Coloured indexed linear logic and higher-order model-checking

Charles Grellois Paul-André Melliès

PPS — Université Paris 7

September 10th, 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.

Automata and recognition

Consider a language $L \subseteq A^*$. Recall that

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.

The interpretation moreover characterizes the words of L, as a particular subset of the structure.

Automata and recognition

Consider a language $L \subseteq A^*$. Recall that

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.

The interpretation moreover characterizes the words of L, as a particular subset of the structure.

Model-checking higher-order programs

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

They will be modelled by recursion schemes, 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.

They will be modelled by recursion schemes, 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)
- current work of Salvati and Walukiewicz (interpretation in finite models)
- ...

Our aim is to deepen the semantic understanding we have of this result, using existing relations between 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

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

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)

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

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

generates:

$$S$$

Value tree of a recursion scheme

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

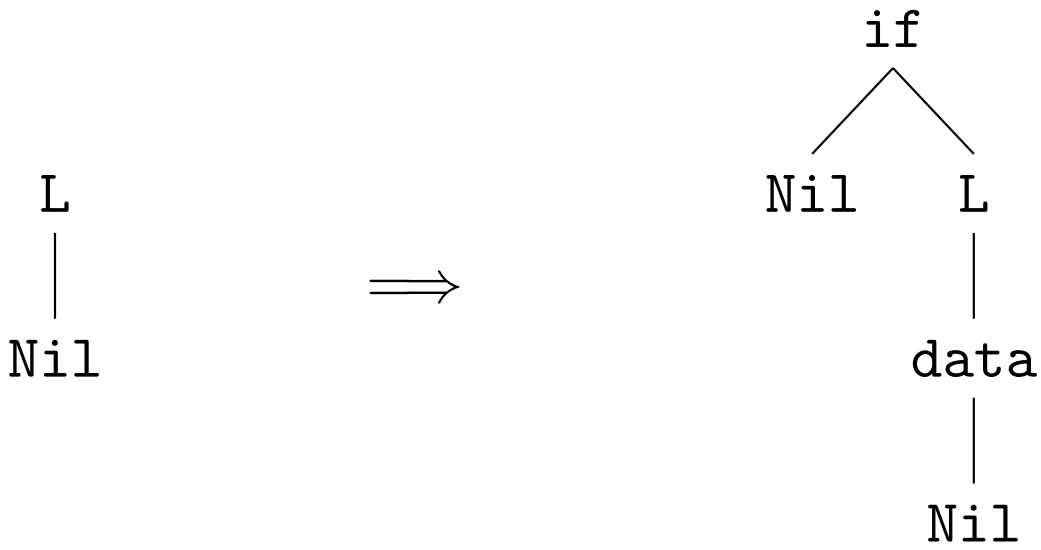
generates:

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

Value tree of a recursion scheme

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

generates:

