Introduction to higher-order verification II
Modal μ-calculus, tree automata and parity games

Charles Grellois

PPS & LIAFA — Université Paris 7

GdT Sémantique et Vérification – December 11th, 2014

Models for higher-order programs

Last time we introduced recursion schemes and lambda Y-calculus, two models for higher-order programs.

These models capture the higher-order flow of program with recursion, but abstracts conditionals, arithmetics, references...

We start by a quick reminder of them.

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

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

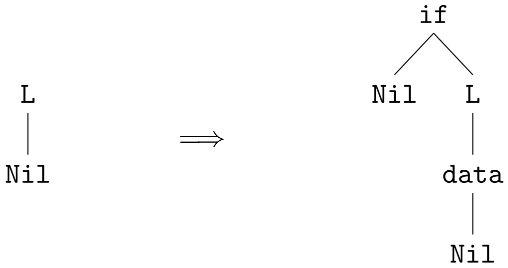
generates:

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

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:

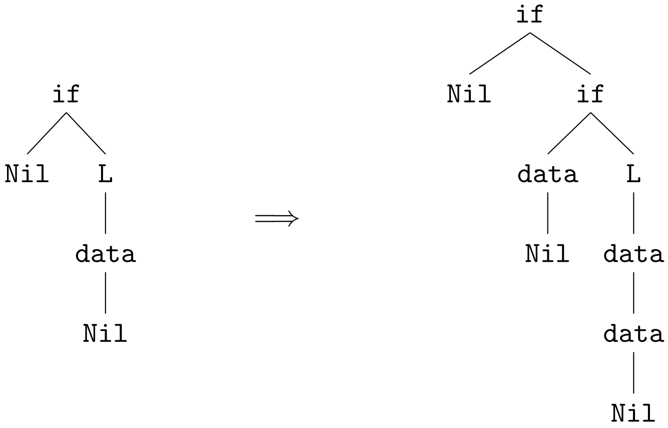


Notice that substitution and expansion occur in one same step.

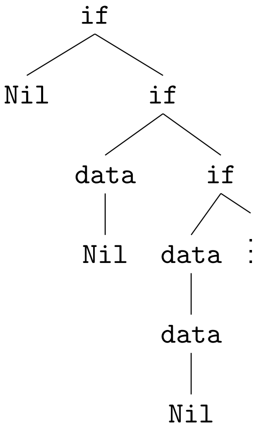
Value tree of a recursion scheme

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

generates:

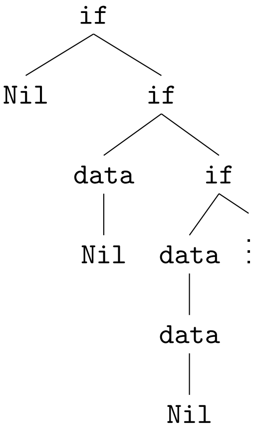


Value tree of a recursion scheme



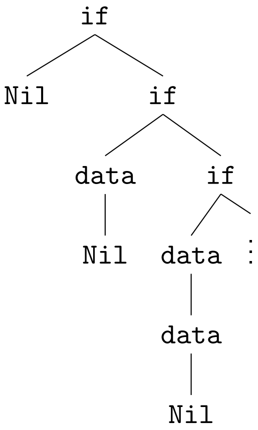
Important remark: this scheme is very simple, yet it produces a tree which is not regular (it does not have a finite number of subtrees).

Value tree of a recursion scheme



Important remark: this scheme is very simple, yet it produces a tree which is not regular (it does not have a finite number of subtrees).

Value tree of a recursion scheme

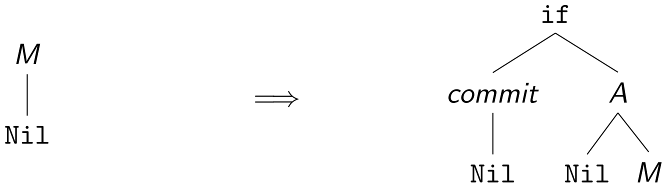


Examples of properties to check:

- the program may run infinitely if needed (liveness property)
- the program's outputs are finite

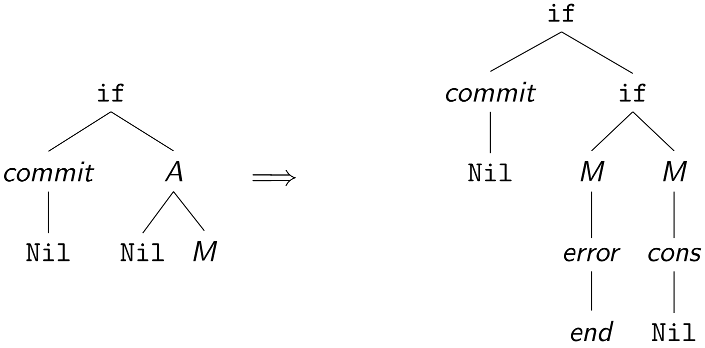
Value tree of a recursion scheme

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



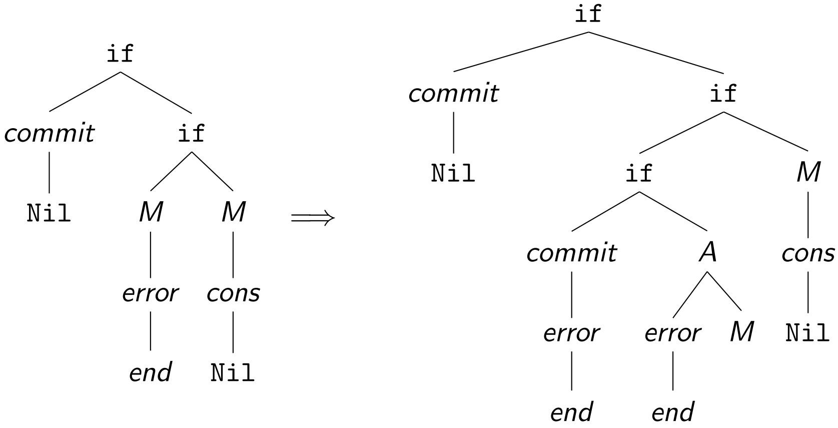
Value tree of a recursion scheme

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

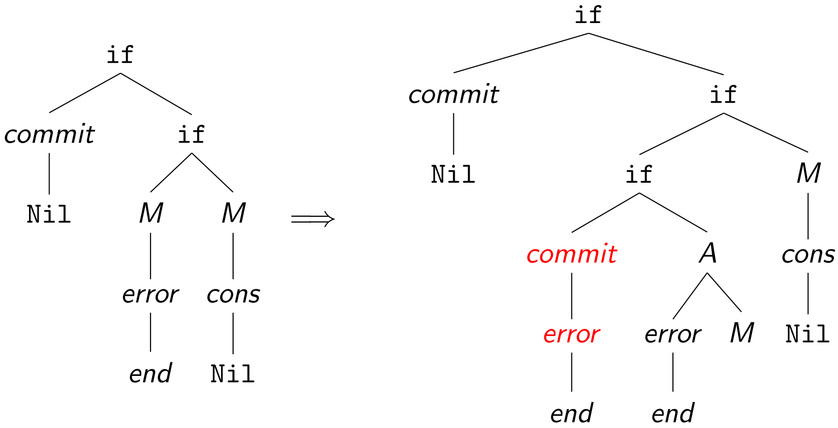


Value tree of a recursion scheme

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



Value tree of a recursion scheme



Example of property to check: the program never commits an *error* (safety property).

Trees: ranked vs unranked

A reminder from last week:

A Σ-labelled (ranked) tree is defined as a function $t : Dom(t) \to \Sigma$ with $Dom(t) \subseteq \mathbb{N}^*$ a prefix-closed set of finite words on natural numbers, satisfying the following property:

$$\forall \alpha \in Dom(t),\ \{i \mid \alpha \cdot i \in Dom(t)\} = \{1, \ldots, ar(t(\alpha))\}$$

When this last condition is relaxed, the tree is called unranked.

Trees: ranked vs unranked

A reminder from last week:

A Σ-labelled (ranked) tree is defined as a function $t : Dom(t) \rightarrow \Sigma$ with $Dom(t) \subseteq \mathbb{N}^*$ a prefix-closed set of finite words on natural numbers, satisfying the following property:

$$\forall \alpha \in Dom(t),\ \{i \mid \alpha \cdot i \in Dom(t)\} = \{1, \ldots, ar(t(\alpha))\}$$

When this last condition is relaxed, the tree is called unranked.

