Type systems and logical models for higher-order verification

Charles Grellois (joint work with Paul-André Melliès)

PPS & LIAFA — Université Paris 7

November 24th, 2014

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.
q_1 means a *read* was seen.

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.
q_1 means a *read* was seen.

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.
q_1 means a *read* was seen.

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

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

This means that *read*, when catenated to the left of a word accepting from q_1, results in a word accepting from q_0.

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

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

This means that *read*, when catenated to the left of a word accepting from q_1, results in a word accepting from q_0.

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

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

This means that *read*, when catenated to the left of a word accepting from q_1, results in a word accepting from q_0.

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

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

Note that the word is seen as a term.

A type-theoretic intuition: a run of the automaton

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

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

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

and so on.

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

or, equivalently, by typing the program with the initial state of the automaton.

Back to types

Typing naturally lifts to terms: from

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

we can deduce

$$\dfrac{\dfrac{\dfrac{\vdash open \;:\; q_0 \to q_0 \qquad x : q_0 \vdash x : q_0}{x : q_0 \vdash open\ x \;:\; q_0}}{\vdash \lambda x.\, open\ x \;:\; q_0 \to q_0} \qquad \dfrac{\pi}{\vdash \; (read \cdot write)^2 \cdot close \;:\; q_0}}{\vdash (\lambda x.\, open\ x) \; \left((read \cdot write)^2 \cdot close\right) \;:\; q_0}$$

Back to types

Typing naturally lifts to terms: from

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

we can deduce

$$\dfrac{\dfrac{\dfrac{\vdash open \ : \ q_0 \to q_0 \qquad x \ : \ q_0 \vdash x \ : \ q_0}{x \ : \ q_0 \vdash open \ x \ : q_0}}{\vdash \lambda x.\, open \ x \ : \ q_0 \to q_0} \qquad \dfrac{\pi}{\vdash \ (read \cdot write)^2 \cdot close \ : \ q_0}}{\vdash (\lambda x.\, open \ x) \ \left((read \cdot write)^2 \cdot close\right) \ : \ q_0}$$

Back to types

This is interesting, as it suggests to directly type programs in order to model-check them.

There is no need to reduce them prior to running automata, we can lift their behaviour to higher-order.

On the other hand, the monoid approach is not suited for that: could we replace it with a higher-order analogue ?

Logical models are good candidates.

Back to types

This is interesting, as it suggests to directly type programs in order to model-check them.

There is no need to reduce them prior to running automata, we can lift their behaviour to higher-order.

On the other hand, the monoid approach is not suited for that: could we replace it with a higher-order analogue ?

Logical models are good candidates.

Back to types

This is interesting, as it suggests to directly type programs in order to model-check them.

There is no need to reduce them prior to running automata, we can lift their behaviour to higher-order.

On the other hand, the monoid approach is not suited for that: could we replace it with a higher-order analogue ?

Logical models are good candidates.

Infinite words and Büchi conditions

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

Recall that a ultimately periodic word can be written $s \cdot t^{\omega}$.

On the type system, we deal with the $(\cdot)^{\omega}$ operation with a new rule:

$$fix \quad \frac{\vdash \; t \cdot t^{\omega} \; : \; q}{\vdash \; t^{\omega} \; : \; q}$$

Infinite words and Büchi conditions

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

Recall that a ultimately periodic word can be written $s \cdot t^{\omega}$.

On the type system, we deal with the $(\cdot)^{\omega}$ operation with a new rule:

$$fix \quad \frac{\vdash \; t \cdot t^{\omega} \; : \; q}{\vdash \; t^{\omega} \; : \; q}$$

Infinite words and Büchi conditions

$$fix \quad \frac{\vdash \ t \cdot t^{\omega} \ : \ q}{\vdash \ t^{\omega} \ : \ q}$$

This leads to infinite-depth derivations, over which we may transpose the Büchi condition, discriminating winning and loosing derivation trees.

The existence of a winning derivation tree will be equivalent to the existence of a successful run of a Büchi automaton over the normal form of the term.