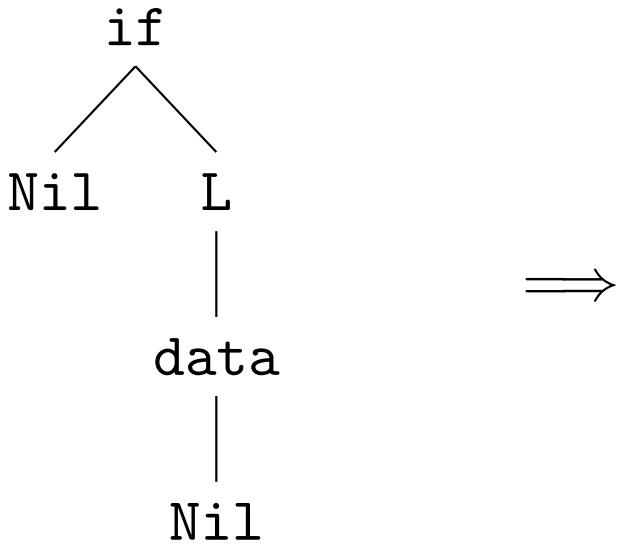


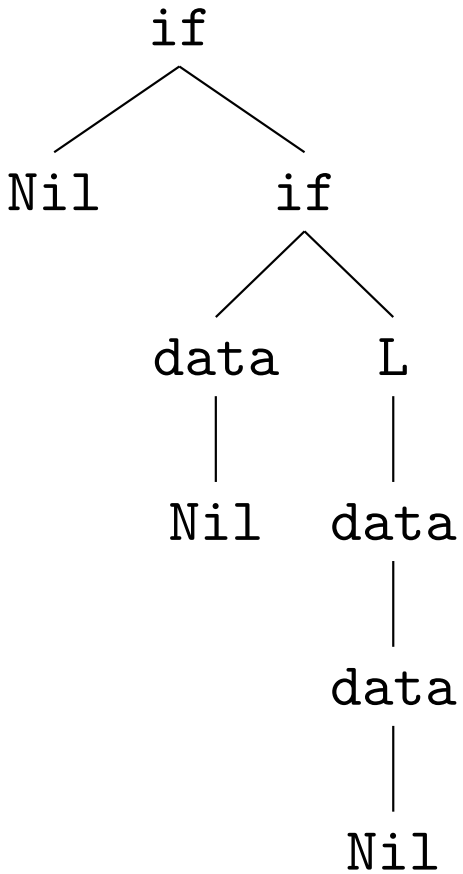
Notice that substitution and expansion occur in one same step.

Value tree of a recursion scheme

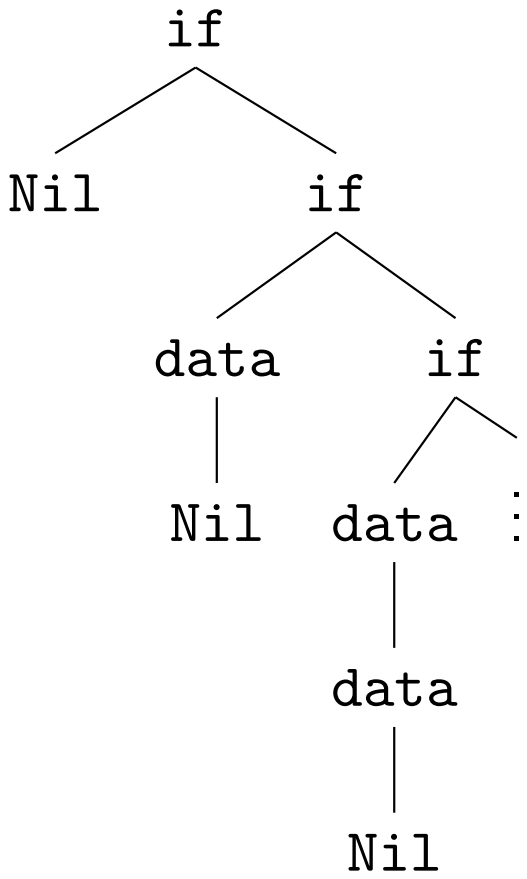
$$\begin{array}{lcl} \mathtt{S} & = & \mathtt{L\ Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x)) \end{array}$$

generates:



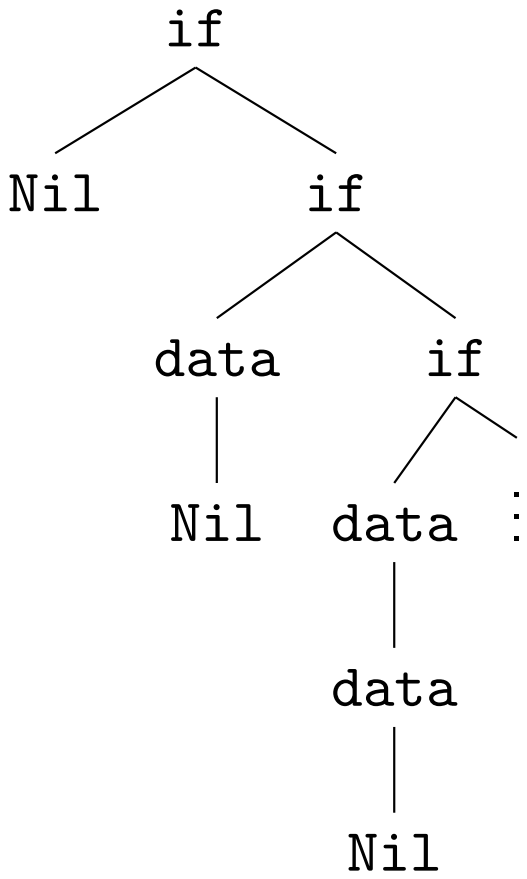


Value tree of a recursion scheme



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.