Trees: ranked vs unranked

A reminder from last week:

A Σ-labelled (ranked) tree is defined as a function $t : Dom(t) \to \Sigma$ with $Dom(t) \subseteq \mathbb{N}^*$ a prefix-closed set of finite words on natural numbers, satisfying the following property:

$$\forall \alpha \in Dom(t),\ \{i \mid \alpha \cdot i \in Dom(t)\} = \{1, \ldots, ar(t(\alpha))\}$$

When this last condition is relaxed, the tree is called unranked.

Trees: ranked vs unranked

The main difference between ranked and unranked trees is that ranked trees have a maximal arity, while unranked ones do not.

In other terms, in an unranked tree, there is no boundary on the number of directions we could take from a node.

So distinguishing directions would require a countable number of predicates.

MSO

The logic on which all the work on higher-order model-checking relies is the monadic second order logic.

It has the advantage of containing other standard temporal logics (CTL, LTL. . .), and to be close to the fronteer of decidability, in the sense that in situations where it is decidable, most of its extensions fail to be.

MSO

MSO extends first-order logic with quantification over monadic relations (in other terms: over sets).

We will not present this logic in this talk, but modal μ-calculus instead, since it is equi-expressive over trees (Janin-Walukiewicz 1996).

Moreover, modal μ-calculus is in a sense more algorithmic than MSO, and as such much closer to automata theory.

Modal μ-calculus

We fix a finite ranked alphabet Σ.

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 (the one in direction i)

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

Note that for an unranked structure, only $\diamond$ would make sense.

Modal μ-calculus

We fix a finite ranked alphabet Σ.

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 (the one in direction i)

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

Note that for an unranked structure, only $\diamond$ would make sense.

Modal μ-calculus

We fix a finite ranked alphabet Σ.

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 (the one in direction i)

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

Note that for an unranked structure, only $\diamond$ would make sense.

Modal μ-calculus

We fix a finite ranked alphabet Σ.

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 (the one in direction i)

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

Note that for an unranked structure, only $\diamond$ would make sense.

Modal μ-calculus

We fix a finite ranked alphabet Σ.

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

We fix a finite ranked alphabet Σ.

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

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

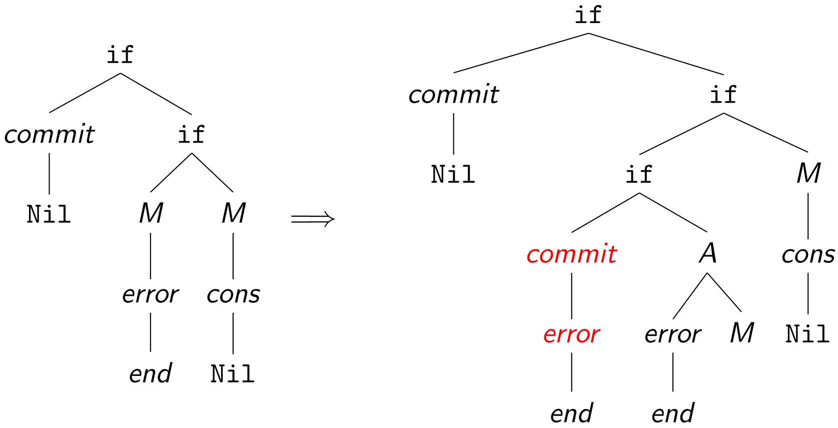
$$\nu X.\, \phi(X) \quad \longrightarrow \quad \phi(\nu X.\, \phi(X)) \quad \longrightarrow \quad \phi(\phi(\nu X.\, \phi(X)))$$

Modal μ-calculus

One can also define negation using usual de Morgan duality. There are just two points to notice:

- $\neg\ a \quad = \quad \bigvee_{b \in \Sigma \setminus \{a\}}\ b$
- and $\mu X.$ and $\nu X.$ are only allowed on formulas in which X only occurs positively.

Value tree of a recursion scheme



Example of property to check: the program never commits an *error* (safety property).

Specifying a property in modal μ-calculus

How do we specify that the second scheme does not commit an error ?
We want to forbid the existence of an instance of the symbol *error* on a branch after *commit* was seen.

There is a branch with an error in a tree $\Longleftrightarrow \mu X.\,(\ \diamond X \vee \mathit{error}\ )$

There is a branch containing an error in a tree whose root is labelled with a commit
$\Longleftrightarrow \mathit{commit} \wedge (\ \mu X.\,(\ \diamond X \vee \mathit{error}\ )\ )$

Specifying a property in modal μ-calculus

How do we specify that the second scheme does not commit an error ?
We want to forbid the existence of an instance of the symbol *error* on a branch after *commit* was seen.

There is a branch with an error in a tree $\Longleftrightarrow \mu X.(\ \diamond X \vee error\ )$

There is a branch containing an error in a tree whose root is labelled with a commit
$\Longleftrightarrow commit \wedge (\ \mu X.(\ \diamond X \vee error\ )\ )$

Specifying a property in modal μ-calculus

There is a branch containing an error in a tree whose root is labelled with a commit
$\iff \mathit{commit} \wedge (\ \mu X.\,(\ \diamond X \vee \mathit{error}\ )\ )$

There is a branch with an error after a commit
$\iff \mu Y.\ (\ \diamond Y \vee (\ \mathit{commit} \wedge (\ \mu X.\,(\ \diamond X \vee \mathit{error}\ )\ )\ )\ )$

Recall this is a safety property — notice we only used the μ quantifier.

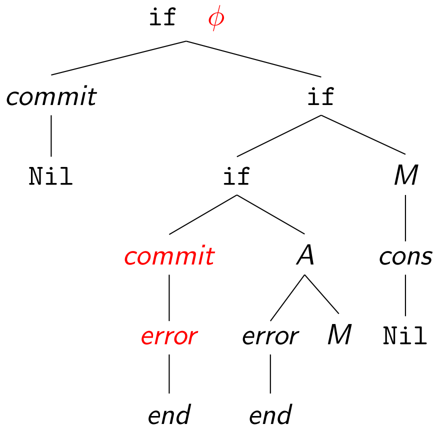
Specifying a property in modal μ-calculus

There is a branch containing an error in a tree whose root is labelled with a commit
$\Longleftrightarrow \mathit{commit} \wedge (\ \mu X.\ (\ \diamond X \vee \mathit{error}\ )\ )$

There is a branch with an error after a commit
$\Longleftrightarrow \mu Y.\ (\ \diamond Y \vee (\ \mathit{commit} \wedge (\ \mu X.\ (\ \diamond X \vee \mathit{error}\ )\ )\ )\ )$

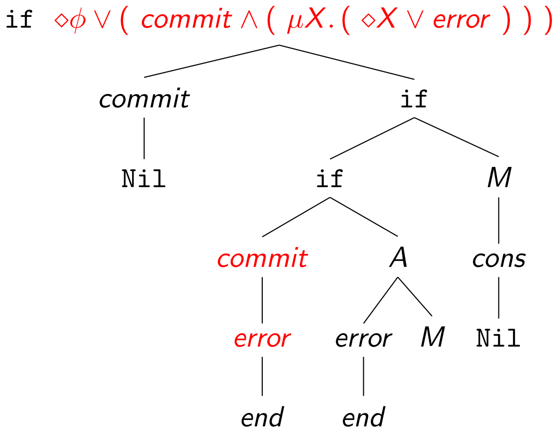
Recall this is a safety property — notice we only used the μ quantifier.

