Tree automata and logical models

Charles Grellois (joint work with Paul-André Melliès)

PPS & LIAFA — Université Paris 7

October 3rd, 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.

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

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

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

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

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

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

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

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

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

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

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 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 λ calculus with recursion and 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

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

formulated as a recursion scheme:

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

or, in λ-calculus style :

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

(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 :

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

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

Value tree of a recursion scheme

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

generates:

S

Value tree of a recursion scheme

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

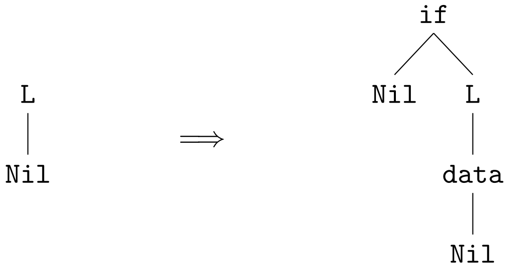
generates:

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

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:

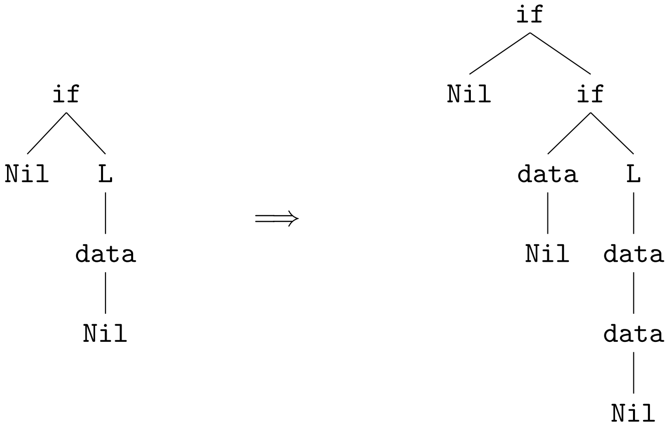


Notice that substitution and expansion occur in one same step.

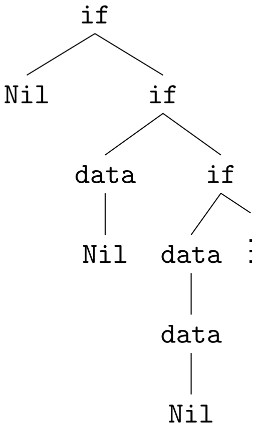
Value tree of a recursion scheme

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

generates:

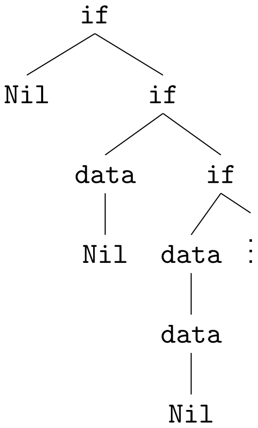


Value tree of a recursion scheme



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 scheme (in fact, the associated λ-term) in an algebraic structure suitable for higher-order interpretations: some logical model.

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

This motivates the use of suitable models of linear logic for interpreting schemes.

Alternating parity tree automata

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

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

This is reminiscent of the exponential modality of linear logic

This motivates the use of suitable models of linear logic for interpreting schemes.

Alternating parity tree automata

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

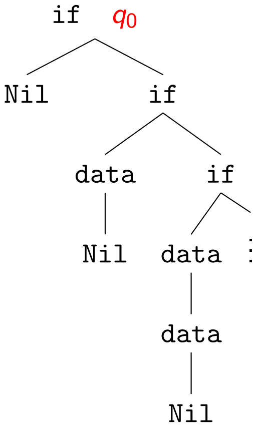
Example: $\delta(q_0, \texttt{if}) \ = \ (2, q_0) \wedge (2, q_1)$.

This is reminiscent of the exponential modality of linear logic

This motivates the use of suitable models of linear logic for interpreting schemes.

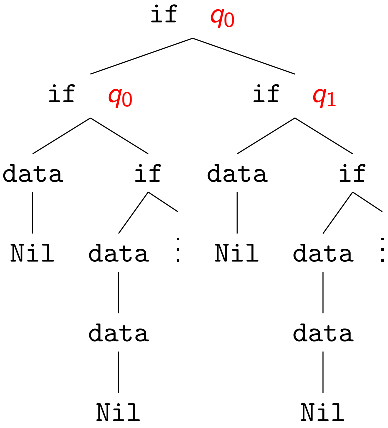
Alternating parity tree automata

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



Alternating parity tree automata

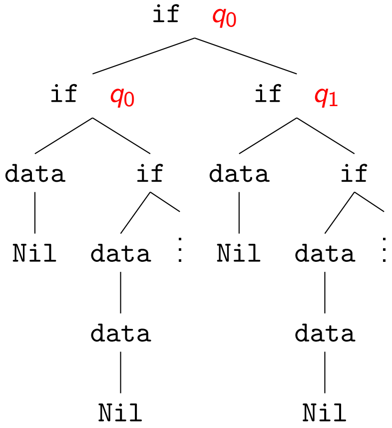
$\delta(q_0, \mathtt{if}) \;=\; (2, q_0) \wedge (2, q_1).$



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

$$a \;:\; \bot \rightarrow \bot \rightarrow \bot$$

with intersection types:

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

This connection with intersection types is a step towards a model-theoretic interpretation.

