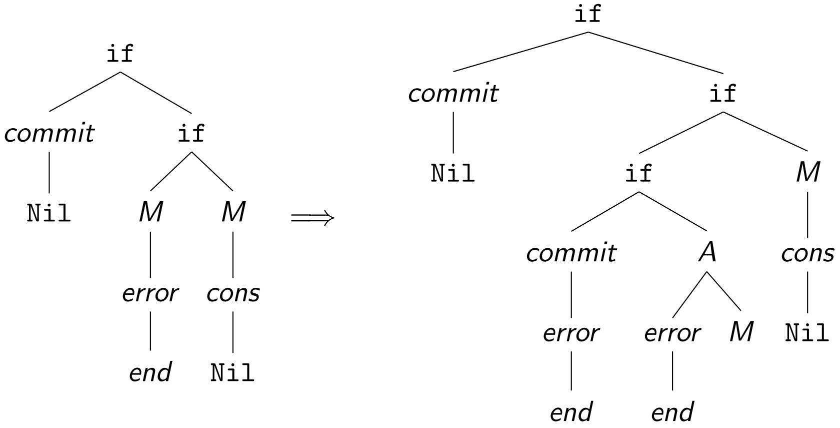
$$S \quad = \quad M \; \texttt{Nil}$$
$$M \; x \quad = \quad \texttt{if} \; (\; commit \; x \;) \; (\; A \; x \; M \;)$$
$$A \; y \; \phi \quad = \quad \texttt{if} \; (\; \phi \; (\; error \; end \;) \;) \; (\; \phi \; (\; cons \; y \;) \;)$$



Notice that two different non-terminals may be reduced here...

Value tree of a recursion scheme

$$\begin{array}{rcl} S & = & M \ \texttt{Nil} \\ M \ x & = & \texttt{if} \ (\ commit \ x \ ) \ (\ A \ x \ M \ ) \\ A \ y \ \phi & = & \texttt{if} \ (\ \phi \ (\ error \ end \ ) \ ) \ (\ \phi \ (\ cons \ y \ ) \ ) \end{array}$$



Evaluation policies

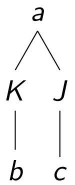
When several non-terminals occur, different reduction sequences are possible.

Consider a rule

$$L \quad = \quad \lambda x.\, \lambda y.\, a\ (K\ x)\ (J\ y)$$

An example of rewriting:

$L\ b\ c \qquad \rightarrow_{\mathcal{G}}$



Which non-terminal should we rewrite first ?

It is not very important in this case, as no rewriting of a non-terminal affects the other.

Evaluation policies

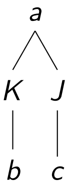
When several non-terminals occur, different reduction sequences are possible.

Consider a rule

$$L \quad = \quad \lambda x.\, \lambda y.\, a\; (K\;\; x)\; (J\;\; y)$$

An example of rewriting:

$$L\;\; b\;\; c \qquad \rightarrow_{\mathcal{G}}$$



Which non-terminal should we rewrite first ?

It is not very important in this case, as no rewriting of a non-terminal affects the other.

Evaluation policies

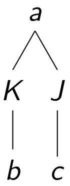
When several non-terminals occur, different reduction sequences are possible.

Consider a rule

$$L \quad = \quad \lambda x.\, \lambda y.\, a\ (K\ x)\ (J\ y)$$

An example of rewriting:

$$L\ b\ c \qquad \rightarrow_{\mathcal{G}}$$



Which non-terminal should we rewrite first ?

It is not very important in this case, as no rewriting of a non-terminal affects the other.

Evaluation policies

Consider now the rules

$$\begin{array}{rcl} S & = & J\ (K\ c) \\ K & = & \lambda x. (\ K\ (K\ x)) \\ J & = & \lambda y.\, c \end{array}$$

Which non-terminal should we rewrite first ?

Evaluation policies

Consider now the rules

$$\begin{array}{rcl} S & = & J\ (K\ c) \\ K & = & \lambda x.(\ K\ (K\ x)) \\ J & = & \lambda y.\, c \end{array}$$

Which non-terminal should we rewrite first ?

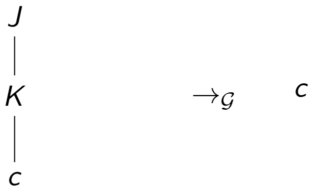
$S \quad \rightarrow_{\mathcal{G}}$

Evaluation policies

Consider now the rules

$$\begin{array}{rcl} S & = & J\ (K\ c) \\ K & = & \lambda x.(\ K\ (K\ x)) \\ J & = & \lambda y.\, c \end{array}$$

If we rewrite J:



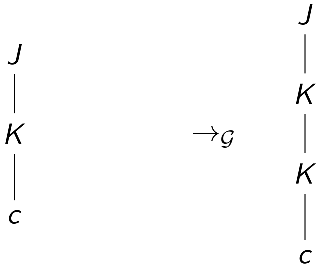
and the evaluation is finished.

Evaluation policies

Consider now the rules

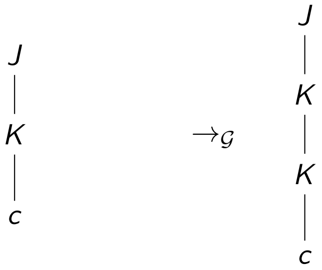
$$\begin{array}{rcl} S & = & J\ (K\ c) \\ K & = & \lambda x.(\ K\ (K\ x)) \\ J & = & \lambda y.\ c \end{array}$$

If we rewrite K:



and we have the same choice again.