Infinite words and Büchi conditions

$$fix \quad \frac{\vdash \; t \cdot t^{\omega} \; : \; q}{\vdash \; t^{\omega} \; : \; q}$$

This leads to infinite-depth derivations, over which we may transpose the Büchi condition, discriminating winning and loosing derivation trees.

The existence of a winning derivation tree will be equivalent to the existence of a successful run of a Büchi automaton over the normal form of the term.

Infinite words and Büchi conditions

In the monoid approach, we now need not only a good model for replacing the monoid, but also a good fixpoint operation.

This fixpoint should only produce denotations complying with the Büchi condition.

A last remark: in this work, the aim is that typing derivations reflect the computation of denotations in the model.

After this long prologue, let's move to higher-order verification !

Infinite words and Büchi conditions

In the monoid approach, we now need not only a good model for replacing the monoid, but also a good fixpoint operation.

This fixpoint should only produce denotations complying with the Büchi condition.

A last remark: in this work, the aim is that typing derivations reflect the computation of denotations in the model.

After this long prologue, let's move to higher-order verification !

Infinite words and Büchi conditions

In the monoid approach, we now need not only a good model for replacing the monoid, but also a good fixpoint operation.

This fixpoint should only produce denotations complying with the Büchi condition.

A last remark: in this work, the aim is that typing derivations reflect the computation of denotations in the model.

After this long prologue, let's move to higher-order verification !

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: $\texttt{compose}\ \phi\ x\ =\ \phi(\phi(x))$

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: $\texttt{compose}\ \phi\ x\ =\ \phi(\phi(x))$

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

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

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)

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

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

generates:

S

Value tree of a recursion scheme

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

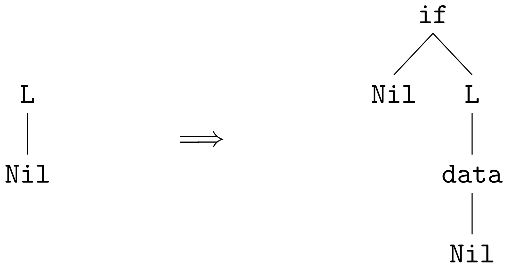
generates:

$\texttt{S} \quad \Longrightarrow$

Value tree of a recursion scheme

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

generates:

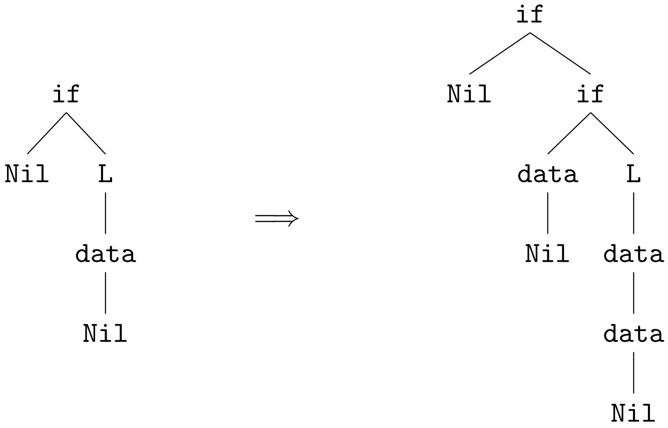


Notice that substitution and expansion occur in one same step.

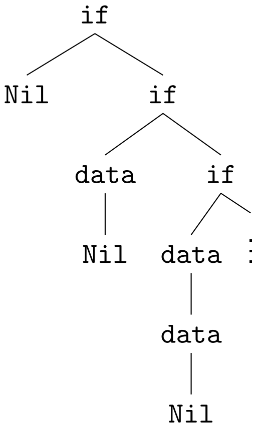
Value tree of a recursion scheme

$$\mathtt{S} = \mathtt{L\ Nil}$$
$$\mathtt{L}\ x = \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ ))$$

generates:

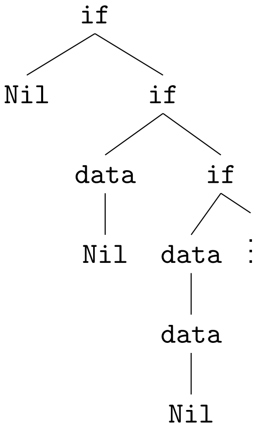


Value tree of a recursion scheme



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.