This suggests that we should interpret the associated λ-term in an algebraic structure suitable for higher-order interpretations: a domain.

Alternating parity tree automata

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

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

Its states are the subformulas of the encoded formula.

Alternating parity tree automata

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

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

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.

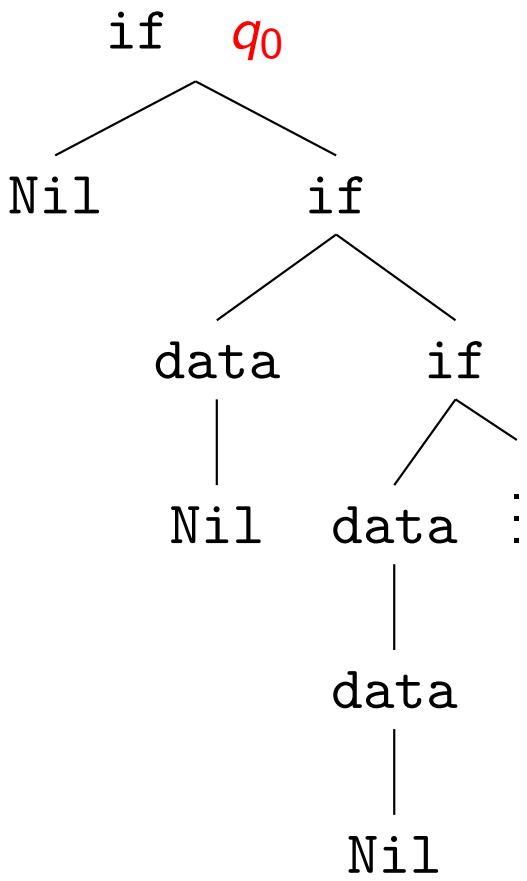
Example: $\delta(q_0, \texttt{if}) = (2, q_0) \wedge (2, q_1)$.

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.