Evaluation policies



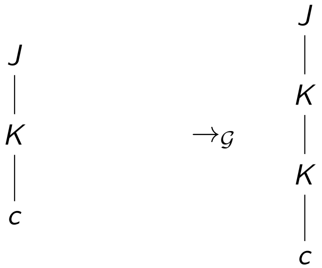
In case we always rewrite K (innermost strategy), the rewriting diverges and produces $\perp$.

For more about this, see

Axel Haddad, IO vs OI in Higher-Order Recursion Schemes

(it is very similar to CBN vs. CBV !)

Evaluation policies



In case we always rewrite K (innermost strategy), the rewriting diverges and produces $\perp$.

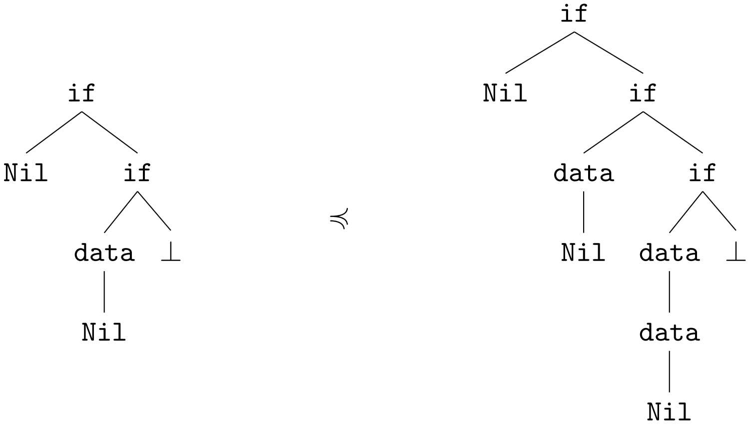
For more about this, see

Axel Haddad, IO vs OI in Higher-Order Recursion Schemes

(it is very similar to CBN vs. CBV !)

Value tree of a recursion scheme

Formally, an order over partial computation trees is defined



The value tree of $\mathcal{G}$ is defined as the supremum in this order.

A quick overview of λY-calculus

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

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

which act as fixpoint. This action is modelled by the relation δ of the λY-calculus:

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

A quick overview of λY-calculus

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

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

Conversely, any λY-term of ground type without free variables can be translated to a recursion scheme.

Important consequence: recursion schemes can be interpreted in models of the λY-calculus.

A quick overview of λY-calculus

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

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

Conversely, any λY-term of ground type without free variables can be translated to a recursion scheme.

Important consequence: recursion schemes can be interpreted in models of the λY-calculus.

Modal μ-calculus

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

In this work we use modal μ-calculus. It is equivalent to MSO over trees.

Grammar: $\phi, \psi ::= X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box \phi \mid \diamond_i \phi \mid \mu X.\phi \mid \nu X.\phi$

Modal μ-calculus

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

In this work we use modal μ-calculus. It is equivalent to MSO over trees.

Grammar: $\phi, \psi ::= X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box \phi \mid \diamond_i \phi \mid \mu X.\, \phi \mid \nu X.\, \phi$

Modal μ-calculus

Grammar: $\phi, \psi \ ::= \ X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box \phi \mid \ \diamond_i \phi \mid \mu X.\phi \mid \nu X.\phi$

X is a variable

a is a predicate corresponding to a symbol of Σ

$\Box \phi$ means that ϕ should hold on every successor of the current node

$\diamond_i \phi$ means that ϕ should hold on one successor of the current node (in direction i)

We can also define (variant) $\diamond \ = \ \bigvee_i \ \diamond_i$.

Modal μ-calculus

Grammar: $\phi, \psi \ ::= \ X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box\, \phi \mid \ \diamond_i\, \phi \mid \mu X.\, \phi \mid \nu X.\, \phi$

X is a variable

a is a predicate corresponding to a symbol of Σ

$\Box\, \phi$ means that ϕ should hold on every successor of the current node

$\diamond_i\, \phi$ means that ϕ should hold on one successor of the current node (in direction i)

We can also define (variant) $\diamond \ = \ \bigvee_i \ \diamond_i$.

Modal μ-calculus

Grammar: $\phi, \psi ::= X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box \phi \mid \Diamond_i \phi \mid \mu X . \phi \mid \nu X . \phi$

X is a variable

a is a predicate corresponding to a symbol of Σ

$\Box \phi$ means that ϕ should hold on every successor of the current node

$\Diamond_i \phi$ means that ϕ should hold on one successor of the current node (in direction i)

We can also define (variant) $\Diamond = \bigvee_i \Diamond_i$.

Modal μ-calculus

Grammar: $\phi, \psi ::= X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box\,\phi \mid \diamond_i\,\phi \mid \mu X.\,\phi \mid \nu X.\,\phi$

$\mu X.\,\phi$ is the least fixpoint of $\phi(X)$. It is computed by expanding finitely the formula:

$$\mu X.\,\phi(X) \quad \longrightarrow \quad \phi(\mu X.\,\phi(X)) \quad \longrightarrow \quad \phi(\phi(\mu X.\,\phi(X)))$$

Modal μ-calculus

Grammar: $\phi, \psi ::= X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box \phi \mid \diamond_i \phi \mid \mu X.\, \phi \mid \nu X.\, \phi$

$\nu X.\, \phi$ is the greatest fixpoint of $\phi(X)$. It is computed by expanding infinitely the formula:

$$\nu X.\, \phi(X) \quad \longrightarrow \quad \phi(\nu X.\, \phi(X)) \quad \longrightarrow \quad \phi(\phi(\nu X.\, \phi(X)))$$

Modal μ-calculus

Grammar: $\phi, \psi \ ::= \ X \mid a \mid \phi \vee \psi \mid \phi \wedge \psi \mid \Box \phi \mid \ \diamond_i \phi \mid \mu X.\, \phi \mid \nu X.\, \phi$

What does:

$$\phi \ = \ \nu X. \left(\texttt{if} \wedge \diamond_1 \left(\mu Y. \left(\texttt{Nil} \vee \Box Y \right) \right) \wedge \diamond_2 X \right)$$

mean ?