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

$$a \;:\; \perp \to \perp \to \perp$$

with intersection types:

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

This connection with intersection types is a step towards a model-theoretic interpretation.

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

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

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

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

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

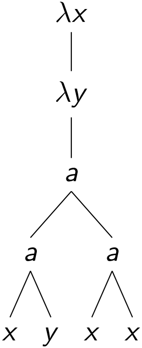
In other words, it is some collection (with multiplicities) of elements of $[\![A]\!]$ producing an element of $[\![B]\!]$.

An example of interpretation

Consider the rule

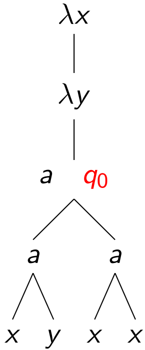
$$F\ x\ y\ =\ a\,(\,a\ x\ y\,)\,(\,a\ x\ x\,)$$

which corresponds to



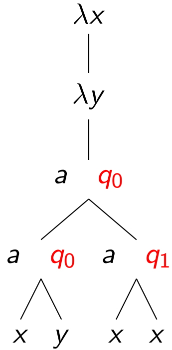
An example of interpretation

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

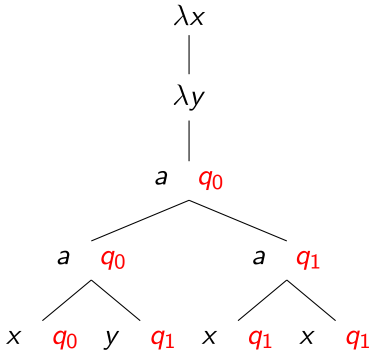


An example of interpretation

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



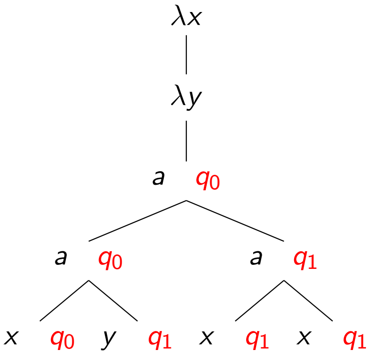
An example of interpretation



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

Theorem (G.-Melliès 2014)

Consider an alternating 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

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

Note that the interpretation of $\mathcal{G}$ depends of the choice of $\mathcal{A}$.
This dependence can be removed by reformulating the problem.

This is a compositional approach of model-checking.

Relational interpretation and automata acceptance

Theorem (G.-Melliès 2014)

Consider an alternating 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

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

Note that the interpretation of $\mathcal{G}$ depends of the choice of $\mathcal{A}$.
This dependence can be removed by reformulating the problem.

This is a compositional approach of model-checking.

Relational interpretation and automata acceptance

Theorem (G.-Melliès 2014)

Consider an alternating 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

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

Note that the interpretation of $\mathcal{G}$ depends of the choice of $\mathcal{A}$.
This dependence can be removed by reformulating the problem.

This is a compositional approach of model-checking.

Elements of proof

The proof relies on the equivalence of

run-trees and intersection typings

by Kobayashi, and of

intersection typings and (some) relational interpretations

by Grellois and Melliès (which uses a logical correspondence of Bucciarelli and Ehrhard).

Higher-order model checking

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

- recursion
- and parity conditions

Recursion can be added to the models and typings in the usual (coinductive) way.

Parity is more challenging.

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.

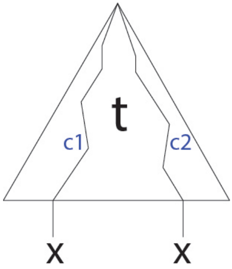
An APT accepts a tree iff it has a winning run-tree over it.

Parity conditions

Kobayashi and Ong extended the typing with a colouring operation:

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

This operation lifts to higher-order.



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

Parity conditions

We investigated the semantic nature of $\Box$, and proved that it has good properties.

It can be added to the model, and there is a very natural coloured interpretation of types:

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

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

Parity conditions

We investigated the semantic nature of $\Box$, and proved that it has good properties.