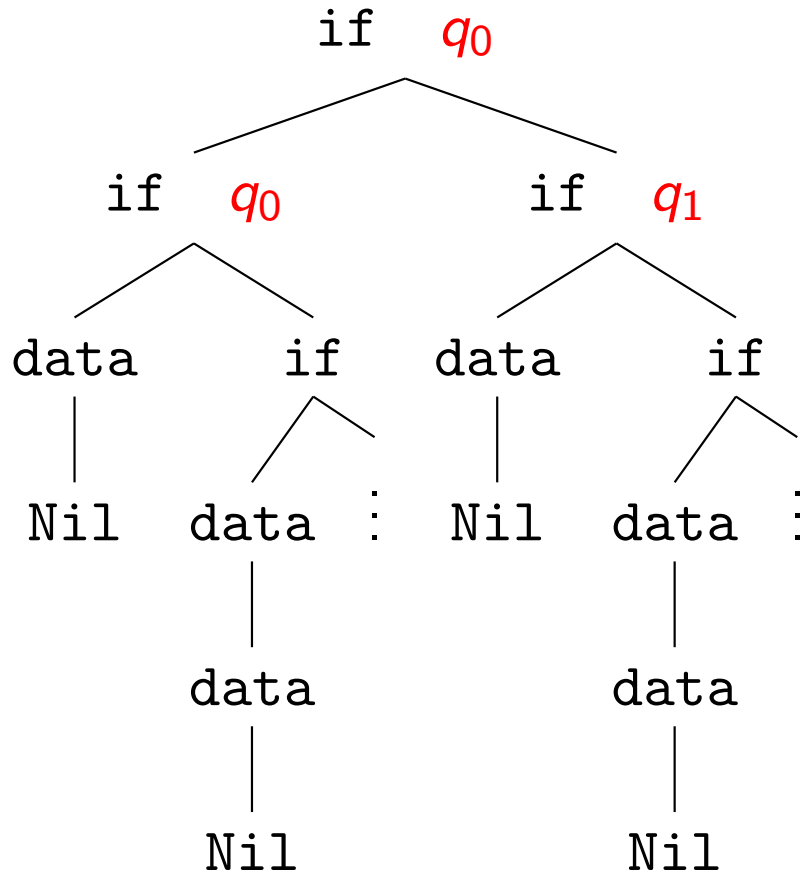
Alternating parity tree automata

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



Alternating parity tree automata

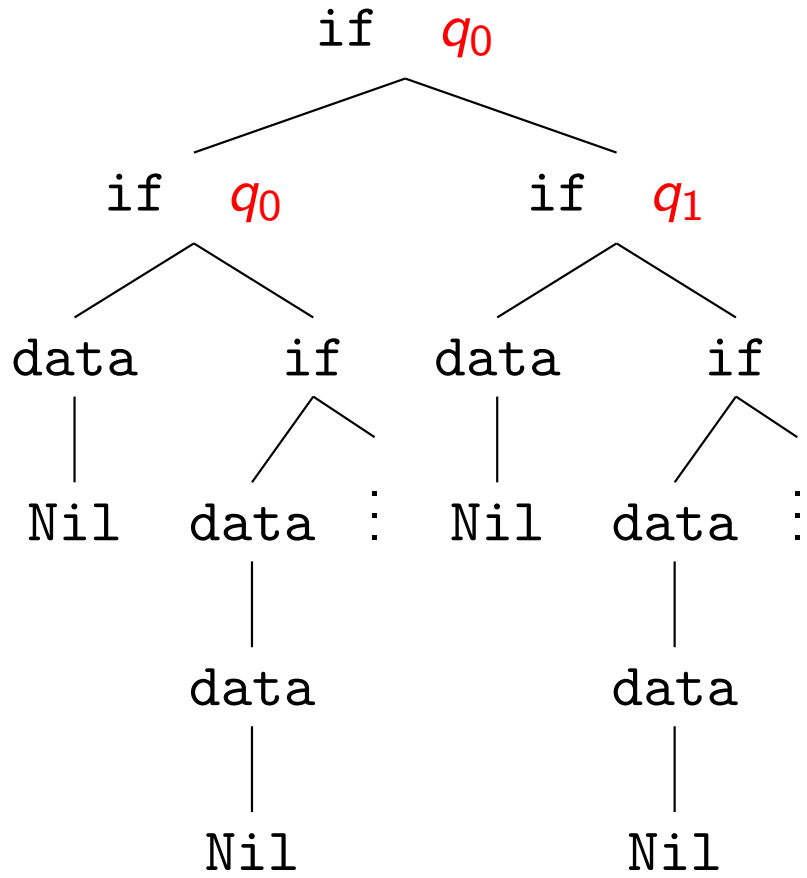
$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1).$



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.

Model-checking higher-order programs

Kobayashi noticed in 2009 that a transition

$$\delta(q, a) \;=\; (1, q_0) \wedge (1, q_1) \wedge (2, q_2)$$

may be understood as a refinement of the simple typing

$$a \;:\; \bot \to \bot \to \bot$$

with intersection types:

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

In this approach, every intersection type refines a simple type.

Model-checking higher-order programs

Kobayashi noticed in 2009 that a transition

$$\delta(q, a) \;=\; (1, q_0) \wedge (1, q_1) \wedge (2, q_2)$$

may be understood as a refinement of the simple typing

$$a \;:\; \bot \to \bot \to \bot$$

with intersection types:

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

In this approach, every intersection type refines a simple type.

Model-checking higher-order programs

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

This connects tree automata to higher-order computations.

In this way, the action of the APT over the infinitary, non-regular value tree of the scheme can be reflected in the finite denotation of its equivalent λY-term.

This is the core idea of the decidability result.

Model-checking higher-order programs

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

This connects tree automata to higher-order computations.

In this way, the action of the APT over the infinitary, non-regular value tree of the scheme can be reflected in the finite denotation of its equivalent λY-term.

This is the core idea of the decidability result.

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 (why ?) of elements of $[\![A]\!]$.

Linear decomposition of the intuitionnistic arrow

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

$$!A \multimap B$$

whose interpretation in this relational model is

$$\mathcal{M}_{fin}(\llbracket A \rrbracket) \times \llbracket B \rrbracket$$

In other words, it is some collection (with multiplicities) of elements of $\llbracket A \rrbracket$ producing an element of $\llbracket B \rrbracket$.

Linear decomposition of the intuitionnistic arrow

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

$$!A \multimap B$$

whose interpretation in this relational model is

$$\mathcal{M}_{fin}(\llbracket A \rrbracket) \times \llbracket B \rrbracket$$

In other words, it is some collection (with multiplicities) of elements of $\llbracket A \rrbracket$ producing an element of $\llbracket B \rrbracket$.