Interaction with trees: a shift to automata theory

Logic is great !

. . . but how does it interact with a tree ?

An usual approach, notably over words, is to find an equi-expressive automaton model.

Interaction with trees: a shift to automata theory

Logic is great !

. . . but how does it interact with a tree ?

An usual approach, notably over words, is to find an equi-expressive automaton model.

Interaction with trees: a shift to automata theory

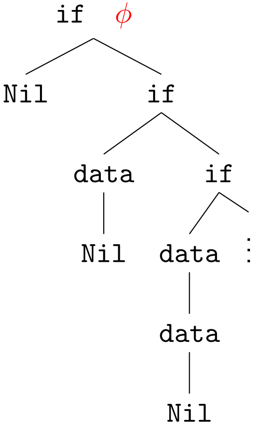
Logic is great !

. . . but how does it interact with a tree ?

An usual approach, notably over words, is to find an equi-expressive automaton model.

Alternating parity tree automata

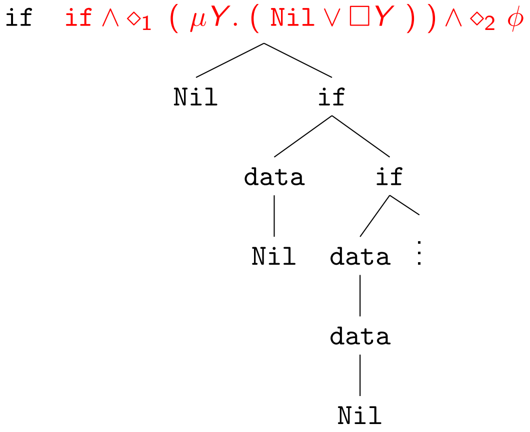
Idea: the formula "starts" on the root



where $\phi \;=\; \nu X.\, (\, \texttt{if} \wedge \diamond_1 \,(\, \mu Y.\, (\, \texttt{Nil} \vee \Box Y \,)\,) \wedge \diamond_2\, X \,)$

Alternating parity tree automata

Idea: the formula "starts" on the root

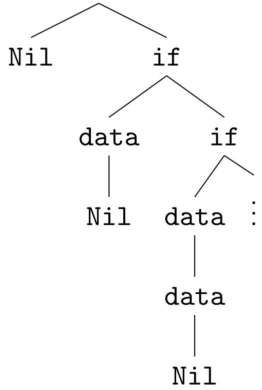


where $\phi = \nu X. \left(\mathtt{if} \wedge \diamond_1 \left(\mu Y. \left(\mathtt{Nil} \vee \Box Y \right) \right) \wedge \diamond_2 X \right)$

Alternating parity tree automata

Idea: the formula "starts" on the root

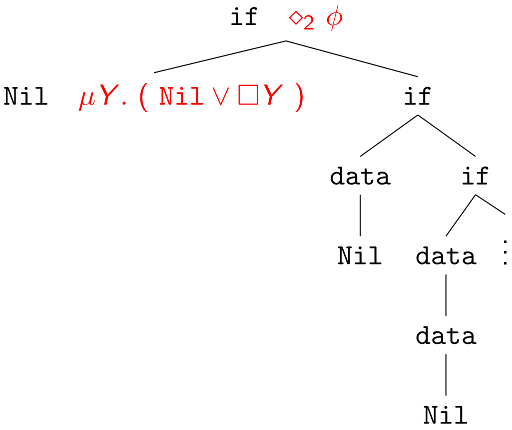
$\texttt{if} \quad \diamond_1 \; (\; \mu Y. \; (\; \texttt{Nil} \vee \Box Y \;)\;) \wedge \diamond_2 \; \phi$



where $\phi \;=\; \nu X. \; (\; \texttt{if} \wedge \diamond_1 \; (\; \mu Y. \; (\; \texttt{Nil} \vee \Box Y \;)\;) \wedge \diamond_2 \; X \;)$

Alternating parity tree automata

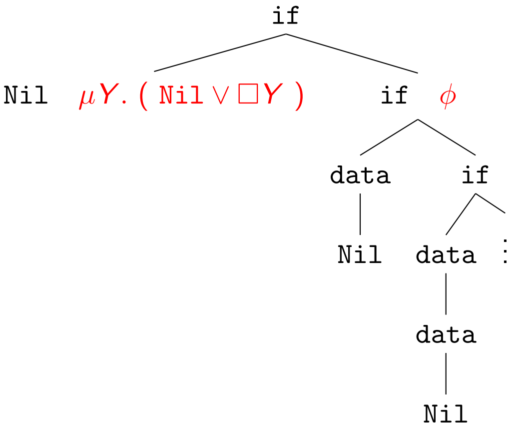
Idea: the formula "starts" on the root



where $\phi\ =\ \nu X.\ (\ \texttt{if} \wedge \Diamond_1\ (\ \mu Y.\ (\ \texttt{Nil} \vee \Box Y\ )\ ) \wedge \Diamond_2\ X\ )$