Value tree of a recursion scheme



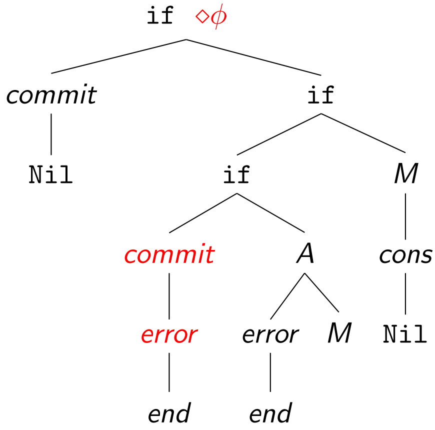
$$\phi \quad = \quad \mu Y.\,(\,\diamond Y \vee (\,commit \wedge (\,\mu X.\,(\,\diamond X \vee error\,)\,)\,)\,)$$

Value tree of a recursion scheme



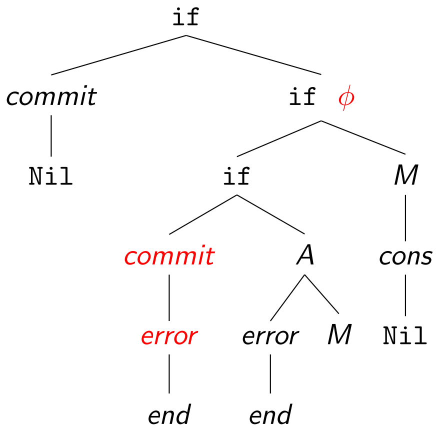
$$\phi \quad = \quad \mu Y.\,(\ \diamond Y \vee (\ \mathit{commit} \wedge (\ \mu X.\,(\ \diamond X \vee \mathit{error}\ )\ )\ )\ )$$

Value tree of a recursion scheme



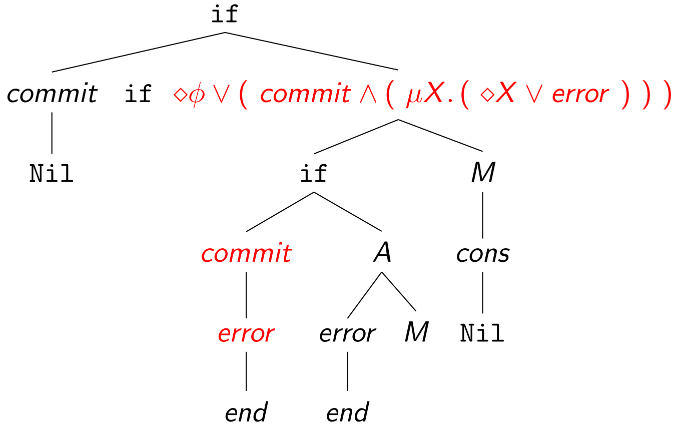
$$\phi \quad = \quad \mu Y.\,(\,\diamond Y \vee (\,\mathit{commit} \wedge (\,\mu X.\,(\,\diamond X \vee \mathit{error}\,)\,)\,)\,)$$

Value tree of a recursion scheme



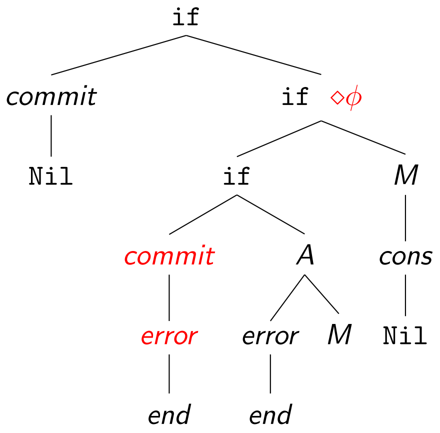
$$\phi \quad = \quad \mu Y.\,(\ \diamond Y \vee (\ commit \wedge (\ \mu X.\,(\ \diamond X \vee error\ )\ )\ )\ )$$

Value tree of a recursion scheme



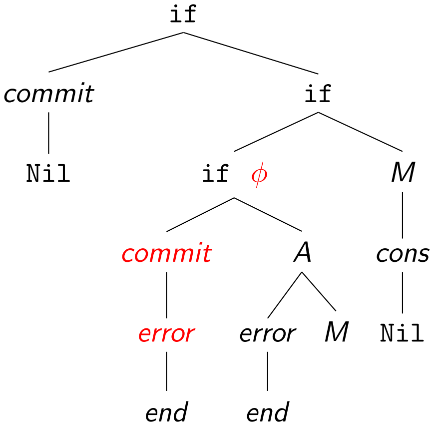
$$\phi \quad = \quad \mu Y. \, (\; \diamond Y \vee (\; commit \wedge (\; \mu X. (\; \diamond X \vee error \;) \;) \;) \;)$$

Value tree of a recursion scheme



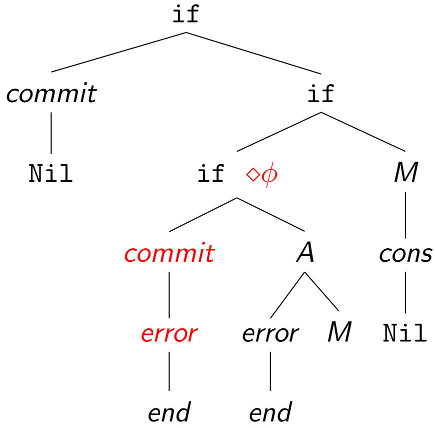
$$\phi \quad = \quad \mu Y.\,(\ \diamond Y \vee (\ commit \wedge (\ \mu X.\,(\ \diamond X \vee error\ )\ )\ )\ )$$

Value tree of a recursion scheme



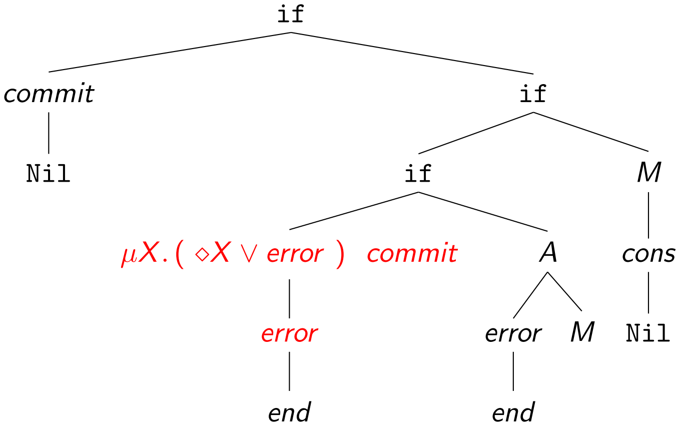
$$\phi \quad = \quad \mu Y.\,(\ \diamond Y \vee (\ commit \wedge (\ \mu X.\,(\ \diamond X \vee error\ )\ )\ )\ )$$

Value tree of a recursion scheme



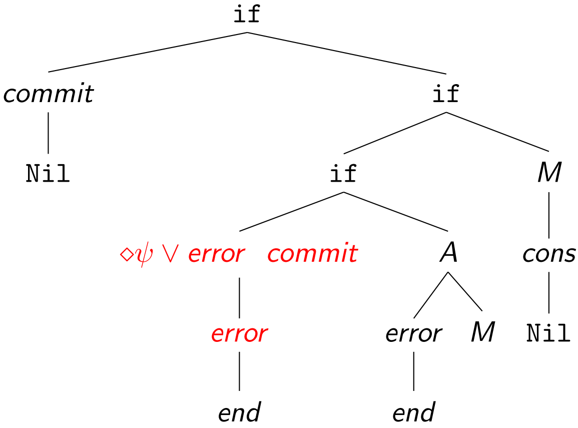
$$\phi \quad = \quad \mu Y.\ (\ \diamond Y \vee (\ commit \wedge (\ \mu X.\ (\ \diamond X \vee error\ )\ )\ )\ )$$

Value tree of a recursion scheme



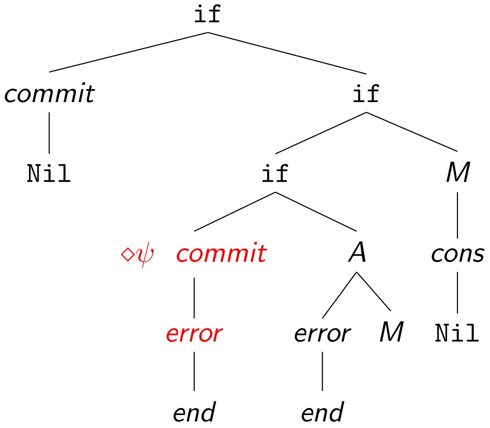
$$\phi \quad = \quad \mu Y. \, (\, \diamond Y \vee (\, \mathit{commit} \wedge (\, \mu X. \, (\, \diamond X \vee \mathit{error} \,) \,) \,) \,)$$

Value tree of a recursion scheme



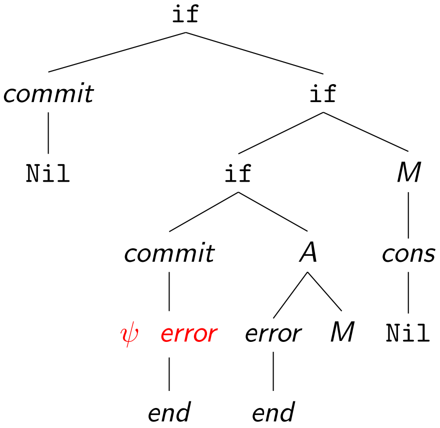
$$\psi \quad = \quad \mu X.\,(\ \Diamond X \vee error\ )$$