Intersection types and relational interpretations

Consider again the typing

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

In the relational model:

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

and this example translates as

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

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

$$\mathcal{M}_{fin}(\llbracket a_1 \rrbracket) \times \cdots \times \mathcal{M}_{fin}(\llbracket a_n \rrbracket) \times Q$$

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 $t :: \bot$ 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

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.

Indexed linear logic

The relational model contains strictly more than denotations of terms.

Actually, if a term uses its argument several times, nothing forbids to give the denotation of a different term of the appropriated type for each occurence.

The whole relational model corresponds to denotations of the lambda calculus with resources.

Indexed linear logic

The relational model contains strictly more than denotations of terms.

Actually, if a term uses its argument several times, nothing forbids to give the denotation of a different term of the appropriated type for each occurence.

The whole relational model corresponds to denotations of the lambda calculus with resources.

Indexed linear logic

Bucciarelli and Ehrhard characterized logically the fragment of the relational model corresponding to terms.

Intuitively, their idea is to modify linear logic so that it is forced to provide a proof term of the same shape for each element of a multiset.

Indexed linear logic

In this goal, proofs are parallelized.

Sequents are indexed by families I, J, $K \ldots$:

$$\Gamma \vdash_I t : \sigma_i :: A$$

This should be understood as the superposition of $|I|$ different typing proofs for a same term.

The proof of index i proves that $t : \sigma_i :: A$.

Indexed linear logic

In this goal, proofs are parallelized.

Sequents are indexed by families I, J, $K \ldots$:

$$\Gamma \vdash_I t : \sigma_i :: A$$

This should be understood as the superposition of $|I|$ different typing proofs for a same term.

The proof of index i proves that $t : \sigma_i :: A$.

Indexed linear logic

In the relational model, the exponential builds multisets.

In indexed linear logic, it should only build uniform multisets.

This is done with the Promotion rule

$$\frac{\ldots\ x_k\ :\ [\sigma_{i_k} \mid i_k \in I_k,\ u_k(i_k) = j]\ ::\ !_{u_k} A_k \ldots \vdash_J\ M\ :\ \tau_j\ ::\ B}{\ldots x_k\ :\ [\sigma_{i_k} \mid i_k \in I_k,\ v(u_k(i_k)) = l]\ ::\ !_{v \circ u_k} A_k \ldots \vdash_L\ M\ :\ [\tau_j \mid v(j) = l]\ ::\ !_v\ B}$$

where $v : J \to L$.

Indexed linear logic

How do we create uniform multisets ?

Consider $c : q_0 \wedge q_1 :: \bot$ in the quantitative system. We build it with the following derivation:

$$\dfrac{\dfrac{\dfrac{(q_1,\, q_2) \in Q^2}{c : q_j :: \bot_{\{1,2\}} \vdash_{j \in \{1,2\}} c : q_j :: \bot_{\{1,2\}}}}{c : [q_j] :: !_{id}\, \bot_{\{1,2\}} \vdash_{j \in \{1,2\}} c : q_j :: \bot_{\{1,2\}}}}{c : [q_1, q_2] :: !_{v \circ id}\, \bot_{\{1,2\}} \vdash_{\{1\}} c : [q_1, q_2] :: !_{v}\, \bot_{\{1,2\}}}$$

where v is the surjection $\{1, 2\} \to \{1\}$.

Indexed linear logic

In general, this structuration rule builds parallel families of (uniform) multisets.

$$\dfrac{\dfrac{\dfrac{(q_1,\, q_2, q_3) \in Q^2}{c \,:\, q_j \,::\, \bot_{\{1,2,3\}} \vdash_{j\in\{1,2,3\}} c \,:\, q_j \,::\, \bot_{\{1,2,3\}}}}{c \,:\, [q_j] \,::\, !_{id}\, \bot_{\{1,2,3\}} \vdash_{j\in\{1,2,3\}} c \,:\, q_j \,::\, \bot_{\{1,2,3\}}}}{c \,:\, [q_j \mid v(j) = i] \,::\, !_v\, \bot_{\{1,2,3\}} \vdash_{i\in\{1,2\}} c \,:\, [q_j \mid v(j) = i] \,::\, !_v\, \bot_{\{1,2,3\}}}$$

where $v : \{1,2,3\} \to \{1,2\}$ maps 1 to 1, and 2 and 3 to 2.

