Coloured tensorial logic and higher-order model-checking

Charles Grellois (joint work with Paul-André Melliès)

PPS & LIAFA — Université Paris 7

April 11th, 2015

Model-checking higher-order programs

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

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

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

Model-checking higher-order programs

For higher-order programs with recursion, a natural model is

higher-order recursion schemes (HORS)

which generate a tree abstracting the set of potential behaviors of a program, and over which we want to run

alternating parity automata (APT)

in order to check whether some MSO formula holds.

Model-checking higher-order programs

For higher-order programs with recursion, a natural model is

higher-order recursion schemes (HORS)

which generate a tree abstracting the set of potential behaviors of a program, and over which we want to run

alternating parity automata (APT)

in order to check whether some MSO formula holds.

Model-checking higher-order programs

This model-checking problem is decidable:

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

As we will see, the challenge is to understand how an automaton acts at higher-order, directly on terms.

Higher-order recursion schemes

Higher-order recursion schemes

Idea: it is a kind of grammar whose parameters may be functions and which generates ranked trees labelled by elements of a ranked alphabet Σ.

Alternatively, it is a formalism equivalent to λY calculus with uninterpreted constants of order at most one.

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

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

is modelled as a recursion scheme:

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

Value tree of a recursion scheme

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

generates:

S

Value tree of a recursion scheme

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

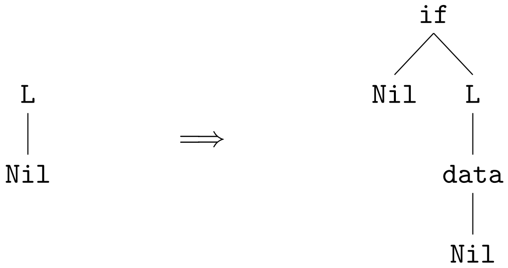
generates:

$$\texttt{S} \quad \Longrightarrow$$

Value tree of a recursion scheme

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

generates:

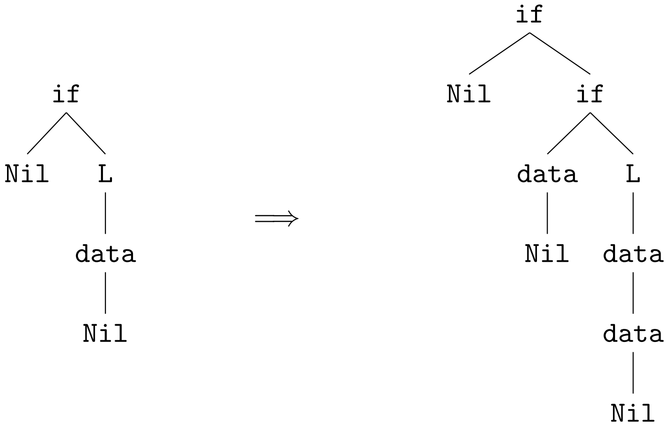


Notice that substitution and expansion occur in one same step.

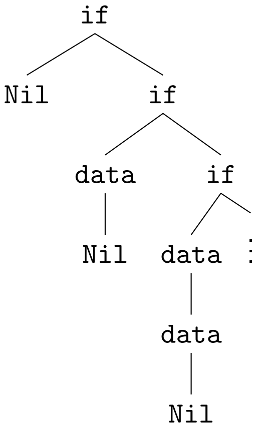
Value tree of a recursion scheme

S	=	L Nil
L x	=	if x (L (data x))

generates:



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, in order to get a decidability proof of the result, we need to analyze the recursion scheme itself, and to predict the behaviour of the automaton directly over it.

In the sequel, we will consider the equivalent formalism of λ-terms with a recursion operator Y.

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, in order to get a decidability proof of the result, we need to analyze the recursion scheme itself, and to predict the behaviour of the automaton directly over it.

In the sequel, we will consider the equivalent formalism of λ-terms with a recursion operator Y.

Alternating tree automata

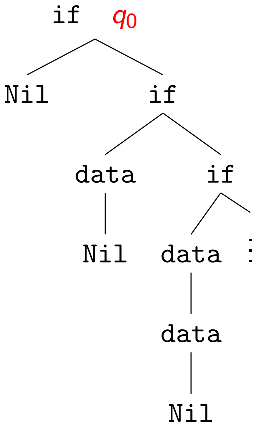
Alternating tree automata (ATA) are non-deterministic tree automata whose transitions may duplicate or drop a subtree.