Value tree of a recursion scheme



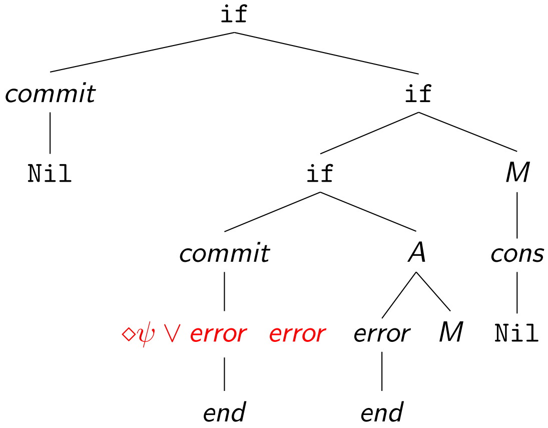
$$\psi \quad = \quad \mu X. (\ \diamond X \vee error\ )$$

Value tree of a recursion scheme



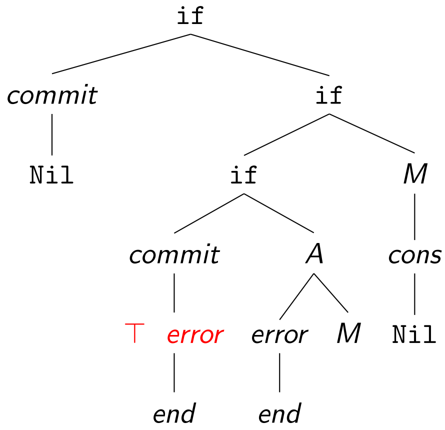
$$\psi \quad = \quad \mu X.\,(\ \diamond X \vee \mathit{error}\ )$$

Value tree of a recursion scheme



$$\psi \quad = \quad \mu X.\,(\ \Diamond X \vee error\ )$$

Value tree of a recursion scheme



so that the formula holds at the root. How can we make this more formal ?

Modal μ-calculus : semantics

Consider a Σ-labelled ranked tree t. Denote N the set of its nodes, and fix a valuation $\mathcal{V} : Var \rightarrow \mathcal{P}(N)$.

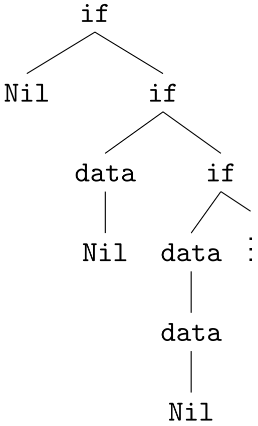
Then the semantics of a closed formula is a subset of N, to be understood as the set of nodes over which the formula is true (that is, from which it can be unravelled consistently with the μ/ν restrictions).

Modal μ-calculus : semantics

- $||a||_{\mathcal{V}} \;=\; \{n \in N \mid label(n) \;=\; a\}$
- $||X||_{\mathcal{V}} \;=\; \mathcal{V}(X)$
- $||\neg\phi||_{\mathcal{V}} \;=\; N \setminus ||\phi||_{\mathcal{V}}$
- $||\phi \vee \psi||_{\mathcal{V}} \;=\; ||\phi||_{\mathcal{V}} \cup ||\psi||_{\mathcal{V}}$
- $||\diamond_i\ \phi||_{\mathcal{V}} \;=\; \{n \in N \mid ar(n) \geq i \text{ and } succ_i(n) \in ||\phi||_{\mathcal{V}}\}$
- $||\mu X.\,\phi(X)||_{\mathcal{V}} \;=\; \bigcap \{M \subseteq N \mid ||\phi(X)||_{\mathcal{V}[X \leftarrow M]} \subseteq M\}$

where $\mathcal{V}[X \leftarrow M]$ coincides with $\mathcal{V}$ except on X to which it maps M.

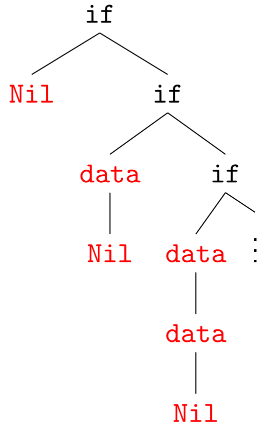
Semantics of a formula



What are the informal meaning and the semantics of

$$\mu X. \, (\, \texttt{Nil} \vee \Box X \,) \qquad ?$$

Semantics of a formula



$\mu X.$ ($\texttt{Nil} \vee \Box X$) means that every branch reaches Nil (and is thus finite, since Nil is nullary). The semantics is the set of coloured nodes.

Semantics of a formula

Formally,

$$||\mu X.\ (\ \mathtt{Nil} \vee \Box X\ )\ || \quad = \quad \bigcap \{ M \subseteq N \mid ||\mathtt{Nil} \vee \Box X||_{[X \mapsto M]} \subseteq M \}$$

where

$$||\mathtt{Nil} \vee \Box X||_{[X \mapsto M]}$$

is the set of nodes of the tree labelled with *Nil* or whose successors all belong to M.

Notice that N itself is a valid such M. But due to the intersection, only the minimal answer is kept: it is the set of non-`if` labelled nodes.

Semantics of a formula

Formally,

$$||\mu X.\ (\ \texttt{Nil} \vee \Box X\ )\,|| \quad = \quad \bigcap \{ M \subseteq N \mid ||\texttt{Nil} \vee \Box X||_{[X \mapsto M]} \subseteq M \}$$

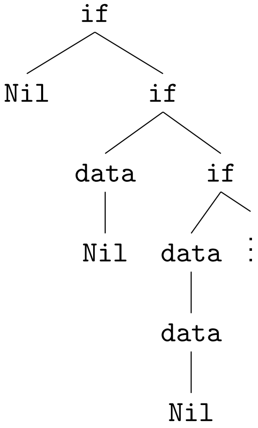
where

$$||\texttt{Nil} \vee \Box X||_{[X \mapsto M]}$$

is the set of nodes of the tree labelled with *Nil* or whose successors all belong to M.

Notice that N itself is a valid such M. But due to the intersection, only the minimal answer is kept: it is the set of non-`if` labelled nodes.

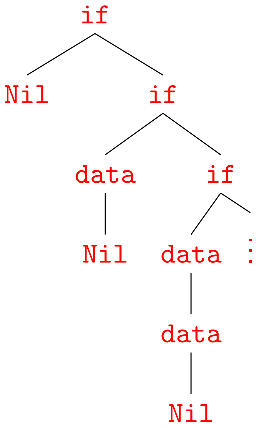
Semantics of a formula



What are the informal meaning and the semantics of

$$\nu X.\ (\ \texttt{Nil} \vee \Box X\ ) \qquad ?$$

Semantics of a formula



$\nu X.\ (\ \texttt{Nil} \vee \Box X\ )$ means that every finite branch reaches Nil. The semantics is the whole tree.

Specifying a property in modal μ-calculus

What does

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

mean ? What is its semantics on the previous tree ?

It is the set of `if`-labelled nodes.

Specifying a property in modal μ-calculus

What does

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

mean ? What is its semantics on the previous tree ?

It is the set of `if`-labelled nodes.

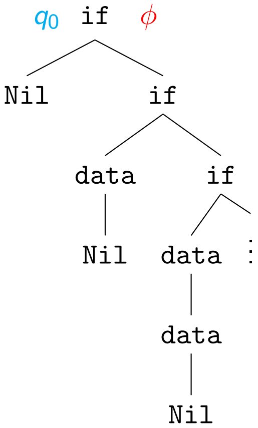
Interaction with trees: a shift to automata theory

The interaction of a formula with a tree is usually performed by an equivalent automaton.

Intuitively, it synchronises the unravelling of the formula with the letters of the tree.

Alternating parity tree automata

Idea: the formula "starts" on the root

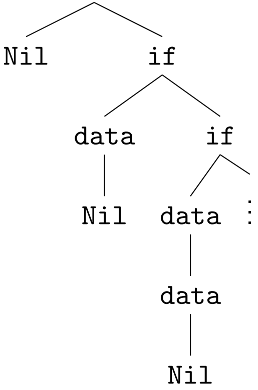


where $\phi = \nu X. \, (\, \mathtt{if} \wedge \diamond_1 \, (\, \mu Y. \, (\, \mathtt{Nil} \vee \Box Y \,) \,) \wedge \diamond_2 \, X \,)$ corresponds to a state q_0.