Indexed linear logic

The derivations of the type system given by ILL can be reconstructed from ILL itself — that is, removing the term and the intermediate level.

$$\dfrac{\dfrac{\dfrac{(q_1,\, q_2, q_3) \in Q^2}{c \,:\, q_j \,::\, \bot_{\{1,2,3\}} \vdash_{j \in \{1,2,3\}} c \,:\, q_j \,::\, \bot_{\{1,2,3\}}}}{c \,:\, [q_j] \,::\, !_{id}\, \bot_{\{1,2,3\}} \vdash_{j \in \{1,2,3\}} c \,:\, q_j \,::\, \bot_{\{1,2,3\}}}}{c \,:\, [q_j \mid v(j) = i] \,::\, !_v \bot_{\{1,2,3\}} \vdash_{i \in \{1,2\}} c \,:\, [q_j \mid v(j) = i] \,::\, !_v \bot_{\{1,2,3\}}}$$

Indexed linear logic

The derivations of the type system given by ILL can be reconstructed from ILL itself — that is, removing the term and the intermediate level.

$$\dfrac{\dfrac{\dfrac{(q_1,\, q_2, q_3) \in Q^2}{c \,:\, q_j \,::\, \bot_{\{1,2,3\}} \vdash_{j\in\{1,2,3\}} c \,:\, q_j \,::\, \bot_{\{1,2,3\}}}}{c \,:\, [q_j] \,::\, !_{id}\, \bot_{\{1,2,3\}} \vdash_{j\in\{1,2,3\}} c \,:\, q_j \,::\, \bot_{\{1,2,3\}}}}{c \,:\, [q_j \mid v(j) = i] \,::\, !_v\, \bot_{\{1,2,3\}} \vdash_{i\in\{1,2\}} c \,:\, [q_j \mid v(j) = i] \,::\, !_v\, \bot_{\{1,2,3\}}}$$

Indexed linear logic

The derivations of the type system given by ILL can be reconstructed from ILL itself — that is, removing the term and the intermediate level.

$$\dfrac{\dfrac{\dfrac{(q_1,\, q_2, q_3) \in Q^2}{q_j \;::\; \bot_{\{1,2,3\}} \;\vdash_{j\in\{1,2,3\}}\; q_j \;::\; \bot_{\{1,2,3\}}}}{[q_j] \;::\; !_{id}\, \bot_{\{1,2,3\}} \;\vdash_{j\in\{1,2,3\}}\; q_j \;::\; \bot_{\{1,2,3\}}}}{[q_j \mid v(j) = i] \;::\; !_v\, \bot_{\{1,2,3\}} \;\vdash_{i\in\{1,2\}}\; [q_j \mid v(j) = i] \;::\; !_v\, \bot_{\{1,2,3\}}}$$

We do not need terms anymore, thanks to the logical structuration of the proof.

Indexed linear logic

The derivations of the type system given by ILL can be reconstructed from ILL itself — that is, removing the term and the intermediate level.

$$\dfrac{\dfrac{\dfrac{(q_1,\, q_2, q_3) \in Q^2}{\bot_{\{1,2,3\}} \;\vdash_{\{1,2,3\}}\; \bot_{\{1,2,3\}}}}{!_{id}\, \bot_{\{1,2,3\}} \;\vdash_{\{1,2,3\}}\; \bot_{\{1,2,3\}}}}{!_v\, \bot_{\{1,2,3\}} \;\vdash_{\{1,2\}}\; !_v\, \bot_{\{1,2,3\}}}$$

We do not need the elements of the relational model. They can be reconstructed from the Axiom information (η-long form is crucial here).

A summary of the proof

Existence of a run-tree

$\iff$ Existence of a typing in an appropriate intersection type system

$\iff$ Existence of a typing by ILL

$\iff$ Existence of a derivation in ILL

$\iff$ Existence of an appropriate subset in the relational semantics of the term

Higher-order model checking

Two major issues of the model-checking problem were not adressed so far:

- recursion
- and parity conditions

Higher-order model checking

Recursion can be added with the rule

$$\text{Fix}\quad \frac{\Gamma \vdash_K M : [\tau_j \,|\, j \in u^{-1}(k)] \multimap \sigma_k :: !_u C \multimap A \qquad \Delta \vdash_J YM : \tau_j :: C}{\Gamma,\ !_u \Delta \vdash_K YM : \sigma_k :: A}$$

where C and A need to refine the same simple type.