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

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

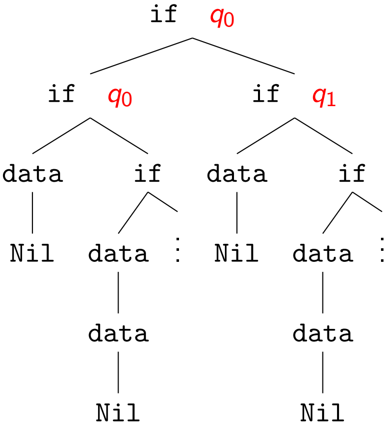
Alternating tree automata

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



Alternating tree automata

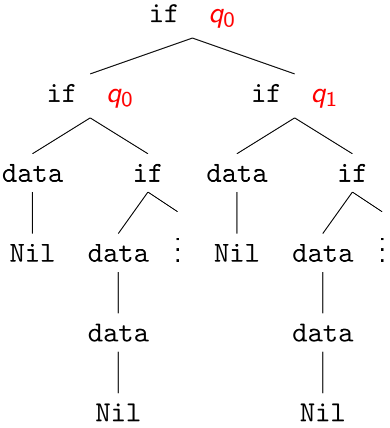
$\delta(q_0, \mathtt{if}) \ = \ (2, q_0) \wedge (2, q_1).$



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

Alternating tree automata

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



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

Alternating 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

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

In previous work, we studied these intersection types at the light of indexed linear logic, and of its relational semantics.

The intersection operation acts as a uniform exponential: it duplicates resources corresponding to the same term.

Alternating 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

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

In previous work, we studied these intersection types at the light of indexed linear logic, and of its relational semantics.

The intersection operation acts as a uniform exponential: it duplicates resources corresponding to the same term.

Alternating 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

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

In previous work, we studied these intersection types at the light of indexed linear logic, and of its relational semantics.

The intersection operation acts as a uniform exponential: it duplicates resources corresponding to the same term.

Tensorial logic

Tensorial logic

Tensorial logic is a obtained from a fragment of linear logic, by relaxing the hypothesis that the negation should be involutive.

It can be understood as a logical description of game semantics, connected to the theory of continuations.

For our talk, a crucial point is that tensorial logic builds a bridge between

- derivations of formulas of the logic,
- typing derivations of terms with formulas of the logic,
- and the construction of denotations in models of the logic.

Tensorial logic

Tensorial logic is a obtained from a fragment of linear logic, by relaxing the hypothesis that the negation should be involutive.

It can be understood as a logical description of game semantics, connected to the theory of continuations.

For our talk, a crucial point is that tensorial logic builds a bridge between

- derivations of formulas of the logic,
- typing derivations of terms with formulas of the logic,
- and the construction of denotations in models of the logic.

Tensorial logic

Define the formulas as

$$
\begin{array}{rcl}
A, B & ::= & 1 \\
 & \mid & A \otimes B \\
 & \mid & \neg_q \ A \\
 & \mid & [\, A_j \mid j \in J \,]
\end{array}
$$

where $q \in Q$ is an element of a finite set of states used to interpret the return type of functions.

Each turn of the interaction therefore exchanges information about the current state.

A natural model: relational semantics.

Tensorial logic

Define the formulas as

$$
\begin{array}{lcl}
A, B & ::= & 1 \\
 & \mid & A \otimes B \\
 & \mid & \neg_q \; A \\
 & \mid & [\, A_j \mid j \in J \,]
\end{array}
$$

where $q \in Q$ is an element of a finite set of states used to interpret the return type of functions.

Each turn of the interaction therefore exchanges information about the current state.

A natural model: relational semantics.

A duality between terms and alternating automata

Linear typing of tree-producing terms

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

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

(we will consider recursion later).

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

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

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

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

using the usual decomposition $A \Rightarrow B = !A \multimap B$.

Linear typing of tree-producing terms

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

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

(we will consider recursion later).

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

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

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

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

using the usual decomposition $A \Rightarrow B \;=\; !\,A \multimap B$.

Linear typing of tree-producing terms

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

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

(we will consider recursion later).

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

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

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

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

using the usual decomposition $A \Rightarrow B = !A \multimap B$.

Linear typing of tree-producing terms

Its dual $A^\perp$ is

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

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

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

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

Linear typing of tree-producing terms

Its dual $A^\perp$ is

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

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

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

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

Linear typing of tree-producing terms

In game semantics, the λ-term producing the tree is of type

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

and can be understood as an innocent strategy visiting the signature to produce a tree.

Dually, we can understand the alternating automaton as a counter-program which controls the change of state during the computation of the tree.

An interaction corresponds to a semantic computation of a proof of acceptance from the initial state – that is, of a run-tree of the automaton over the tree computed by the term.

Linear typing of tree-producing terms

In game semantics, the λ-term producing the tree is of type

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

and can be understood as an innocent strategy visiting the signature to produce a tree.