Alternating parity tree automata

Idea: the formula "starts" on the root

$$q_0 \quad \texttt{if} \qquad \texttt{if} \wedge \diamond_1 \left(\mu Y. \left(\texttt{Nil} \vee \Box Y \right) \right) \wedge \diamond_2 \, \phi$$

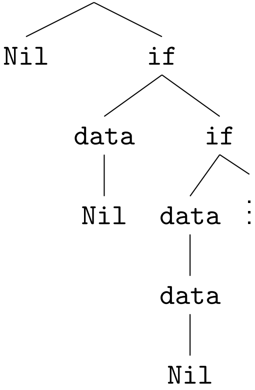


where $\phi \; = \; \nu X. \left(\texttt{if} \wedge \diamond_1 \left(\mu Y. \left(\texttt{Nil} \vee \Box Y \right) \right) \wedge \diamond_2 \, X \right)$ corresponds to a state q_0.

Alternating parity tree automata

Idea: the formula "starts" on the root

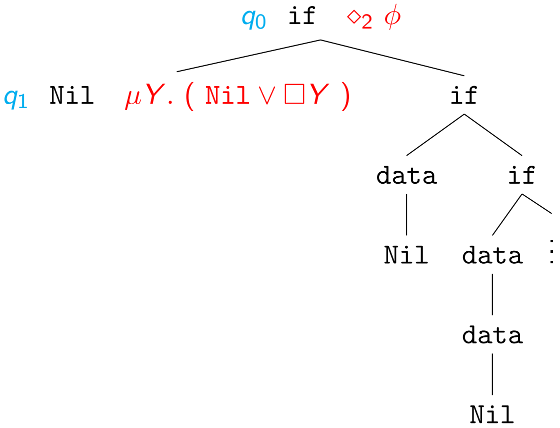
$$q_0 \quad \texttt{if} \quad \Diamond_1 \, (\, \mu Y. \, (\, \texttt{Nil} \vee \Box Y \,) \,) \wedge \Diamond_2 \, \phi$$



where $\phi \ = \ \nu X. \, (\, \texttt{if} \wedge \Diamond_1 \, (\, \mu Y. \, (\, \texttt{Nil} \vee \Box Y \,) \,) \wedge \Diamond_2 \, X \,)$ corresponds to a state q_0.

Alternating parity tree automata

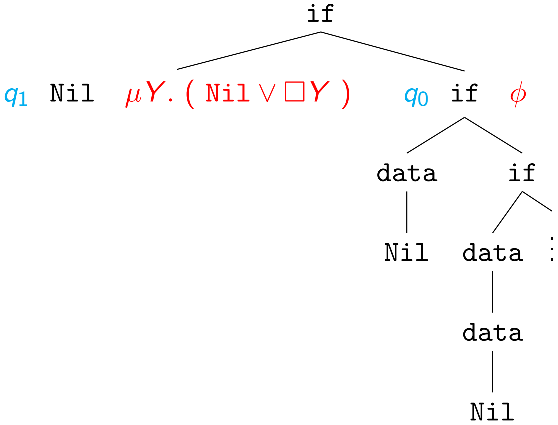
Idea: the formula "starts" on the root



where ϕ corresponds to a state q_0 and $\psi\ =\ \mu Y.\ (\ \mathtt{Nil} \vee \Box Y\ )$ to a state q_1.

Alternating parity tree automata

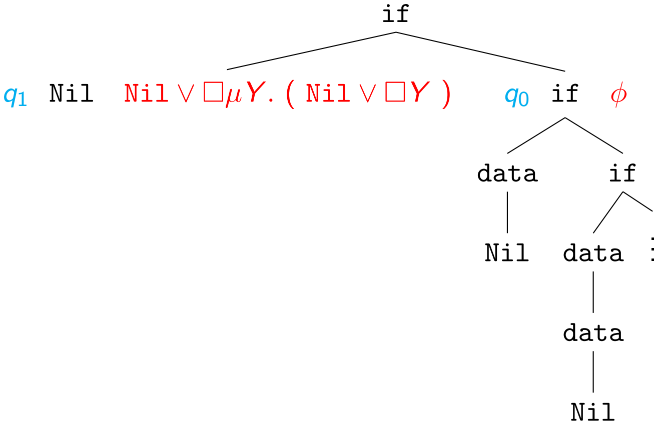
Idea: the formula "starts" on the root



where ϕ corresponds to a state q_0 and ψ to a state q_1

Alternating parity tree automata

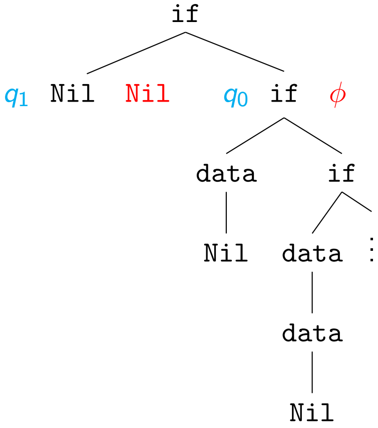
Idea: the formula "starts" on the root



where ϕ corresponds to a state q_0 and ψ to a state q_1

Alternating parity tree automata

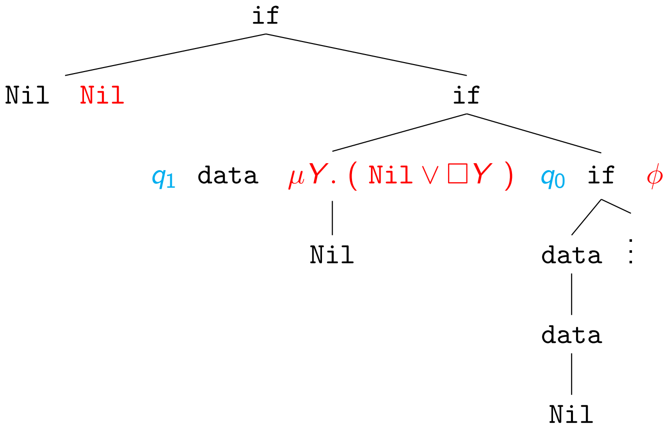
Idea: the formula "starts" on the root



So, *Nil* is accepted from q_1.

