1. Higher-Order Recursion Schemes, Alternating Parity Tree Automata, and Higher-Order Model Checking

Model Checking: Given: Program P, specification S.
Question: $P \models S$?

Here: P = Functional program
represented by higher-order recursion scheme.
S = Modal μ-calculus formula
represented by alternating parity tree automaton
(Emerson & Jutla FOCS '91).

Goal: · Show that
modal μ-calculus model checking
of trees generated by order-n recursion schemes
is n-EXPTIME-complete.
· Result was first established by Ong LICS '06.

1.1 Higher-Order Recursion Schemes

Idea: · A (higher-order recursion) scheme is a grammar
for describing an infinite tree.
· Non-terminals expect parameters (function values)
and hence need a type, called kind to avoid clashes.

Definition:
· The set of kinds K is defined by
$$k ::= o \mid k_1 \rightarrow k_2.$$

Intuitively: • Ground kind σ describes a type.

• Kind $k_1 \to k_2$ describes a function that takes an entity of kind k_1 and returns an entity of kind k_2.

We assume the (function) arrow associates to the right and omit brackets:

$$\sigma \to \sigma \to \sigma = \sigma \to (\sigma \to \sigma)$$
$$\neq (\sigma \to \sigma) \to \sigma.$$

• The number of arguments to a kind is called the arity, with

$$\text{arity}(\sigma) := 0$$
$$\text{arity}(k_1 \to k_2) := \text{arity}(k_2) + 1.$$

So • $\text{arity}(\sigma \to (\sigma \to \sigma)) = 1 + \text{arity}(\sigma \to \sigma)$
$= 1 + 1 + \text{arity}(\sigma) = 1 + 1 + 0 = 2.$

• $\text{arity}((\sigma \to \sigma) \to \sigma) = 1 + \text{arity}(\sigma) = 1 + 0 = 1.$

• The order of a kind defines the functionality of the arguments:

$$\text{ord}(\sigma) := 0$$
$$\text{ord}(k_1 \to k_2) := \max(\text{ord}(k_1) + 1, \text{ord}(k_2)).$$

A first-order kind defines functions acting on values.
A second-order kind defines functions expecting functions as parameters.

So $\cdot\ \mathrm{ord}(o \to (o \to o)) = \max(\mathrm{ord}(o)+1, \mathrm{ord}(o \to o))$

$= \max(0+1, \max(\mathrm{ord}(o)+1, \mathrm{ord}(o)))$

$= \max(1, \max(1,0))$

$= 1.$

$\cdot\ \mathrm{ord}((o \to o) \to o) = \max(\mathrm{ord}(o \to o)+1, \mathrm{ord}(o))$

$= \max(\max(\mathrm{ord}(o)+1, \mathrm{ord}(o))+1, 0)$

$= \max(\max(1,0)+1, 0)$

$= 2.$

<u>Definition</u> (<u>Syntax of higher-order recursion schemes</u>):

A (<u>deterministic</u>) <u>higher-order recursion scheme</u> is a quadruple

$$G = (\Sigma, N, R, S)$$

where

- Σ is a <u>ranked alphabet</u>, a function from a <u>finite</u> set of symbols, called <u>terminals</u>, to kinds of order 0 or 1. // Either ground or functions that expect ground.
- N is a function from a <u>finite</u> set of <u>non-terminals</u> to kinds,
- R is a function from non-terminals to <u>terms</u> of the form $\lambda \tilde{x}.t$, defined below.
- S is a non-terminal, the <u>start symbol</u>. We require that $N(S) = o$.

We usually write $F \in \mathcal{N}$ and $a \in \Sigma$
rather than $F \in dom(\mathcal{N})$ and $a \in dom(\Sigma)$.

- The <u>order</u> of the recursion scheme is the highest order of its non-terminals.

<u>Definition (Terms):</u>

- To define terms, let Υ be a set of <u>kinded symbols</u>,
  a function from symbols to kinds.

  For each kind k, let $\Upsilon^k := \Upsilon^{-1}(k) = \{ s \in dom(\Upsilon) \mid \Upsilon(s) = k \}$
  be the symbols of kind k in Υ.

  The <u>terms $\mathcal{T}^k(\Upsilon)$ of kind k over Υ</u> are defined by simultaneous induction over all kinds:

  (1) $\Upsilon^k \subseteq \mathcal{T}^k(\Upsilon)$.