So, it is the λY-term itself which we should use for verification.

Our aim will thus be to extend the ideas of the prologue to our current setting, so that the interpretation of the term in a suitable domain would reflect the automaton's behaviour on its infinite, non-regular normal form.

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.

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

Dually, $\nu X.\, \phi$ 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:

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

mean ?

Interaction with trees: a shift to automata theory

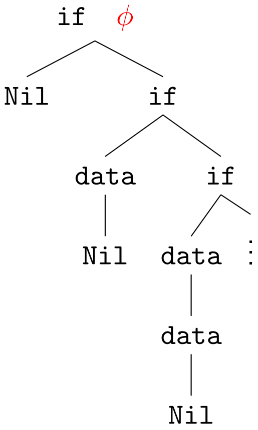
Logic is great !

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

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

Alternating parity tree automata

Idea: the formula "starts" on the root

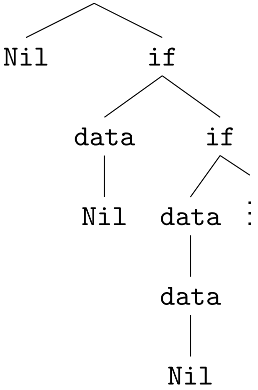


where $\phi = \nu X. \, (\, \texttt{if} \wedge \diamond_1 \, (\, \mu Y. \, (\, \texttt{Nil} \vee \Box Y \,) \,) \wedge \diamond_2 \, X \,)$

Alternating parity tree automata

Idea: the formula "starts" on the root

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

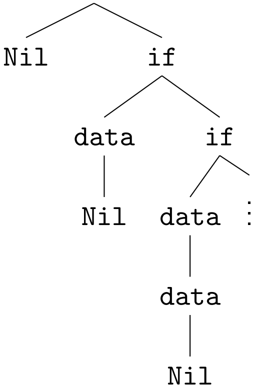


where $\phi \;=\; \nu X. \, (\, \texttt{if} \wedge \diamond_1 \; (\, \mu Y. \, (\, \texttt{Nil} \vee \Box Y \,)\,) \wedge \diamond_2 \; X \,)$

Alternating parity tree automata

Idea: the formula "starts" on the root

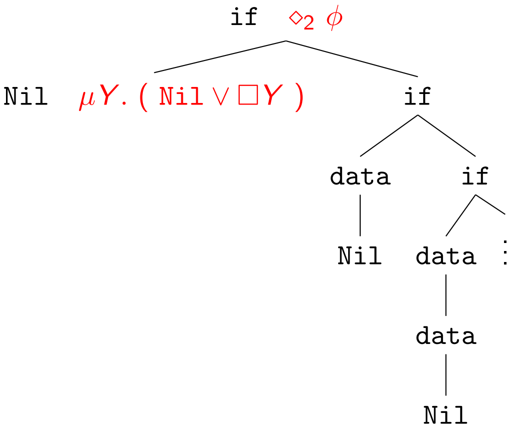
if $\quad \diamond_1 \; (\; \mu Y. \; (\; \texttt{Nil} \vee \Box Y \;)\;) \wedge \diamond_2 \; \phi$