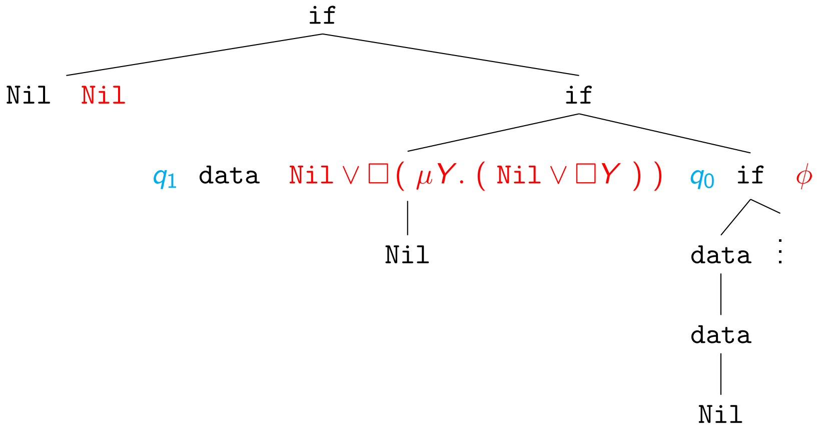
Alternating parity tree automata

Idea: the formula "starts" on the root



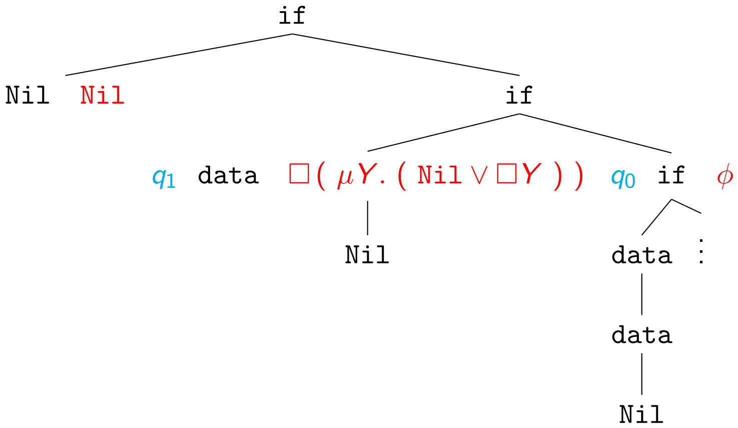
Alternating parity tree automata

Idea: the formula "starts" on the root



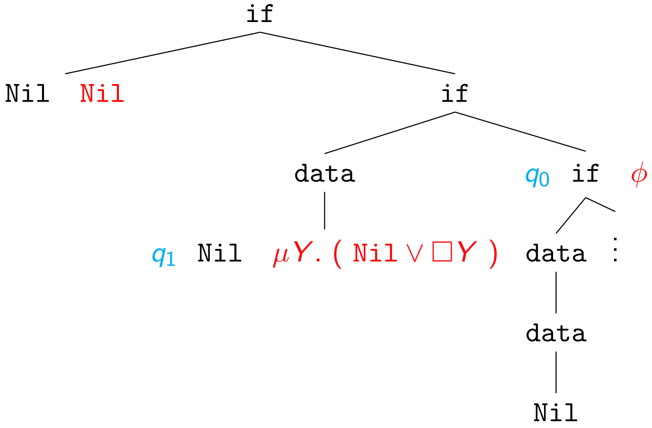
Alternating parity tree automata

Idea: the formula "starts" on the root



Alternating parity tree automata

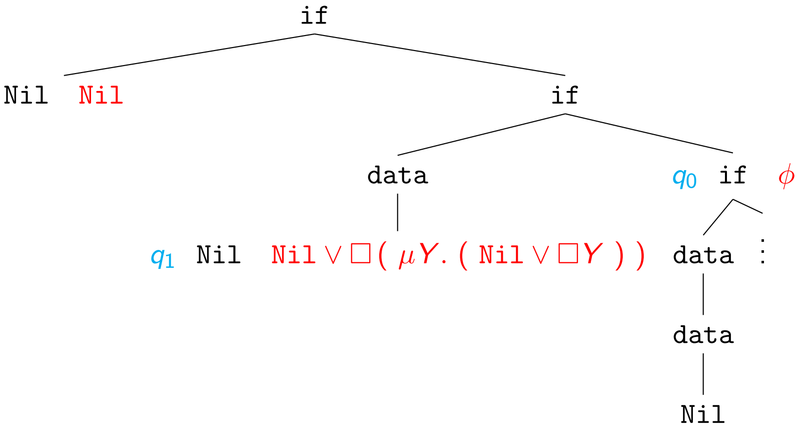
Idea: the formula "starts" on the root



So, reading data from q_1 should propagate q_1.

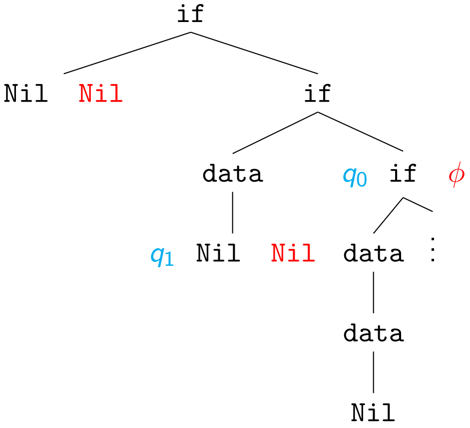
Alternating parity tree automata

Idea: the formula "starts" on the root



Alternating parity tree automata

Idea: the formula "starts" on the root



And the automaton accepts on Nil, and so on.

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 automata

We define first the set $B^+(X)$ of positive Boolean formulas θ over a set X as:

$$\theta \;::=\; \mathit{true} \mid \mathit{false} \mid x \mid \theta \wedge \theta \mid \theta \vee \theta \qquad (x \in X)$$

Alternating parity automata

An alternating parity automaton (APT) $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ is the data

- of a ranked alphabet Σ of symbols of maximal arity n_{max},
- of a finite set of states Q,
- of a transition function $\delta : Q \times \Sigma \to B^+(\{1, \ldots, n_{max}\} \times Q)$, such that

$$\forall a \in \Sigma \; \forall q \in Q \quad \delta(q, a) \in B^+(\{1, \ldots, ar(a)\} \times Q)$$

- of an initial state $q_0 \in Q$, and of a colouring function $\Omega : Q \to \mathbb{N}$.

We will be particularly interested in the set $Col = \Omega(Q)$ of colours of $\mathcal{A}$.

Alternating parity automata

An alternating parity automaton (APT) $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ is the data

- of a ranked alphabet Σ of symbols of maximal arity n_{max},
- of a finite set of states Q,
- of a transition function $\delta : Q \times \Sigma \rightarrow B^+(\{1, \ldots, n_{max}\} \times Q)$, such that

$$\forall a \in \Sigma \ \forall q \in Q \quad \delta(q, a) \in B^+(\{1, \ldots, ar(a)\} \times Q)$$

- of an initial state $q_0 \in Q$, and of a colouring function $\Omega : Q \rightarrow \mathbb{N}$.

We will be particularly interested in the set $Col = \Omega(Q)$ of colours of $\mathcal{A}$.

Alternating parity automata

An alternating parity automaton (APT) $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ is the data

- of a ranked alphabet Σ of symbols of maximal arity n_{max},
- of a finite set of states Q,
- of a transition function $\delta : Q \times \Sigma \rightarrow B^+(\{1, \ldots, n_{max}\} \times Q)$, such that

$$\forall a \in \Sigma \ \forall q \in Q \quad \delta(q, a) \in B^+(\{1, \ldots, ar(a)\} \times Q)$$

- of an initial state $q_0 \in Q$, and of a colouring function $\Omega : Q \rightarrow \mathbb{N}$.

We will be particularly interested in the set $Col = \Omega(Q)$ of colours of $\mathcal{A}$.

Meaning of a transition

The APT has a transition function:

$$\delta(q_0, a) \quad = \quad \bigvee_{i \in I} \bigwedge_{j \in J} (d_{i,j}, q_{i,j})$$

There is first a non-deterministic choice, then alternation.

A clause

$$\bigwedge_{j \in J} (d_j, q_j)$$

means that the automaton runs $|J|$ copies of itself (each in direction d_j), this potentially involving duplication or weakening of subtrees.

Alternating parity tree automata

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

To translate ϕ to an automaton, consider its set of states Q as the set of subformulas of ϕ. Its initial state q_0 corresponds to ϕ, and q_1 to $\mu Y.\,(\,\texttt{Nil} \vee \Box Y\,)$.

Then:

- $\delta(q_0, \texttt{Nil}) \;=\; \bot$
- $\delta(q_0, \texttt{data}) \;=\; \bot$
- $\delta(q_0, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \texttt{Nil}) \;=\; \top$
- $\delta(q_1, \texttt{data}) \;=\; (1, q_1)$
- $\delta(q_1, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_1)$

Inductive/coinductive behaviour limitations: you can only play q_1 finitely, but there are no restrictions over q_0.

Alternating parity tree automata

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

To translate ϕ to an automaton, consider its set of states Q as the set of subformulas of ϕ. Its initial state q_0 corresponds to ϕ, and q_1 to $\mu Y.\,(\;\texttt{Nil} \vee \Box Y\;)$.

Then:

- $\delta(q_0, \texttt{Nil}) \;=\; \bot$
- $\delta(q_0, \texttt{data}) \;=\; \bot$
- $\delta(q_0, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \texttt{Nil}) \;=\; \top$
- $\delta(q_1, \texttt{data}) \;=\; (1, q_1)$
- $\delta(q_1, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_1)$

Inductive/coinductive behaviour limitations: you can only play q_1 finitely, but there are no restrictions over q_0.

Alternating parity tree automata

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

To translate ϕ to an automaton, consider its set of states Q as the set of subformulas of ϕ. Its initial state q_0 corresponds to ϕ, and q_1 to $\mu Y.\,(\;\texttt{Nil} \vee \Box Y\;)$.

Then:

- $\delta(q_0, \texttt{Nil}) \;=\; \bot$
- $\delta(q_0, \texttt{data}) \;=\; \bot$
- $\delta(q_0, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \texttt{Nil}) \;=\; \top$
- $\delta(q_1, \texttt{data}) \;=\; (1, q_1)$
- $\delta(q_1, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_1)$

Inductive/coinductive behaviour limitations: you can only play q_1 finitely, but there are no restrictions over q_0.

Alternating parity tree automata

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

To translate ϕ to an automaton, consider its set of states Q as the set of subformulas of ϕ. Its initial state q_0 corresponds to ϕ, and q_1 to $\mu Y. \, (\, \texttt{Nil} \vee \Box Y \,)$.