Dually, we can understand the alternating automaton as a counter-program which controls the change of state during the computation of the tree.

An interaction corresponds to a semantic computation of a proof of acceptance from the initial state – that is, of a run-tree of the automaton over the tree computed by the term.

Relational interpretation and automata acceptance

This duality leads to a semantic model-checking theorem:

Theorem (G.-Melliès 2014)

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

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

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

where the interpretation is computed in the relational model, the base type (of trees) being interpreted as Q.

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

Relational interpretation and automata acceptance

This duality leads to a semantic model-checking theorem:

Theorem (G.-Melliès 2014)

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

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

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

where the interpretation is computed in the relational model, the base type (of trees) being interpreted as Q.

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

An infinitary model of linear logic

An infinitary relation semantics

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

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

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

(finite-or-countable multisets)

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

An infinitary relation semantics

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

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

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

(finite-or-countable multisets)

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

An infinitary relation semantics

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

Theorem (G.-Melliès 2014)

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

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

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

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

An infinitary relation semantics

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

Theorem (G.-Melliès 2014)

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

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

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

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

An infinitary relation semantics

We can extend tensorial logic with countable multisets, and allow infinite-depth derivations.

This suggests an extension of the connection between tensorial logic and game semantics to infinite derivations and infinite interactions.

(which is not precisely formalized yet)

Specifying inductive and coinductive behaviours: parity conditions

Alternating parity tree automata

MSO allows to discriminate inductive from coinductive behaviour.

This allows to express properties as

> "a given operation is executed infinitely often in some execution"

or

> "after a `read` operation, a `write` eventually occurs".

Alternating parity tree automata

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

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

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

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

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

Alternating parity tree automata

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

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

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

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

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

The coloring parametric comonad

We discovered that the coloring operation behaves as a

parametric comonad

Informally, there is

- a neutral color ϵ, corresponding to the absence of box,
- and a composition mechanism which computes the maximum color of (finitely many) boxes.

It can be incorporated to tensorial logic, as well as to its relational semantics.

Tensorial logic with colors

This modality can be incorporated to tensorial logic, as well as to its relational semantics.

In the logic, we add a family of indexed operations $\Box_m$, and recast the parity condition on derivation trees: a rule

$$\frac{\Gamma \vdash M : A :: \kappa}{\Box_m \Gamma \vdash M : \Box_m A :: \kappa} \quad \text{Right } \Box_m$$

produces color m, and the others have no color (or ϵ).

We adapt a notion of winning derivation tree, which can be intuitively understood as a game semantics with parity conditions.

Tensorial logic with colors

We obtain:

Theorem

Consider an alternating parity automaton $\mathcal{A}$ and a HORS $\mathcal{G}$.
Then $\mathcal{A}$ has a winning run-tree over $[\![\mathcal{G}]\!]$ iff there exists a context Γ and a winning derivation tree in colored tensorial logic of the sequent

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

(where Γ types the tree constructors).

Colored relational semantics

In the relational semantics, a distributivity law between $\lightning$ and $\Box$ allows to consider their composition as a new exponential.

We add to the resulting model of λ-calculus a parity fixed point operator, and obtain

Theorem

An alternating parity tree automaton $\mathcal{A}$ with a set of states Q has a winning run-tree with initial state q_0 over $[\![\mathcal{G}]\!]$ if and only if there exists

$$u \in [\![\, \delta^{\dagger} \,]\!]_{col}$$

such that

$$(u, q_0) \in [\![\, \mathcal{G} \,]\!]_{col}$$

Colored relational semantics

In the relational semantics, a distributivity law between $\lightning$ and $\Box$ allows to consider their composition as a new exponential.

We add to the resulting model of λ-calculus a parity fixed point operator, and obtain

Theorem

An alternating parity tree automaton $\mathcal{A}$ with a set of states Q has a winning run-tree with initial state q_0 over $[\![\mathcal{G}]\!]$ if and only if there exists

$$u \in [\![\, \delta^{\dagger} \,]\!]_{col}$$

such that

$$(u, q_0) \in [\![\, \mathcal{G} \,]\!]_{col}$$

A finitary colored model of the λY-calculus

The Scott model of linear logic

In order to get a decidability proof, we need to recast our approach in a finitary setting.

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

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

We could make this adaptation and obtain a new decidability proof. But it raises a new question:

Is there a connection between this finitary semantics and an appropriate game semantics/tensorial logic ?

The Scott model of linear logic

In order to get a decidability proof, we need to recast our approach in a finitary setting.

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

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

We could make this adaptation and obtain a new decidability proof. But it raises a new question:

Is there a connection between this finitary semantics and an appropriate game semantics/tensorial logic ?