Alternating parity tree automata

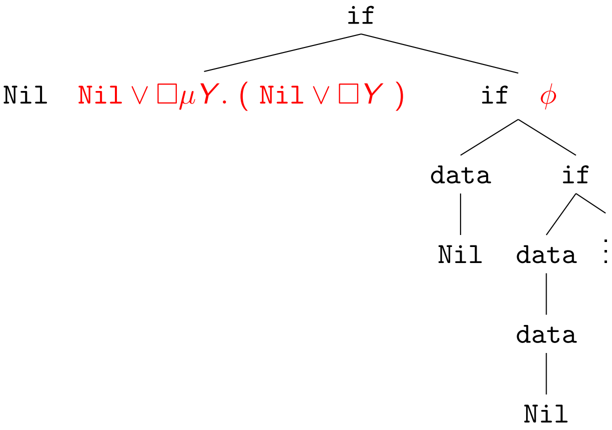
Idea: the formula "starts" on the root



where $\phi \;=\; \nu X.\,(\,\texttt{if} \wedge \diamond_1\,(\,\mu Y.\,(\,\texttt{Nil} \vee \Box Y\,)\,) \wedge \diamond_2\,X\,)$

Alternating parity tree automata

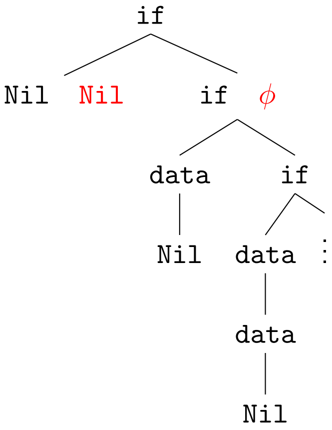
Idea: the formula "starts" on the root



where $\phi \;=\; \nu X.\, (\, \texttt{if} \wedge \diamond_1\, (\, \mu Y.\, (\, \texttt{Nil} \vee \Box Y\,)\,) \wedge \diamond_2\, X\,)$

Alternating parity tree automata

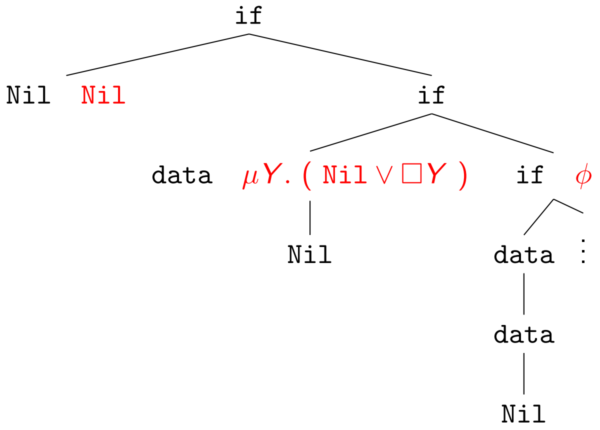
Idea: the formula "starts" on the root



where $\phi = \nu X. \; (\; \texttt{if} \wedge \diamond_1 \; (\; \mu Y. \; (\; \texttt{Nil} \vee \Box Y \;) \;) \wedge \diamond_2 X \;)$

Alternating parity tree automata

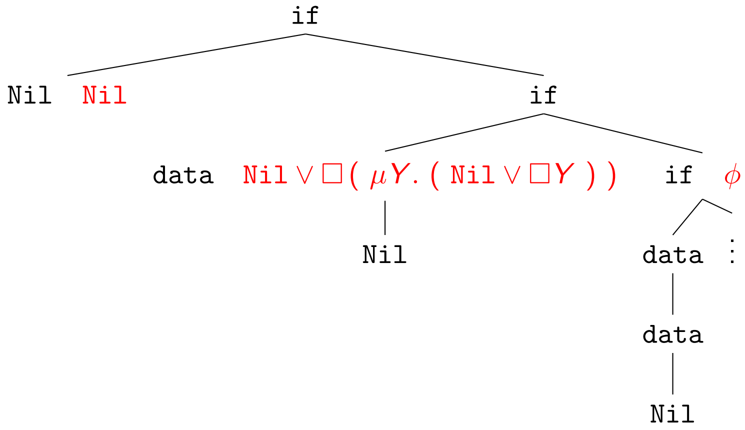
Idea: the formula "starts" on the root



where $\phi\ =\ \nu X.\ (\ \texttt{if} \wedge \diamond_1\ (\ \mu Y.\ (\ \texttt{Nil} \vee \Box Y\ )\ ) \wedge \diamond_2\ X\ )$

Alternating parity tree automata

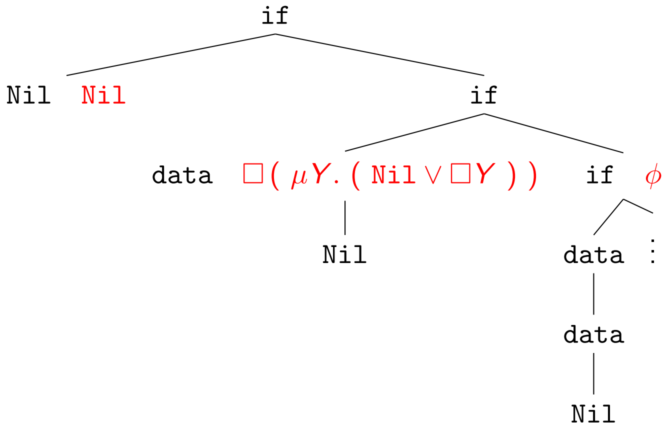
Idea: the formula "starts" on the root



where $\phi = \nu X. (\texttt{if} \wedge \diamond_1 (\mu Y. (\texttt{Nil} \vee \square Y)) \wedge \diamond_2 X)$