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

Alternating parity tree automata

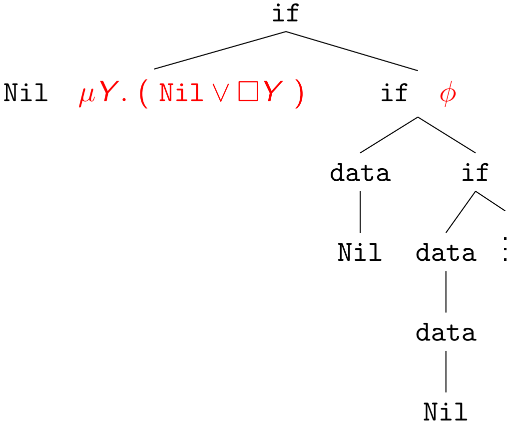
Idea: the formula "starts" on the root



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

Alternating parity tree automata

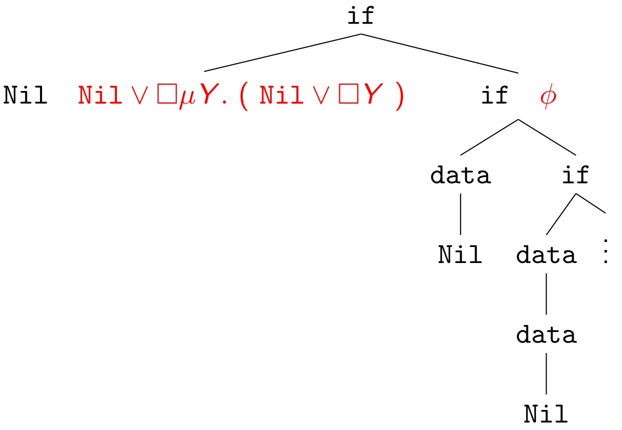
Idea: the formula "starts" on the root



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

Alternating parity tree automata

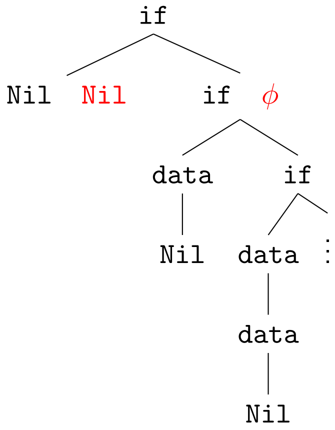
Idea: the formula "starts" on the root



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

Alternating parity tree automata

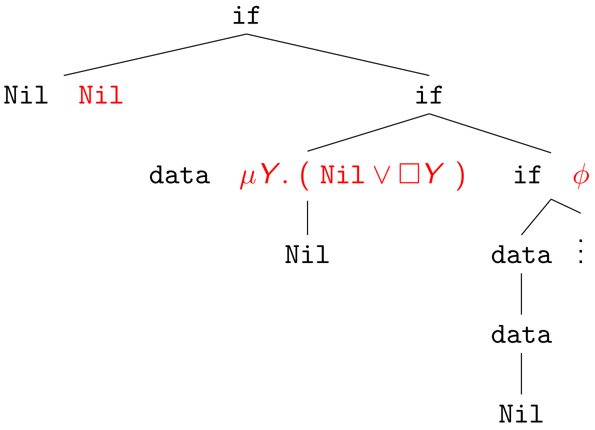
Idea: the formula "starts" on the root



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

Alternating parity tree automata

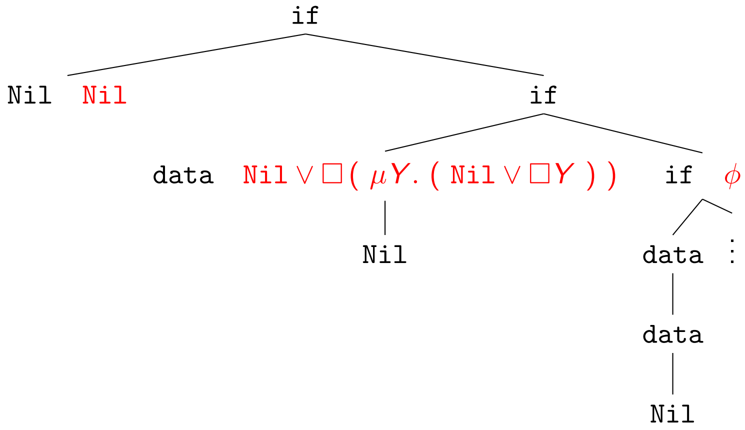
Idea: the formula "starts" on the root



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