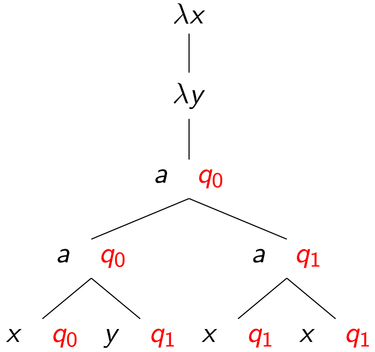
It can be added to the model, and there is a very natural coloured interpretation of types:

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

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

An example of coloured interpretation

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

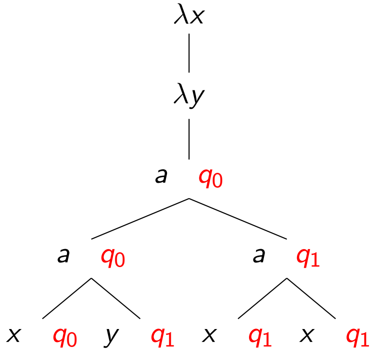


This rule will be interpreted in the model as

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

An example of coloured interpretation

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



This rule will be interpreted in the model as

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

Parity conditions and typing derivations

There are two crucial points:

- this can still be formulated equivalently in typing derivations
- the colouring of branches is preserved by the rewriting dynamics of the scheme

So we can decide directly from a typing of the scheme's rules whether the tree it produces will be accepted.

Parity conditions and typing derivations

There are two crucial points:

- this can still be formulated equivalently in typing derivations
- the colouring of branches is preserved by the rewriting dynamics of the scheme

So we can decide directly from a typing of the scheme's rules whether the tree it produces will be accepted.

Parity conditions and typing derivations

The typing derivations are now of infinite depth.

We introduce the notion of winning derivation by setting the usual parity condition over typing trees.

Theorem (G.-Melliès 2014)

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 $\mathcal{G}$.

Parity conditions and typing derivations

The typing derivations are now of infinite depth.

We introduce the notion of winning derivation by setting the usual parity condition over typing trees.

Theorem (G.-Melliès 2014)

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 $\mathcal{G}$.

Connection with the coloured relational model

In order to obtain the corresponding model-theoretic version of this theorem, we need to add an appropriate fixpoint to the model.

In some sense, this fixpoint operator composes elements of the interpretation of a term in all possible ways which satisfy the parity condition.

This leads to an extension of the theorem to the coloured model-theoretic case.

Connection with the coloured relational model

In order to obtain the corresponding model-theoretic version of this theorem, we need to add an appropriate fixpoint to the model.

In some sense, this fixpoint operator composes elements of the interpretation of a term in all possible ways which satisfy the parity condition.

This leads to an extension of the theorem to the coloured model-theoretic case.

A last remark: extensional collapse and decidability

Ehrhard proved in 2012 a correspondence between the relational model we studied, and another model where multisets are replaced with sets.

We can extend this correspondence to the relational model proposed in this talk, so that all interpretations are taken in a finite model.

In other terms: the higher-order model-checking problem is decidable, again.

Conclusions and perspectives

- Terms can be interpreted in models reflecting the behaviour of APT.
- Model-checking reduces to computing the interpretation of a scheme, and checking whether it contains the initial state.
- This approach is equivalent to a type-theoretic one.
- Results of extensional collapse lead - again - to decidability.
- There is still a lot to do: axiomatize this extension of "recognition by monoid", develop a notion of game semantics with parity, extend the approach to other models of tree automata. . .

Conclusions and perspectives

- Terms can be interpreted in models reflecting the behaviour of APT.
- Model-checking reduces to computing the interpretation of a scheme, and checking whether it contains the initial state.
- This approach is equivalent to a type-theoretic one.
- Results of extensional collapse lead - again - to decidability.
- There is still a lot to do: axiomatize this extension of "recognition by monoid", develop a notion of game semantics with parity, extend the approach to other models of tree automata...

Conclusions and perspectives

- Terms can be interpreted in models reflecting the behaviour of APT.
- Model-checking reduces to computing the interpretation of a scheme, and checking whether it contains the initial state.
- This approach is equivalent to a type-theoretic one.
- Results of extensional collapse lead - again - to decidability.
- There is still a lot to do: axiomatize this extension of "recognition by monoid", develop a notion of game semantics with parity, extend the approach to other models of tree automata. . .

Conclusions and perspectives

- Terms can be interpreted in models reflecting the behaviour of APT.
- Model-checking reduces to computing the interpretation of a scheme, and checking whether it contains the initial state.
- This approach is equivalent to a type-theoretic one.
- Results of extensional collapse lead - again - to decidability.
- There is still a lot to do: axiomatize this extension of "recognition by monoid", develop a notion of game semantics with parity, extend the approach to other models of tree automata. . .

Conclusions and perspectives

- Terms can be interpreted in models reflecting the behaviour of APT.
- Model-checking reduces to computing the interpretation of a scheme, and checking whether it contains the initial state.
- This approach is equivalent to a type-theoretic one.
- Results of extensional collapse lead - again - to decidability.
- There is still a lot to do: axiomatize this extension of "recognition by monoid", develop a notion of game semantics with parity, extend the approach to other models of tree automata...