Alternating parity tree automata

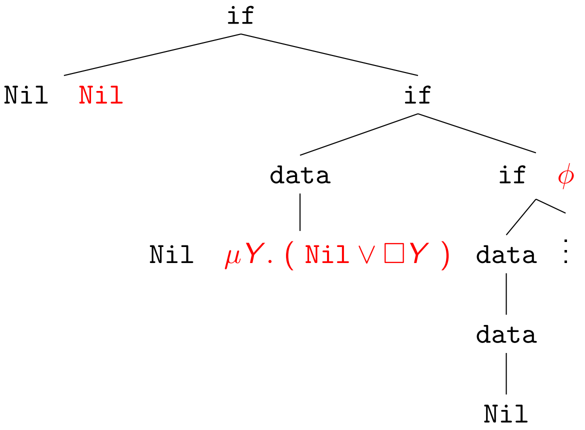
Idea: the formula "starts" on the root



where $\phi = \nu X.(\texttt{if} \wedge \diamond_1 (\mu Y.(\texttt{Nil} \vee \Box Y)) \wedge \diamond_2 X)$

Alternating parity tree automata

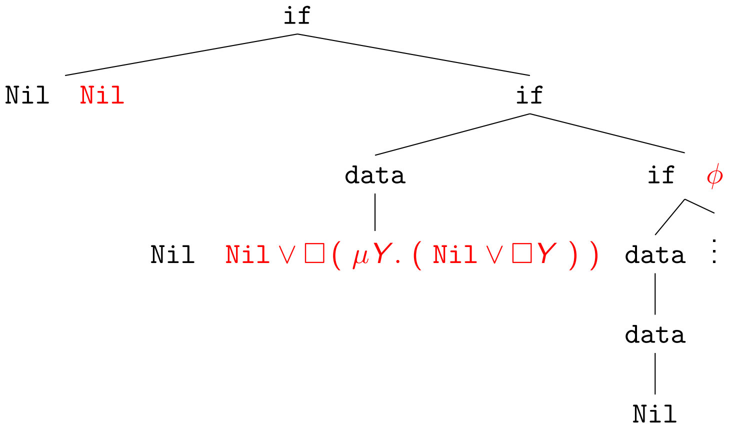
Idea: the formula "starts" on the root



where $\phi \;=\; \nu X.\,(\,\texttt{if} \wedge \diamond_1\,(\,\mu Y.\,(\,\texttt{Nil} \vee \Box Y\,)\,) \wedge \diamond_2\, X\,)$

Alternating parity tree automata

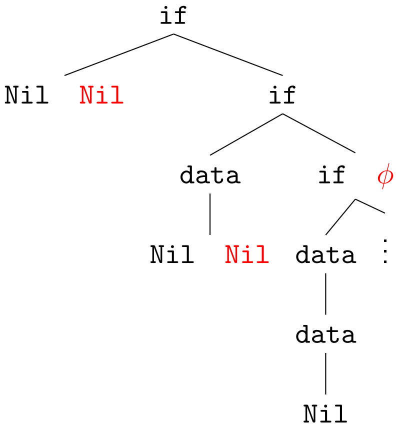
Idea: the formula "starts" on the root



where $\phi = \nu X. (\texttt{if} \wedge \diamond_1 (\mu Y. (\texttt{Nil} \vee \Box Y)) \wedge \diamond_2 X)$

Alternating parity tree automata

Idea: the formula "starts" on the root



where $\phi \ = \ \nu X. \ (\ \texttt{if} \wedge \diamond_1 \ (\ \mu Y. \ (\ \texttt{Nil} \vee \Box Y \ ) \ ) \wedge \diamond_2 \ X \ )$

Alternating parity tree automata

Conversion to an automaton ?

- Needs to play the formula over the tree, but always by reading a letter.
- Idea: iterate the formula several times until you find a letter.
- Needs non-determinism for $\vee$ and alternation for $\wedge$
- Needs a parity condition for distinguishing μ and ν

Alternating parity tree automata

Conversion to an automaton ?

- Needs to play the formula over the tree, but always by reading a letter.
- Idea: iterate the formula several times until you find a letter.
- Needs non-determinism for $\vee$ and alternation for $\wedge$
- Needs a parity condition for distinguishing μ and ν

Alternating parity tree automata

Conversion to an automaton ?

- Needs to play the formula over the tree, but always by reading a letter.
- Idea: iterate the formula several times until you find a letter.
- Needs non-determinism for $\vee$ and alternation for $\wedge$
- Needs a parity condition for distinguishing μ and ν

Alternating parity tree automata

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

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

This is reminiscent of the exponential modality of linear logic

So, in the sequel, we shall interpret recursion schemes in suitable domain-theoretic models of linear logic.

Alternating parity tree automata

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

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

This is reminiscent of the exponential modality of linear logic

So, in the sequel, we shall interpret recursion schemes in suitable domain-theoretic models of linear logic.

Alternating parity tree automata

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

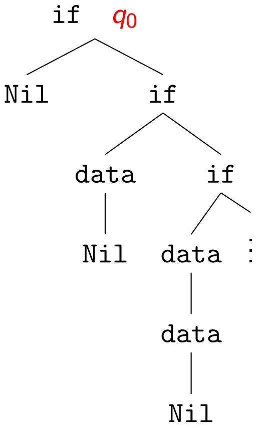
Example: $\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$.

This is reminiscent of the exponential modality of linear logic

So, in the sequel, we shall interpret recursion schemes in suitable domain-theoretic models of linear logic.