Alternating parity tree automata

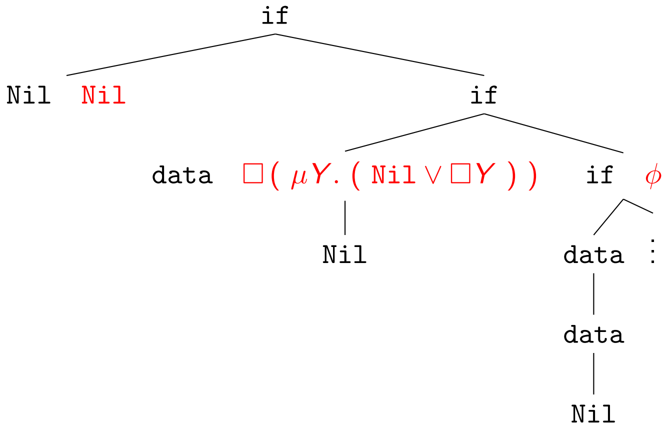
Idea: the formula "starts" on the root



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

Alternating parity tree automata

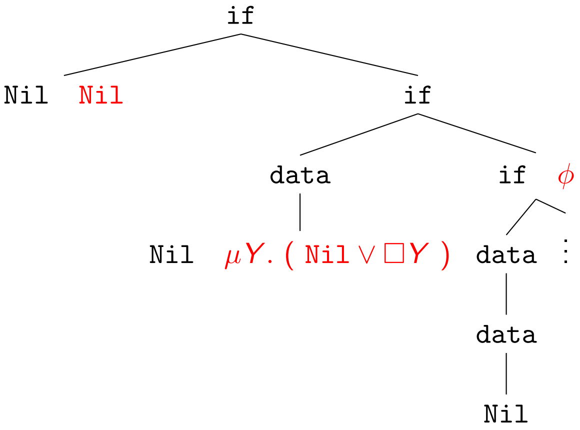
Idea: the formula "starts" on the root



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

Alternating parity tree automata

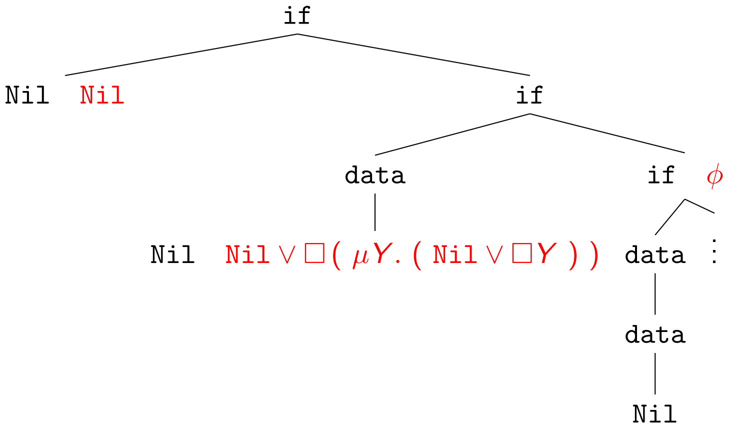
Idea: the formula "starts" on the root



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

Alternating parity tree automata

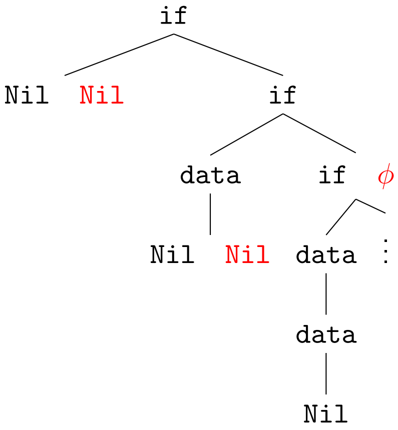
Idea: the formula "starts" on the root



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

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.

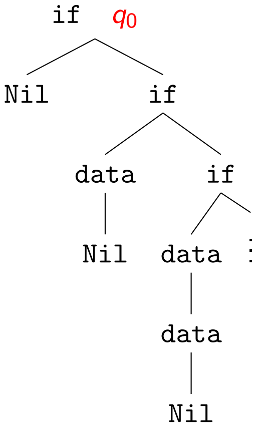
Example: $\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$.

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.