  (2) $\bigcup_{k_1 \in K} \{ tv \mid t \in \mathcal{T}^{k_1 \to k_2}(\Upsilon), v \in \mathcal{T}^{k_1}(\Upsilon) \} \subseteq \mathcal{T}^{k_2}(\Upsilon)$.

  (3) $\{ \lambda x.t \mid x \in \mathcal{T}^{k_1}(\Upsilon), t \in \mathcal{T}^{k_2}(\Upsilon) \} \subseteq \mathcal{T}^{k_1 \to k_2}(\Upsilon)$.

- If a term t is of kind k, i.e. $t \in \mathcal{T}^k(\Upsilon)$,
  we also write $t : k$.
- We use $\mathcal{T}(\Upsilon)$ for the set of all terms over Υ.
- We say a term is <u>λ-free</u>, if it does not contain a subterm of the form $\lambda x.t$.
- A term is <u>closed</u>, if all occurring variables (will appear in a moment)
  are <u>bound</u> by a preceding λ-expression.

Back to the assignment of terms to non-terminals:

If $\mathcal{N}(F) = k_1 \to \ldots \to k_n \to o$

then

- $\mathcal{R}(F) = \lambda x_1 \ldots \lambda x_n . t$
- $x_1 : k_1, \ldots, x_n : k_n$
- t is λ-free
- $t \in \mathcal{T}^o(\Sigma \cup N \cup \{x_1 : k_1, \ldots, x_n : k_n\})$.

In particular, the rhs of a non-terminal is
↳ variable-closed and
↳ t is ground.

Example:

Consider the scheme $G = (\Sigma, N, R, S)$

with $\Sigma = \{ a : o \to o \to o, b : o \to o, c : o \}$

$\mathcal{N} = \{ S : o, F : o \to o \}$

$R = \{ S \mapsto Fc, F \mapsto \lambda x. \, a \, x \, (F(bx)) \}$.

Definition (Rewriting relation):

Let G be a scheme.

It induces a rewriting relation, also called reduction relation, on terms — intuitively by following the rules in R.

- A context is a term $C[\bullet] \in \mathcal{T}(T \cup \{\bullet : o\})$ in which $\bullet$ occurs precisely once.

Given a context $C[\bullet]$ and a term $t:o$,
we obtain $C[t]$ by replacing
the unique occurrence of $\bullet$ by t.

- With this,

$t \Rightarrow_G t'$, if there is
- a context $C[\bullet]$,
- a rule $R(F) = \lambda x_1 \ldots x_n . e$,
- and a term $F\, t_1 \ldots t_n : o$,

so that $t = C[F\, t_1 \ldots t_n]$
and $t' = C[e[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]]$.

In short, we replace F in t by the rhs of the rule,
while properly instantiating variables.
We call $F\, t_1 \ldots t_n$ a <u>reducible</u> <u>expression</u> (<u>redex</u>).

- The rewriting step is <u>outermost</u> <u>to</u> <u>innermost</u> ($\underline{OI}$),
if there is no redex that contains
the rewritten one as a proper subterm.

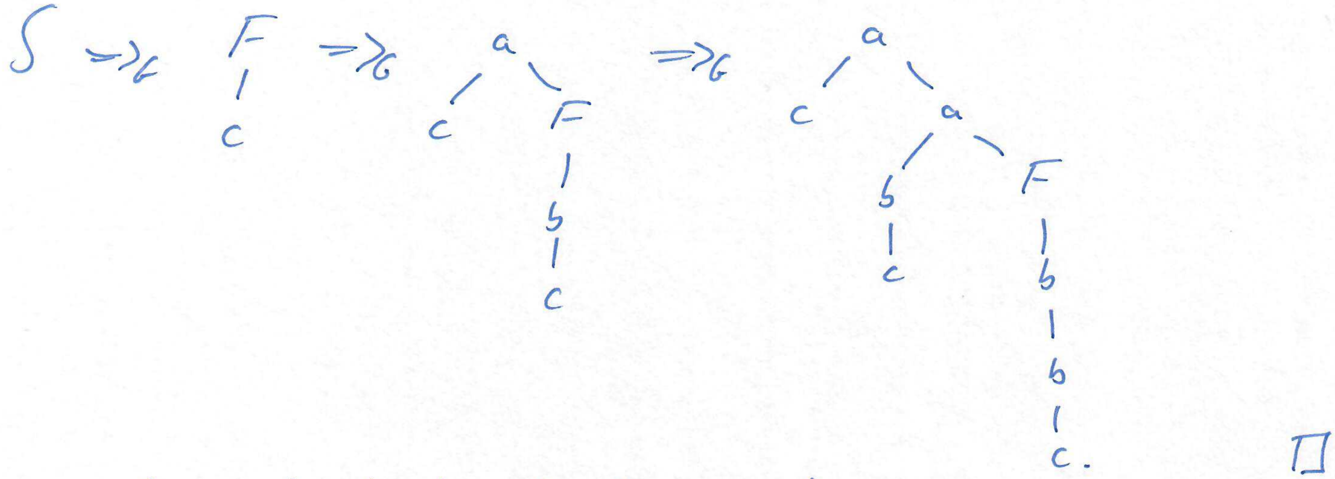
Alternative definition
- with contexts left implicit
- but structural induction over terms (SOS, Plotkin '81):