Then:

- $\delta(q_0, \texttt{Nil}) \; = \; \bot$
- $\delta(q_0, \texttt{data}) \; = \; \bot$
- $\delta(q_0, \texttt{if}) \; = \; (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \texttt{Nil}) \; = \; \top$
- $\delta(q_1, \texttt{data}) \; = \; (1, q_1)$
- $\delta(q_1, \texttt{if}) \; = \; (1, q_1) \wedge (2, q_1)$

Inductive/coinductive behaviour limitations: you can only play q_1 finitely, but there are no restrictions over q_0.

Alternating parity tree automata

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

To translate ϕ to an automaton, consider its set of states Q as the set of subformulas of ϕ. Its initial state q_0 corresponds to ϕ, and q_1 to $\mu Y.\;(\;\texttt{Nil} \vee \Box Y\;)$.

Then:

- $\delta(q_0, \texttt{Nil}) \;=\; \bot$
- $\delta(q_0, \texttt{data}) \;=\; \bot$
- $\delta(q_0, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \texttt{Nil}) \;=\; \top$
- $\delta(q_1, \texttt{data}) \;=\; (1, q_1)$
- $\delta(q_1, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_1)$

Inductive/coinductive behaviour limitations: you can only play q_1 finitely, but there are no restrictions over q_0.

Alternating parity tree automata

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

To translate ϕ to an automaton, consider its set of states Q as the set of subformulas of ϕ. Its initial state q_0 corresponds to ϕ, and q_1 to $\mu Y.\; (\; \mathtt{Nil} \vee \Box Y \;)$.

Then:

- $\delta(q_0, \mathtt{Nil}) \;=\; \perp$
- $\delta(q_0, \mathtt{data}) \;=\; \perp$
- $\delta(q_0, \mathtt{if}) \;=\; (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \mathtt{Nil}) \;=\; \top$
- $\delta(q_1, \mathtt{data}) \;=\; (1, q_1)$
- $\delta(q_1, \mathtt{if}) \;=\; (1, q_1) \wedge (2, q_1)$

Inductive/coinductive behaviour limitations: you can only play q_1 finitely, but there are no restrictions over q_0.

Alternating parity tree automata

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

To translate ϕ to an automaton, consider its set of states Q as the set of subformulas of ϕ. Its initial state q_0 corresponds to ϕ, and q_1 to $\mu Y.\; (\; \texttt{Nil} \vee \Box Y \;)$.

Then:

- $\delta(q_0, \texttt{Nil}) \;=\; \perp$
- $\delta(q_0, \texttt{data}) \;=\; \perp$
- $\delta(q_0, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \texttt{Nil}) \;=\; \top$
- $\delta(q_1, \texttt{data}) \;=\; (1, q_1)$
- $\delta(q_1, \texttt{if}) \;=\; (1, q_1) \wedge (2, q_1)$

Inductive/coinductive behaviour limitations: you can only play q_1 finitely, but there are no restrictions over q_0.

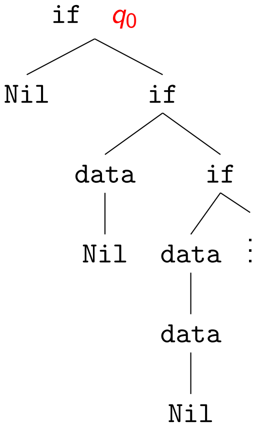
Alternating parity tree automata

In general, transitions may duplicate or drop a subtree.

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

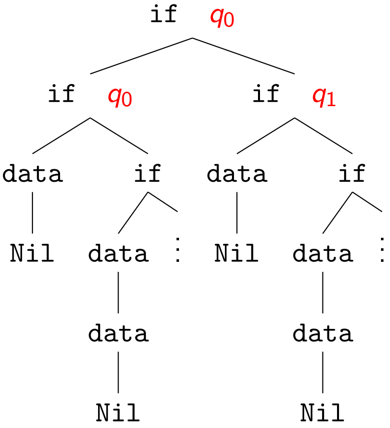
Alternating parity tree automata

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



Alternating parity tree automata

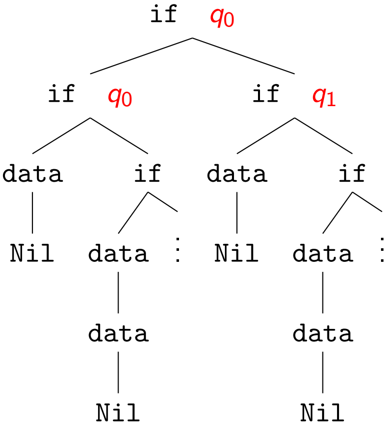
$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1).$



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.

Run-trees: formal definition

Consider a Σ-labelled tree t. A run-tree of $\mathcal{A}$ over t is then a $(\mathbb{N} \times Q)$-labelled unranked tree r such that:

- $\epsilon \in dom(r)$ and $r(\epsilon) = (\epsilon, q_0)$
- $\forall \beta \in dom(r)$, denoting $r(\beta) = (\alpha, q)$, there exists $S \subset \mathbb{N} \times Q$ satisfying $\delta(q,\ t(\alpha))$ and such that $\forall (i, q') \in S,\ \exists j \in \mathbb{N},\ (\beta j \in dom(r)) \wedge (r(\beta j) = (\alpha i,\ q'))$.

Run-trees: formal definition

Remark that alternating tree automata are equivalent to non-deterministic tree automata, yet the translation from alternating to non-deterministic automata makes the size grow.

But there is no determinization result for non-deterministic tree automata.

Alternating parity tree automata

How do we model the inductive/coinductive behaviour of modal μ-calculus properties ? As such, run-trees are purely coinductive...

$\rightarrow$ parity conditions will discriminate a posteriori the trees respecting the inductive semantics of μ

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

How do we model the inductive/coinductive behaviour of modal μ-calculus properties ? As such, run-trees are purely coinductive...

$\rightarrow$ parity conditions will discriminate a posteriori the trees respecting the inductive semantics of μ

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

How do we model the inductive/coinductive behaviour of modal μ-calculus properties ? As such, run-trees are purely coinductive...

$\rightarrow$ parity conditions will discriminate a posteriori the trees respecting the inductive semantics of μ

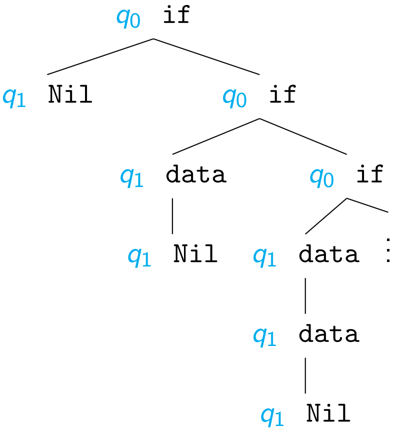
Over a branch of a run-tree, say q_0 has colour 0 and q_1 has colour 1.

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



where $\phi = \nu X.\ (\ \mathtt{if} \wedge \diamond_1\ (\ \mu Y.\ (\ \mathtt{Nil} \vee \Box Y\ )\ ) \wedge \diamond_2\ X\ )$ corresponds to q_0, and q_1 to $\psi = \mu Y.\ (\ \mathtt{Nil} \vee \Box Y\ )$.

Accepting run-trees

Consider an infinite branch $b = i_0 \cdots i_n \cdots$ of a run-tree r and set $m_n = \Omega(\pi_2(r(i_0 \cdots i_n)))$ where π_2 is the projection giving the state labelling a run-tree.

This branch is accepting (or winning) if the greatest colour among the ones occuring infinitely often in the list $(m_n)_{n \in \mathbb{N}}$ is even.

A run-tree is accepting (or winning) iff every infinite branch is winning.

Accepting run-trees and μ-calculus

Formally, given a formula, one considers its quantifier depth – the number of quantifiers alternances.

Examples:

- $\mu X.\ a \vee (\nu Y.b \wedge (\nu Z.c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$
  has one alternance of quantifiers (a μ, then some ν)

- $\mu X.\ a \vee (\nu Y.b \wedge (\mu Z.c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$
  has two alternances of quantifiers (a μ, then a ν, then a μ again)

Accepting run-trees and μ-calculus

Formally, given a formula, one considers its quantifier depth – the number of quantifiers alternances.