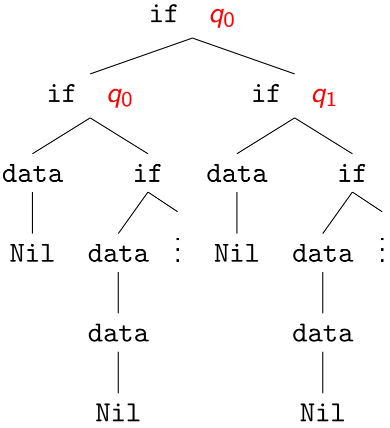
Alternating parity tree automata

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



Alternating parity tree automata

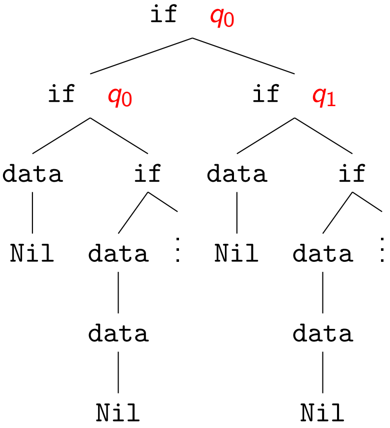
$\delta(q_0, \texttt{if}) = (2, q_0) \wedge (2, q_1).$



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 ?

We introduce 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 ?

We introduce parity conditions.

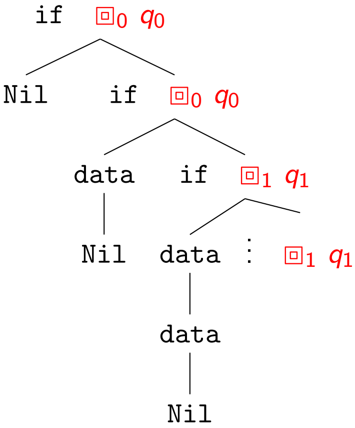
Over a branch of a run-tree, say q_0 has colour 0 and q_1 has colour 1.

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.

Alternating parity tree automata

In general, every state is given a colour, and a run-tree is winning if and only if all of 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.

In a sense, run-trees are coinductive (they treat μ as ν), then the parity condition selects *a posteriori* the ones complying with the inductive μ policy.

Alternating parity tree automata

In general, every state is given a colour, and a run-tree is winning if and only if all of 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.

In a sense, run-trees are coinductive (they treat μ as ν), then the parity condition selects *a posteriori* the ones complying with the inductive μ policy.

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

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

Typing lifts to higher-order, and reflects the behaviour of the APT directly over the finite representation of the program as a HORS.

This lifting to a finite representation is the key to decidability.

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

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

Typing lifts to higher-order, and reflects the behaviour of the APT directly over the finite representation of the program as a HORS.

This lifting to a finite representation is the key to decidability.

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

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

Typing lifts to higher-order, and reflects the behaviour of the APT directly over the finite representation of the program as a HORS.

This lifting to a finite representation is the key to decidability.

A type-system for verification: without colours

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

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

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

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

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

A type-system for verification: without colours

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

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

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: without colours

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

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

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

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

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

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

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

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

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

For this work, we just need one fact from linear logic:

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

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

In other terms, it consists in a replication of arguments and then in a linear use of them.

Linear decomposition of the intuitionnistic arrow

For this work, we just need one fact from linear logic:

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

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

In other terms, it consists in a replication of arguments and then in a linear use of them.

Linear decomposition of the intuitionnistic arrow

For this work, we just need one fact from linear logic:

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

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

In other terms, it consists in a replication of arguments and then in 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

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

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

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

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

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

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 $[\![B]\!]$.

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

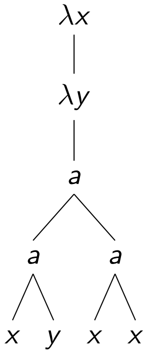
$$([q_0, q_1], ([q_2], q)) \in [\![a]\!]$$

An example of interpretation

Consider the rule

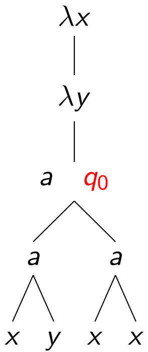
$$F\ x\ y\ =\ a\,(\,a\ x\ y\,)\,(\,a\ x\ x\,)$$