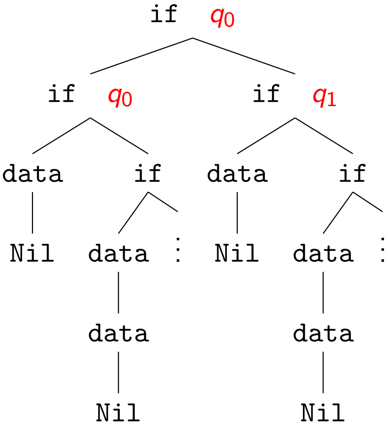
Alternating parity tree automata

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



Alternating parity tree automata

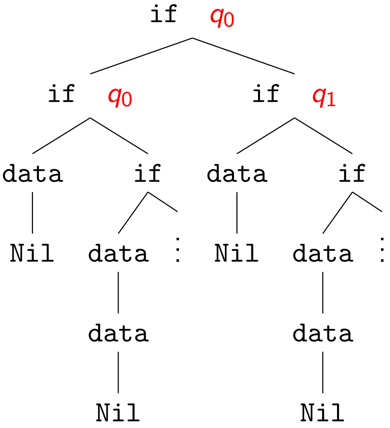
$\delta(q_0, \texttt{if}) = (2, q_0) \wedge (2, q_1).$



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

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

And for the inductive/coinductive behaviour ?

$\rightarrow$ parity conditions

Over a branch of a run-tree, say q_0 has colour 0 and q_1 has colour 1.

Now consider an infinite branch, and the maximal colour you see infinitely often on this branch.

If it is even, accept: it means you looped infinitely on ν.

Else if it is odd the automaton rejects: it means μ was unfolded infinitely, and this is forbidden.

Alternating parity tree automata

And for the inductive/coinductive behaviour ?

→ parity conditions

Over a branch of a run-tree, say q_0 has colour 0 and q_1 has colour 1.

Now consider an infinite branch, and the maximal colour you see infinitely often on this branch.

If it is even, accept: it means you looped infinitely on ν.

Else if it is odd the automaton rejects: it means μ was unfolded infinitely, and this is forbidden.

Alternating parity tree automata

And for the inductive/coinductive behaviour ?

$\rightarrow$ parity conditions

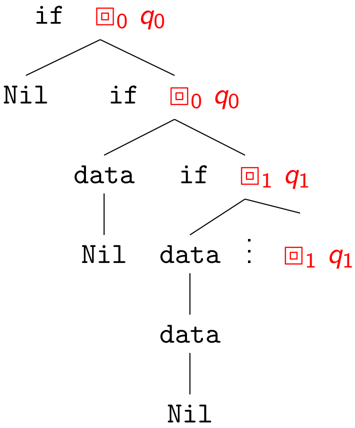
Over a branch of a run-tree, say q_0 has colour 0 and q_1 has colour 1.

Now consider an infinite branch, and the maximal colour you see infinitely often on this branch.

If it is even, accept: it means you looped infinitely on ν.

Else if it is odd the automaton rejects: it means μ was unfolded infinitely, and this is forbidden.

Parity condition on an example



would not be a winning run-tree: the automaton unfolded μ infinitely on the infinite branch (note: δ needs to be modified a little to produce this run-tree).

Alternating parity tree automata

In general, every state is given a colour, and a run-tree is accepting if and only if all its branches have an even maximal infinitely seen colour.

A tree is accepted iff it admits a winning run-tree. This is equivalent to satisfying the modal μ-calculus property encoded by the automaton.

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.

This is a way to prove the decidability of the higher-order model-checking problem.

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.

This is a way to prove the decidability of the higher-order model-checking problem.

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

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

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

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

A type-system for verification: without colours

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

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

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

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

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

A type-system for verification: 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 $$\frac{\Delta \vdash t : (\ \theta_1 \ \wedge \cdots \wedge \ \theta_k) \rightarrow \theta :: \kappa \rightarrow \kappa' \quad \Delta_i \vdash u : \theta_i :: \kappa}{\Delta + \Delta_1 + \ldots + \Delta_k \ \vdash \ t\, u : \theta :: \kappa'}$$

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

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

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

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

A type-system for verification

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.

Typings may be understood as the construction of denotations in appropriate models of linear logic.

A type-system for verification

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.

Typings may be understood as the construction of denotations in appropriate models of linear logic.

Linear decomposition of the intuitionnistic arrow

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

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

In other terms, in a model of linear logic, a program of type $A \Rightarrow B$ is interpreted as a replication of its inputs, followed by a linear use of them.

Linear decomposition of the intuitionnistic arrow

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

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

In other terms, in a model of linear logic, a program of type $A \Rightarrow B$ is interpreted as a replication of its inputs, followed by a linear use of them.

Models of linear logic

There are two main classes of 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.

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 – their are bigger, yet simpler.

They correspond to non-idempotent intersection types.

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 – their are bigger, yet simpler.

They correspond to non-idempotent intersection types.

Relational model of linear logic

Consider a relational model where

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

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

As a consequence,

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

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

Relational model of linear logic

Consider a relational model where

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

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

As a consequence,

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

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

Relational model of linear logic

Consider a relational model where

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

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

As a consequence,

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

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

Intersection types and relational interpretations

Consider again the typing

$$a \,:\, (q_0 \wedge q_1) \rightarrow q_2 \rightarrow q \,::\, \perp \rightarrow \perp \rightarrow \perp$$

In the relational model:

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

and this example translates as

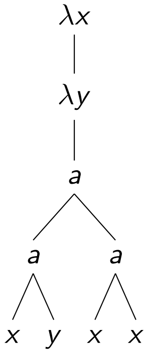
$$([q_0, q_1], ([q_2], q)) \in [\![a]\!]$$