Examples:

- $\mu X.\ a \vee (\nu Y.b \wedge (\nu Z.c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$
  has one alternance of quantifiers (a μ, then some ν)

- $\mu X.\ a \vee (\nu Y.b \wedge (\mu Z.c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$
  has two alternances of quantifiers (a μ, then a ν, then a μ again)

Accepting run-trees and μ-calculus

$\mu X.\ a \vee (\nu Y.b \wedge (\nu Z.c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$

will be encoded such that the states corresponding to subformlas under the immediate scope of νY or νZ have colour 0, while the ones under immediate scope of μX will have colour 1.

So, if the maximal colour seen infinitely often is 1, it means that μ was unravelled infinitely: it is forbidden and thus the run-tree is non-accepting (loosing).

Accepting run-trees and μ-calculus

$$\mu X.\ a \vee (\nu Y. b \wedge (\nu Z. c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$$

will be encoded such that the states corresponding to subformlas under the immediate scope of νY or νZ have colour 0, while the ones under immediate scope of μX will have colour 1.

So, if the maximal colour seen infinitely often is 1, it means that μ was unravelled infinitely: it is forbidden and thus the run-tree is non-accepting (loosing).

Accepting run-trees and μ-calculus

$$\mu X.\ a \vee (\nu Y.b \wedge (\mu Z.c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$$

needs more colours, due to its higher quantifier depth.

Moreover, we need to start with colour 1 for μZ, so that νY will have colour 2, and μX colour 3.

Note that infinitely many instances of the colour 1 may occur in a perfectly valid run-tree: if the maximal colour seen infinitely often is 2, it means that μZ was called an infinite number of times, and eventually ceased looping at each call (else νY would have stopped being called and the maximal infinitely occuring colour would not be 2).

Accepting run-trees and μ-calculus

$$\mu X.\ a \vee (\nu Y.b \wedge (\mu Z.c \vee \diamond Z) \wedge \Box Y) \vee \diamond_1 X$$

needs more colours, due to its higher quantifier depth.

Moreover, we need to start with colour 1 for μZ, so that νY will have colour 2, and μX colour 3.

Note that infinitely many instances of the colour 1 may occur in a perfectly valid run-tree: if the maximal colour seen infinitely often is 2, it means that μZ was called an infinite number of times, and eventually ceased looping at each call (else νY would have stopped being called and the maximal infinitely occuring colour would not be 2).

APT and μ-calculus

The connection we sketched can be fully formalized, so that

a modal μ-calculus (or MSO) formula stands at the root of an infinite tree

iff

the associated APT has a winning run-tree over it

Model-checking higher-order programs

(reminder)

Verification met semantics with Ong's decidability result (2006):

"It is decidable whether a given MSO formula holds
at the root of the value tree of a higher-order recursion scheme"

Parity games

A parity game $P_G = \langle (V_E \uplus V_A, E), v_0, \Omega \rangle$ is the data

- of a directed graph $G = (V = V_E \uplus V_A, E)$,
- of an initial vertex $v_0 \in V$,
- and of a colouring function $\Omega : V \to \mathbb{N}$.

We say that $v \in V$ is controlled by Eve if $v \in V_E$, else it is controlled by Adam.

Parity games

A parity game $P_G = \langle (V_E \uplus V_A,\ E),\ v_0,\ \Omega \rangle$ is the data

- of a directed graph $G = (V = V_E \uplus V_A, E)$,
- of an initial vertex $v_0 \in V$,
- and of a colouring function $\Omega : V \to \mathbb{N}$.

We say that $v \in V$ is controlled by Eve if $v \in V_E$, else it is controlled by Adam.

Parity games

A play of P_G is a sequence $\pi = v_0 \cdot v_1 \cdots$ such that $\forall i \ (v_i, v_{i+1}) \in E$.

It is understood as a two-player interaction starting from v_0, and where at each step i the player controlling v_i chooses v_{i+1} according to the edges of G.

A play is maximal if it is finite and ends with a vertice which is source of no edge, or if it is infinite.

The colour of an infinite maximal play is the maximal colour among the ones occuring infinitely often in $(\Omega(v_i))_{i \in \mathbb{N}}$.

A maximal play $\pi = v_0 \cdots v_1 \cdots$ is winning for Eve if it is finite and ends with a node controlled by Adam, or if it is infinite and has an even colour. Else π is winning for Adam.

Parity games

A strategy for Eve is a map σ from the set of plays ending in V_E to V, and such that for every play π ending in V_E $\pi \cdot \sigma(\pi)$ is a play of P_G.

We say that Eve follows σ in the play π if for every prefix π' of π ending in V_E $\pi' \cdot \sigma(\pi')$ is a prefix of π.

If every maximal play in which Eve follows σ is winning for her, we say that σ is a winning strategy. Dual notions are defined for Adam.

Given strategies σ_E for Eve and σ_A for Adam, define their interaction $\langle \sigma_E | \sigma_A \rangle$ as the maximal play starting from v_0 where each player plays its strategy.

Two major facts about parity games

A parity game is determined: on every vertex, one of the players has a winning strategy.

Moreover, this strategy is positional (memoryless), and can be effectively computed (this problem is in $NP \cap co - NP$).

Parity games and APT run-trees

Computing the existence of an accepting run-tree of an APT over a tree t is equivalent to solving a parity game over an arena obtained from t:

- The interaction starts on the root ϵ of t with state q_0, labelled with a symbol a. The APT has a transition function:

$$\delta(q_0,\, a) \quad = \quad \bigvee_{i \in I}\ \bigwedge_{j \in J} (d_{i,j},\, q_{i,j})$$

- Eve starts by picking $i \in I$.
- Adam picks $j \in J$, and the game reaches the node $\epsilon \cdot d_{i,j}$ with state $q_{i,j}$. This move plays the colour $\Omega(q_{i,j})$
- ...and so on ...

Adam has a winning strategy iff there is no run-tree or every run-tree is loosing for the parity condition.

Eve has a winning strategy iff there is a winning run-tree.

Parity games and APT run-trees

Computing the existence of an accepting run-tree of an APT over a tree t is equivalent to solving a parity game over an arena obtained from t:

- The interaction starts on the root ϵ of t with state q_0, labelled with a symbol a. The APT has a transition function:

$$\delta(q_0,\, a) \quad = \quad \bigvee_{i \in I} \bigwedge_{j \in J} (d_{i,j},\, q_{i,j})$$

- Eve starts by picking $i \in I$.
- Adam picks $j \in J$, and the game reaches the node $\epsilon \cdot d_{i,j}$ with state $q_{i,j}$. This move plays the colour $\Omega(q_{i,j})$
- ... and so on ...

Adam has a winning strategy iff there is no run-tree or every run-tree is loosing for the parity condition.

Eve has a winning strategy iff there is a winning run-tree.

Parity games and APT run-trees

But this will not give us the decidability result: the value tree of a scheme is non-regular in general...

So, a way to obtain Ong's decidability result is to obtain a regular tree (= a finite graph), and to compute whether Eve has a winning strategy at the root.

This regular tree is the λ-term itself, over which some higher-order version of the APT runs.

These are the key ideas of Ong's 2006 proof.

More generally, investigating the higher-order behaviour of APT is the key of most proofs of the decidability theorem.

Thank you for coming !

Parity games and APT run-trees

But this will not give us the decidability result: the value tree of a scheme is non-regular in general. . .

So, a way to obtain Ong's decidability result is to obtain a regular tree (= a finite graph), and to compute whether Eve has a winning strategy at the root.

This regular tree is the λ-term itself, over which some higher-order version of the APT runs.

These are the key ideas of Ong's 2006 proof.

More generally, investigating the higher-order behaviour of APT is the key of most proofs of the decidability theorem.

Thank you for coming !

Parity games and APT run-trees

But this will not give us the decidability result: the value tree of a scheme is non-regular in general. . .

So, a way to obtain Ong's decidability result is to obtain a regular tree (= a finite graph), and to compute whether Eve has a winning strategy at the root.

This regular tree is the λ-term itself, over which some higher-order version of the APT runs.

These are the key ideas of Ong's 2006 proof.

More generally, investigating the higher-order behaviour of APT is the key of most proofs of the decidability theorem.

Thank you for coming !

Parity games and APT run-trees

But this will not give us the decidability result: the value tree of a scheme is non-regular in general...

So, a way to obtain Ong's decidability result is to obtain a regular tree (= a finite graph), and to compute whether Eve has a winning strategy at the root.

This regular tree is the λ-term itself, over which some higher-order version of the APT runs.

These are the key ideas of Ong's 2006 proof.

More generally, investigating the higher-order behaviour of APT is the key of most proofs of the decidability theorem.

Thank you for coming !