which corresponds to



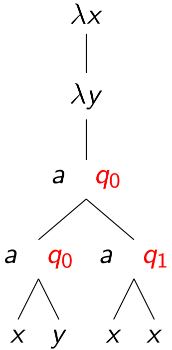
An example of interpretation

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

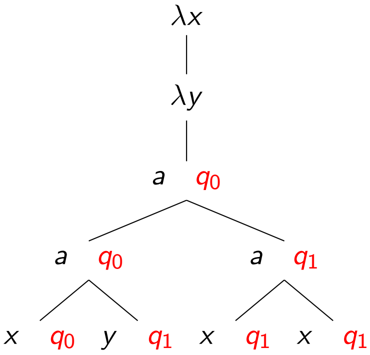


An example of interpretation

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



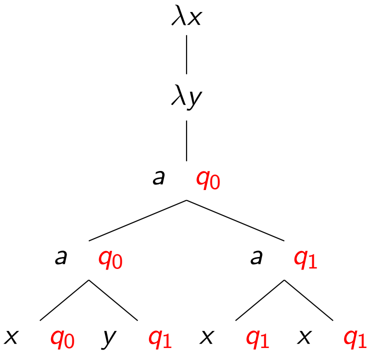
An example of interpretation



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 $t :: \bot$ 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 infinite recursion, and can not interpret in general the rule

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

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

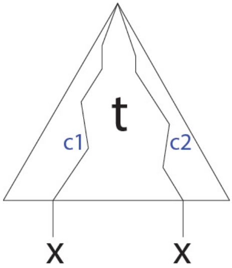
$$\alpha \times \{q_0\} \subseteq [\![t]\!]$$

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

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 $\dfrac{}{x \,:\, \bigwedge_{\{i\}} \boxdot_1 \, \theta_i \,::\, \kappa \;\vdash\; x \,:\, \theta_i \,::\, \kappa}$

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

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

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

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

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 $$\dfrac{}{x \,:\, \bigwedge_{\{i\}} \Box_1\, \theta_i \,::\, \kappa \;\vdash\; x \,:\, \theta_i \,::\, \kappa}$$

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

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

fix $$\dfrac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{F \,:\, \Box_1\, \theta \,::\, \kappa \;\vdash\; F \,:\, \theta \,::\, \kappa}$$

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

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

Axiom $$\frac{}{x : \bigwedge_{\{i\}} \Box_1 \theta_i :: \kappa \ \vdash \ x : \theta_i :: \kappa}$$

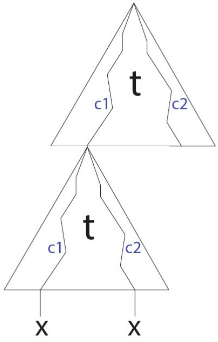
δ $$\frac{\{ (i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i \} \quad \text{satisfies} \quad \delta_A(q, a)}{\emptyset \vdash a : \bigwedge_{j=1}^{k_1} \Box_{m_{1j}} q_{1j} \rightarrow \ldots \rightarrow \bigwedge_{j=1}^{k_n} \Box_{m_{nj}} q_{nj} \rightarrow q :: \bot \rightarrow \cdots \rightarrow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

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

fix $$\frac{\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa}{F : \Box_1 \theta :: \kappa \vdash F : \theta :: \kappa}$$

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

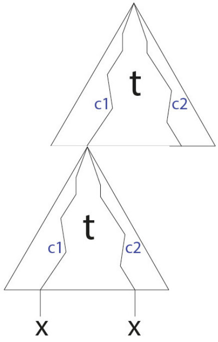
A type-system for verification: the App rule



The colouring modality updates the colours of the variables.

A key fact of this talk: parity conditions are invariant under β-reduction and β-expansion.

A type-system for verification: the App rule



The colouring modality updates the colours of the variables.

A key fact of this talk: parity conditions are invariant under β-reduction and β-expansion.

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 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 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 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.

This gives a new exponential $\lightning$.

In the model, setting

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

we obtain a very natural coloured interpretation of types:

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

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.

This gives a new exponential $\lightning$.

In the model, setting

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

we obtain a very natural coloured interpretation of types:

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

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.

This gives a new exponential $\lightning$.