An example of interpretation

Consider the rule

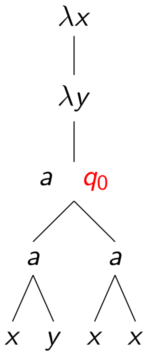
$$F\ x\ y\ =\ a\,(\,a\ x\ y\,)\,(\,a\ x\ x\,)$$

which corresponds to



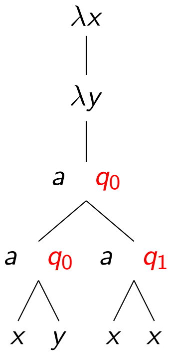
An example of interpretation

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

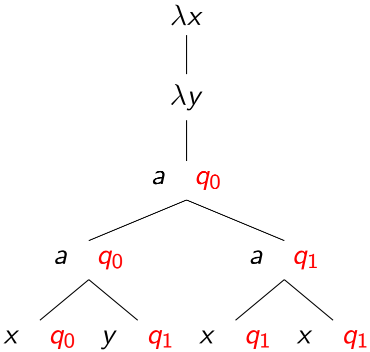


An example of interpretation

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



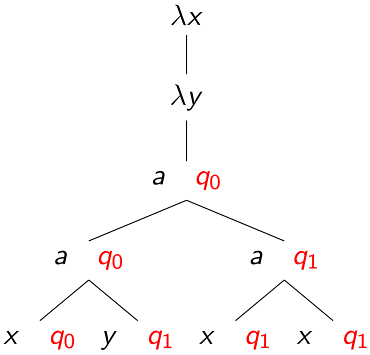
An example of interpretation



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

A tree over a ranked alphabet $\Sigma = \{a_1 : i_1, \cdots, a_n : i_n\}$ is interpreted as a λ-term

$$\lambda a_1 \cdots \lambda a_n . \, t$$

with $t :: \perp$ in normal form.

This is the Girard-Reynolds interpretation of trees.

So, in the model, a term building a Σ-tree is interpreted as a subset of

$$\mathcal{M}_{fin}([\![a_1]\!]) \times \cdots \times \mathcal{M}_{fin}([\![a_n]\!]) \times Q$$

Relational interpretation and automata acceptance

A tree over a ranked alphabet $\Sigma = \{a_1 : i_1, \cdots, a_n : i_n\}$ is interpreted as a λ-term

$$\lambda a_1 \cdots \lambda a_n . t$$

with $t :: \perp$ in normal form.

This is the Girard-Reynolds interpretation of trees.

So, in the model, a term building a Σ-tree is interpreted as a subset of

$$\mathcal{M}_{fin}([\![a_1]\!]) \times \cdots \times \mathcal{M}_{fin}([\![a_n]\!]) \times Q$$

Relational interpretation and automata acceptance

Theorem (G.-Melliès 2014)

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

Then $\mathcal{A}$ has a run-tree over T if and only if there exists $\alpha \subseteq [\![\delta]\!]$ such that

$$\alpha \times \{q_0\} \subseteq [\![t]\!]$$

The interpretation $[\![\delta]\!]$ of the transition function is defined as expected.

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. . .

Recursion: the *fix* rule

This model lacks recursion, and can not interpret in general the rule

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

Since schemes produce infinite trees, we need to shift to an infinitary variant of *Rel*.

We set:

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

Recursion: the *fix* rule

A function may now have a countable number of inputs.

This model has a coinductive fixpoint. The Theorem then extends:

Theorem (G.-Melliès 2014)

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

Then $\mathcal{A}$ has a run-tree over T if and only if there exists $\alpha \subseteq [\![\delta]\!]$ such that

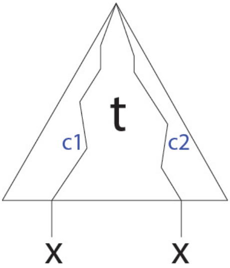
$$\alpha \times \{q_0\} \subseteq [\![t]\!]$$

Parity conditions

Kobayashi and Ong extended the typing with a colouring operation:

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

This operation lifts to higher-order.



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

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

Axiom $$\frac{}{x : \bigwedge_{\{i\}} \boxdot_{-1}\ \theta_i :: \kappa \ \vdash\ x : \theta_i :: \kappa}$$

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

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

fix $$\frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \boxdot_{-1}\ \theta :: \kappa \vdash F : \theta :: \kappa}$$

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

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

Axiom $$\frac{}{x \,:\, \bigwedge_{\{i\}} \boxdot_{-1}\, \theta_i \,::\, \kappa \;\vdash\; x \,:\, \theta_i \,::\, \kappa}$$

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

fix $$\frac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{F \,:\, \boxdot_{-1}\, \theta \,::\, \kappa \vdash F \,:\, \theta \,::\, \kappa}$$

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

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

Axiom $$\frac{}{x \,:\, \bigwedge_{\{i\}} \boxdot_{-1}\, \theta_i \,::\, \kappa \;\vdash\; x \,:\, \theta_i \,::\, \kappa}$$

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

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

fix $$\frac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{F \,:\, \boxdot_{-1}\, \theta \,::\, \kappa \vdash F \,:\, \theta \,::\, \kappa}$$

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

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

Axiom $\dfrac{}{x : \bigwedge_{\{i\}} \boxdot_{-1} \theta_i :: \kappa \vdash x : \theta_i :: \kappa}$

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

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

$$\mathit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \boxdot_{-1} \theta :: \kappa \vdash F : \theta :: \kappa}$$

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

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

Axiom $$\frac{}{x : \bigwedge_{\{i\}} \boxdot_{-1}\, \theta_i :: \kappa \;\vdash\; x : \theta_i :: \kappa}$$

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

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

fix $$\frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \boxdot_{-1}\, \theta :: \kappa \vdash F : \theta :: \kappa}$$

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

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.

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

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

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

This reformulation comes from a game semantics perspective.

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.

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 $\mathcal{G}$.

This reformulation comes from a game semantics perspective.

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.

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

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

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

This reformulation comes from a game semantics perspective.

Parity conditions

We investigated the semantic nature of $\Box$, and proved that it has good properties — it is a parameterized comonad, which distributes over the exponential.