- $F\tilde{s} \Rightarrow_G t[\tilde{x} \mapsto \tilde{s}]$, if $R(F) = \lambda \tilde{x}.t$
- If $t \Rightarrow_G t'$, then $ts \Rightarrow_G t's$ and $st \Rightarrow_G st'$.

Example (cont.):

$$S \Rightarrow_G Fc \Rightarrow_G a\ c\ (F(bc))$$
$$\Rightarrow_G a\ c\ (a\ (bc)\ (F(b(bc)))).$$

Better understood in terms of trees:



To use the understanding of terms as trees, we have to define trees.

Definition:

Let Δ be a set of symbols.

- A *Δ-labelled tree* is a partial function

  $$t : \{1, \dots, n\}^* \rightharpoonup \Delta$$
  (some fixed n)

  so that $dom(t)$ is prefix-closed.

  Note that t is *unranked*.
  Nodes labelled by $a \in \Delta$ may have a different number of children.
- If Δ is a ranked alphabet, we require n to be the largest arity of a symbol in Δ

In this case, we say that t itself is <u>ranked</u>, if whenever $t(w) = a$ and $\mathrm{arity}(\Delta(a)) = m$, then

$$\{ i \mid w.i \in \mathrm{dom}(t) \} = \{1, \dots, m\}.$$

- A possibly infinite sequence π over $\{1, \dots, m\}$ is a <u>path</u> in t, if every finite prefix of π is in $\mathrm{dom}(t)$.

<u>Notation:</u>

We write trees as terms:

$$\{ \varepsilon \mapsto a,\ 1 \mapsto c,\ 2 \mapsto b,\ 2.1 \mapsto c \}$$

$$= \begin{tikzcd}[column sep=small] & a \arrow[dl, dash] \arrow[dr, dash] & \\ c & & b \arrow[d, dash] \\ & & c \end{tikzcd}$$

$$= a\ c\ (b\ c).$$

Our goal is to define the infinite tree obtained by indefinite rewriting.
This needs an auxiliary definition.

<u>Definition:</u>

- Given a term t, we define the finite tree $t^\perp$ by

$$t^\perp := \begin{cases} f, & \text{if } t \text{ is a terminal } f \\ t_1^\perp\, t_2^\perp, & \text{if } t \text{ is of the form } t_1 t_2 \text{ with } t_1^\perp \neq \perp \\ \perp, & \text{otherwise.} \end{cases}$$

For example, $(f\ (Fa)\ b)^\perp = ((f\ (Fa))\ b)^\perp$
$= (f\ (Fa))^\perp\ b^\perp$

$$= (f^{\perp}\ (Fa)^{\perp})\ b$$
$$= (f\ \perp)\ b$$
$$= f\ \perp\ b.$$

- Let $\sqsubseteq$ be the partial order (refl., trans., antisym.) on $dom(\Sigma) \cup \{\perp\}$ defined by
$$\perp \sqsubseteq a \quad \text{for all } a \in dom(\Sigma).$$
We extend it to a partial order on trees:
$$t \sqsubseteq s, \text{ if } \forall w \in dom(t).\ w \in dom(s) \wedge t(w) \sqsubseteq s(w).$$
For example,
$$\perp \sqsubseteq f \perp \perp \sqsubseteq f \perp b \sqsubseteq f\ a\ b.$$
- A <u>directed set</u> in a partial order (for us the partial order on trees) is a subset T so that
$$\forall t_1, t_2 \in T\ \exists t \in T:\ t_1 \sqsubseteq t \wedge t_2 \sqsubseteq t.$$
The partial order is <u>directed-complete</u>, if the least upper bound (also called join) $\bigsqcup T$ of every countable directed subset T is again in the partial order.
The partial order is <u>pointed</u>, if it has a least element $\perp$.
A pointed, directed-complete partial order is often just referred to as a <u>complete partial order</u>.
A function f on the partial order is <u>$\sqcup$-continuous</u>, if for every countable chain $t_0 \sqsubseteq t_1 \sqsubseteq \dots$ we have
$$f(\textstyle\bigsqcup_{i\in\mathbb{N}} t_i) = \bigsqcup_{i\in\mathbb{N}} f(t_i).$$