This rule reflects recursion in an infinitary variant of the relational model: the only change is that now

$$[\![!A]\!] \;=\; \mathcal{M}_{\leq\omega}([\![A]\!])$$

Fixpoints are interpreted coinductively.

Higher-order model checking

The theorem connecting automata and models extends to this infinitary setting.

Checking whether

an alternating automaton has a run-tree over the tree produced by a recursion scheme

can thus be reduced to

computing the semantics of the corresponding λY-term and checking whether it contains a subset of $[\![\delta]\!] \times \{q_0\}$

Parity conditions

To capture all MSO, the alternating automaton needs to check whether it iterated finitely the properties whose infinite recursion was forbidden.

This is done *a posteriori*, by discriminating run-trees.

States are now coloured by a function $\Omega : Q \to \mathbb{N}$.

- A branch of a run-tree is winning if it is finite or if, among the colours it contains infinitely often, the greatest is even.
- A run-tree is winning if and only if all its branches are.

An APT accepts a tree iff it has a winning run-tree over it.

Parity conditions

To capture all MSO, the alternating automaton needs to check whether it iterated finitely the properties whose infinite recursion was forbidden.

This is done *a posteriori*, by discriminating run-trees.

States are now coloured by a function $\Omega : Q \to \mathbb{N}$.

- A branch of a run-tree is winning if it is finite or if, among the colours it contains infinitely often, the greatest is even.
- A run-tree is winning if and only if all its branches are.

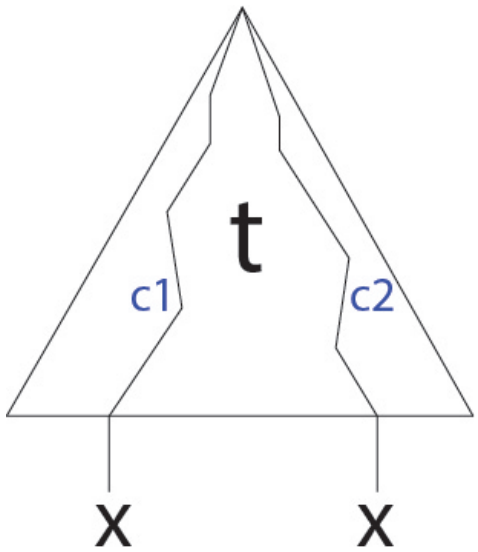
An APT accepts a tree iff it has a winning run-tree over it.

Parity conditions

Kobayashi and Ong extended the typing with a colouring operation:

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

This operation lifts to higher-order.



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

Parity conditions

We investigated the semantic nature of $\Box$, and proved that it is a parametric comonad. It can be added to the relational model by setting

$$[\![\Box\, A]\!] \;=\; \mathit{Col} \times [\![A]\!]$$

Moreover, $\Box$ is distributive over $!$, such that the intuitionnistic arrow $A \Rightarrow B$ can now be interpreted as

$$!\,\Box\, A \multimap B$$

Parity conditions

We investigated the semantic nature of $\Box$, and proved that it is a parametric comonad. It can be added to the relational model by setting

$$[\![\Box\, A]\!] \;=\; \mathit{Col} \times [\![A]\!]$$

Moreover, $\Box$ is distributive over !, such that the intuitionnistic arrow $A \Rightarrow B$ can now be interpreted as

$$!\Box\, A \multimap B$$

Parity conditions

For example, setting $\Omega(q_i) = c_i$, consider

$$\delta(q_0, a) = (2, q_2) \quad \text{and} \quad \delta(q_0, a) = (1, q_1) \wedge (1, q_2) \wedge (2, q_0)$$

which corresponds to the intersection typing

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

is interpreted in the model as

$$([\,], ([(c_2, q_2)], q_0)) \text{ and } ([(c_1, q_1), (c_2, q_2)], ([(c_0, q_0)], q_0)) \in [\![a]\!]$$