In the model, setting

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

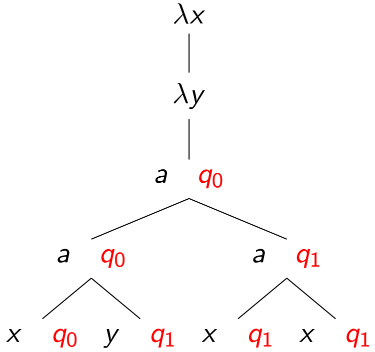
we obtain a very natural coloured interpretation of types:

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

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

An example of coloured interpretation

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

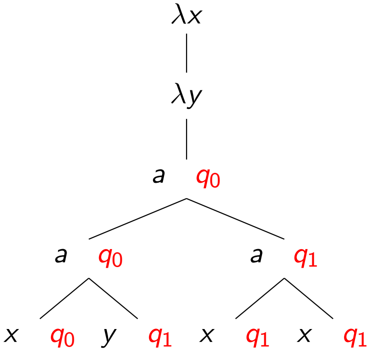


This rule will be interpreted in the model as

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

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.

It can be defined as an operator $\mathbf{Y}$ which transports a relation

$$f \quad : \quad \lightning X \otimes \lightning A \multimap A$$

to a relation

$$\mathbf{Y}_{X,A}(f) \quad : \quad \lightning X \multimap A.$$

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.

It can be defined as an operator **Y** which transports a relation

$$f \quad : \quad \lightning X \otimes \lightning A \ \multimap \ A$$

to a relation

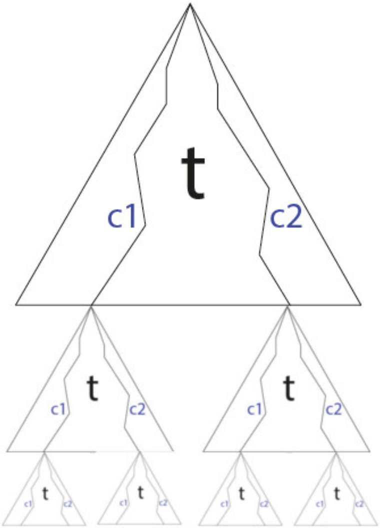
$$\mathbf{Y}_{X,A}(f) \quad : \quad \lightning X \ \multimap \ A.$$

Connection with the coloured relational model

Elements of $\mathbf{Y}_{X,A}(f)$ are obtained from compositions of denotations of f.

The composition tree may be infinite; in this case, it has to be winning for the parity condition.

Connection with the coloured relational model



Connection with the coloured relational model

Note that the fixpoint operator **Y** may be understood as defined from the inductive and coinductive operators over this infinitary relational model.

We obtain the general Theorem:

Theorem (G.-Melliès 2014)

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

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

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

where the interpretation is computed in the coloured relational model.

Connection with the coloured relational model

Note that the fixpoint operator **Y** may be understood as defined from the inductive and coinductive operators over this infinitary relational model.

We obtain the general Theorem:

Theorem (G.-Melliès 2014)

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

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

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

where the interpretation is computed in the coloured relational model.

A last remark: 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.

Melliès gave a version of the previous Theorem in a variant of this qualitative model.

We are currently adapting the extensional collapse theorem (in a type-theoretic version) to the infinitary and coloured settings, in order to relate these two version of the semantic model-checking theorem.

A last remark: 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.

Melliès gave a version of the previous Theorem in a variant of this qualitative model.

We are currently adapting the extensional collapse theorem (in a type-theoretic version) to the infinitary and coloured settings, in order to relate these two version of the semantic model-checking theorem.

A last remark: 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.

Melliès gave a version of the previous Theorem in a variant of this qualitative model.

We are currently adapting the extensional collapse theorem (in a type-theoretic version) to the infinitary and coloured settings, in order to relate these two version of the semantic model-checking theorem.

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.
- Results of extensional collapse lead to new approaches for decidability.
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", game semantics with parity, extend 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.
- Results of extensional collapse lead to new approaches for decidability.
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", game semantics with parity, extend 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.
- Results of extensional collapse lead to new approaches for decidability.
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", game semantics with parity, extend 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.
- Results of extensional collapse lead to new approaches for decidability.
- There is still a lot to do: study the coloured extensional collapse, axiomatize this extension of "recognition by monoid", game semantics with parity, extend to other models of automata ...