It can be added to the model by setting

$$[\![\Box A]\!] \quad = \quad Col \times [\![A]\!]$$

and there is a very natural coloured interpretation of types:

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

Again, there is a correspondence between interpretations in the model and typings.

Parity conditions

We investigated the semantic nature of $\Box$, and proved that it has good properties — it is a parameterized comonad, which distributes over the exponential.

It can be added to the model by setting

$$[\![\Box A]\!] \quad = \quad Col \times [\![A]\!]$$

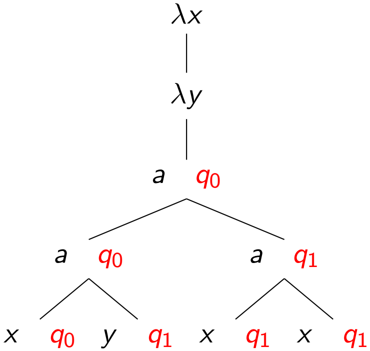
and there is a very natural coloured interpretation of types:

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

Again, there is a correspondence between interpretations in the model and typings.

An example of coloured interpretation

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

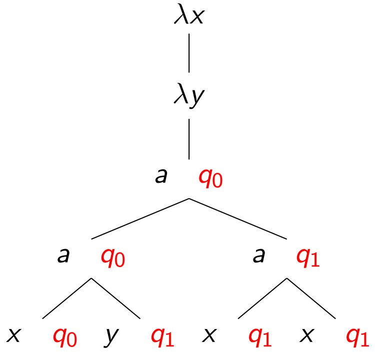


This rule will be interpreted in the model as

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

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 corresponds to 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 corresponds to 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 (Grellois-Melliès)

An APT $\mathcal{A}$ has a winning execution tree from state q_0 over the value tree $[\![\mathcal{G}]\!]$ of a HORS $\mathcal{G}$ if and only if $q_0 \in [\![\, t(\mathcal{G}) \,]\!]_{rel}$.

Extensional collapses

If the exponential modality ! is interpreted with finite sets, we obtain the poset-based model of linear logic.

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

Basically, the Scott model of linear logic is a qualitative model in which $!A$ is interpreted as $\mathcal{P}_{fin}(A)$. 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

If the exponential modality ! is interpreted with finite sets, we obtain the poset-based model of linear logic.

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

Basically, the Scott model of linear logic is a qualitative model in which $!A$ is interpreted as $\mathcal{P}_{fin}(A)$. 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,
2. It features a parametric comonad $\Box$, which propagates the colouring information of the APT in the denotations,
3. There is a distributive law $\lambda : \lightning \Box \rightarrow \Box \lightning$, 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,
4. There is a coloured parameterized fixed point operator Y which extends this cartesian closed category to a model of the λY-calculus.

Denotations, type-theoretically

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

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

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

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

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

Denotations, type-theoretically

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

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

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

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

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

Denotations, type-theoretically

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

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

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

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

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

Denotations, type-theoretically

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

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

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

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

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

Denotations, type-theoretically

The fixpoint rule

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

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

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

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

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

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

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

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

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

An APT $\mathcal{A}$ has a winning execution tree of initial state q_0 over the value tree $[\![\mathcal{G}]\!]$ of a HORS $\mathcal{G}$ if and only if its interpretation in the coloured Scott semantics with fixpoint $[\![\,\mathcal{G}\,]\!]_{fin}$ contains q_0.

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.

Conclusions and perspectives

- We studied domains-based models of linear logic designed to reflect the behaviour of alternating parity tree automata, in order to interpret λY-terms.
- In the relational case, our approach is reflected by a logic (coloured ILL) which also gives a type system equivalent to the one of Kobayashi and Ong.
- In the finitary case, our approach is reflected by a type system equivalent to the one of Kobayashi and Ong (is there a qualitative version of ILL ?).
- We obtain new proofs of decidability using the finitary semantics.
- 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 domains-based models of linear logic designed to reflect the behaviour of alternating parity tree automata, in order to interpret λY-terms.
- In the relational case, our approach is reflected by a logic (coloured ILL) which also gives a type system equivalent to the one of Kobayashi and Ong.
- In the finitary case, our approach is reflected by a type system equivalent to the one of Kobayashi and Ong (is there a qualitative version of ILL ?).
- We obtain new proofs of decidability using the finitary semantics.
- 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 domains-based models of linear logic designed to reflect the behaviour of alternating parity tree automata, in order to interpret λY-terms.
- In the relational case, our approach is reflected by a logic (coloured ILL) which also gives a type system equivalent to the one of Kobayashi and Ong.
- In the finitary case, our approach is reflected by a type system equivalent to the one of Kobayashi and Ong (is there a qualitative version of ILL ?).
- We obtain new proofs of decidability using the finitary semantics.
- 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 domains-based models of linear logic designed to reflect the behaviour of alternating parity tree automata, in order to interpret λY-terms.
- In the relational case, our approach is reflected by a logic (coloured ILL) which also gives a type system equivalent to the one of Kobayashi and Ong.
- In the finitary case, our approach is reflected by a type system equivalent to the one of Kobayashi and Ong (is there a qualitative version of ILL ?).
- We obtain new proofs of decidability using the finitary semantics.
- 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 domains-based models of linear logic designed to reflect the behaviour of alternating parity tree automata, in order to interpret λY-terms.
- In the relational case, our approach is reflected by a logic (coloured ILL) which also gives a type system equivalent to the one of Kobayashi and Ong.
- In the finitary case, our approach is reflected by a type system equivalent to the one of Kobayashi and Ong (is there a qualitative version of ILL ?).
- We obtain new proofs of decidability using the finitary semantics.
- 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 . . .