Parity conditions

For example, setting $\Omega(q_i) = c_i$, consider

$$\delta(q_0, a) = (2, q_2) \quad \text{and} \quad \delta(q_0, a) = (1, q_1) \wedge (1, q_2) \wedge (2, q_0)$$

which corresponds to the intersection typing

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

is interpreted in the model as

$$([], ([(c_2, q_2)], q_0)) \text{ and } ([(c_1, q_1), (c_2, q_2)], ([(c_0, q_0)], q_0)) \in [\![a]\!]$$

Parity conditions

For example, setting $\Omega(q_i) = c_i$, consider

$$\delta(q_0, a) = (2, q_2) \quad \text{and} \quad \delta(q_0, a) = (1, q_1) \wedge (1, q_2) \wedge (2, q_0)$$

which corresponds to the intersection typing

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

is interpreted in the model as

$$([], ([(c_2, q_2)], q_0)) \text{ and } ([(c_1, q_1), (c_2, q_2)], ([(c_0, q_0)], q_0)) \in [\![a]\!]$$

Parity conditions and indexed linear logic

This comonad also extends indexed linear logic with colouration.

$$\text{Left } \Box \quad \frac{\Gamma,\ x : \sigma_j :: A \vdash_J M : \tau_j :: B}{\Gamma,\ x : (-1, \sigma_j) :: \Box_{\vec{c}}\, A \vdash_J M : \tau_j :: B} \qquad \vec{c} = (-1)_{j \in J}$$

$$\text{Right } \Box \quad \frac{\Gamma \vdash_J M : \tau_j :: B}{\Box_{\vec{c}}\, \Gamma \vdash_J M : (c_j, \tau_j) :: \Box_{\vec{c}}\, B} \qquad \vec{c} = (c_j) \in Col^J$$

Parity conditions and indexed linear logic

In order to reflect the notion of winning run-tree in higher-order derivations, we introduce a notion of winning derivation.

This corresponds to the usual parity condition, but over the "branches" of coloured ILL derivation trees - note that the definition needs to be adapted to indexation, the parity computation is somehow "parallelized".

This parity condition has a smooth behaviour under β-reduction: the set of branch colourings can only decrease, so that the normal form (= the tree produced by the scheme) obtained from a winning derivation gives a winning run-tree of the automaton.

Parity conditions and indexed linear logic

In order to reflect the notion of winning run-tree in higher-order derivations, we introduce a notion of winning derivation.

This corresponds to the usual parity condition, but over the "branches" of coloured ILL derivation trees - note that the definition needs to be adapted to indexation, the parity computation is somehow "parallelized".

This parity condition has a smooth behaviour under β-reduction: the set of branch colourings can only decrease, so that the normal form (= the tree produced by the scheme) obtained from a winning derivation gives a winning run-tree of the automaton.

Parity conditions and indexed linear logic

The theorem extends to coloured indexed linear calculus.

In other words, $\mathcal{A}$ has a winning run-tree over the value tree of a scheme iff, denoting t the corresponding λY-term, there is a winning derivation tree of the sequent

$$\Gamma \vdash_{\{\star\}} t : q_0 :: \bot$$

where Γ complies with δ.

Connection with the coloured relational model

However, in the coloured relational model we considered, there is no way to exclude denotations violating the parity condition.

We need to consider an adapted fixpoint operator, as Melliès recently did for poset-based models of linear logic.

This is left to future work.

A last remark: extensional collapses

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

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

We are currently adapting this theorem type-theoretically to the infinitary and coloured settings.

This leads us to interpretations in finite domains of finite λY-terms, thus to decidability.

A last remark: extensional collapses

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

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

We are currently adapting this theorem type-theoretically to the infinitary and coloured settings.

This leads us to interpretations in finite domains of finite λY-terms, thus to decidability.

A last remark: extensional collapses

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

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

We are currently adapting this theorem type-theoretically to the infinitary and coloured settings.

This leads us to interpretations in finite domains of finite λY-terms, thus to decidability.

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: give a proper "parity" fixpoint for the relational model, finish the coloured extensional collapse result, axiomatize this extension of "recognition by monoid", ...

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: give a proper "parity" fixpoint for the relational model, finish the coloured extensional collapse result, axiomatize this extension of "recognition by monoid", ...

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: give a proper "parity" fixpoint for the relational model, finish the coloured extensional collapse result, axiomatize this extension of "recognition by monoid", ...

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: give a proper "parity" fixpoint for the relational model, finish the coloured extensional collapse result, axiomatize this extension of "recognition by monoid", ...