Theorem (Kleene):

Consider a complete partial order $(D, \sqsubseteq)$ and a $\sqcup$-continuous function $f : D \to D$.
Then f has a least fixed point, namely

$$lfp(f) = \bigsqcup_{i \in \mathbb{N}} f^i(\bot).$$

In the above setting, one can show that the trees over $dom(\Sigma) \cup \{\bot\}$ form a complete partial order.
Moreover, iterative rewriting yields a countable directed set (even a chain).
Hence, the following definition is guaranteed to yield a tree (exists).
The definition can be phrased as a least fixed point.

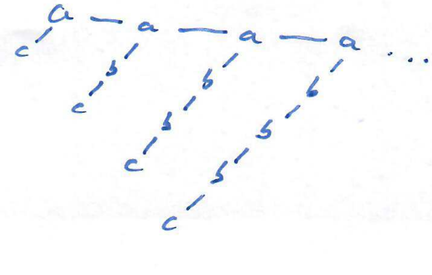
Definition (Semantics of a higher-order recursion scheme):

Let $G = (\Sigma, N, R, S)$.
Its semantics, the tree generated by G, also called the value tree of G, is

$$[\![G]\!] := \bigsqcup \{ t^\bot \mid S \Rightarrow_G^* t \}.$$

By construction, $[\![G]\!]$ is a possibly infinite, ranked, $(dom(\Sigma) \cup \{\bot\})$-labelled tree.
There is a remark below on the use of $\bot$.

Example: $[\![G]\!] =$



1.2 Alternating Parity Tree Automata

Goal: Define the model.

Idea: · Automaton that acts on infinite trees (top down).

· Acceptance by parity condition (priority that repeats indefinitely).

· Alternation to simultaneously check several constraints.

Definition:

Let X be a finite set.

· The set $\mathcal{B}^+(X)$ of positive Boolean formulas over X is defined by

$$\theta ::= true \mid false \mid x \mid \theta \wedge \theta \mid \theta \vee \theta,$$

where $x \in X$.

· A subset $Y \subseteq X$ satisfies θ, if assigning true to the elements in Y and false to the elements in $X \setminus Y$ makes θ true.

Definition (Syntax of alternating parity tree automata):

An alternating parity tree automaton (APTA) over Σ-labelled trees is a tuple

$$A = (\Sigma, Q, \delta, q_I, \Omega),$$

where · Σ is a ranked alphabet (finite), let m be the largest arity of a terminal symbol,

· Q is a finite set of states with $q_I \in Q$ the initial state,

· $\delta : Q \times \Sigma \longrightarrow \mathcal{B}^+(\{1, \dots, m\} \times Q)$

is the transition function, satisfying

$$\forall q \in Q\ \forall f \in \Sigma :\ \delta(q, f) \in \mathcal{B}^+(\{1, \dots, \mathrm{arity}(\Sigma(f))\} \times Q).$$

- $\Omega : Q \to \{0, \dots, k-1\}$ is the <u>priority function</u> used to define acceptance.

<u>Definition (Semantics of alternating parity tree automata)</u>:

- A <u>run tree</u> of an APTA A over a Σ-labelled tree t is a $(\mathrm{dom}(t) \times Q)$-labelled, unranked tree r so that
  - $\varepsilon \in \mathrm{dom}(r)$ and $r(\varepsilon) = (\varepsilon, q_I)$ // Run starts in the initial state.
  - for all $\beta \in \mathrm{dom}(r)$ with $r(\beta) = (\alpha, q)$, there is a set S
    - that satisfies $\delta(q, t(\alpha))$ and
    - for each $(i, q') \in S$ there is j so that $\beta.j \in \mathrm{dom}(r)$ and $r(\beta.j) = (\alpha.i, q')$.

- Let $\pi = \pi_1 \pi_2 \dots$ be an infinite path in r. For each $i \geq 0$, let the state label of node $\pi_1 \dots \pi_i$ be q_{n_i}. Note that for q_{n_0}, the state label of ε, we have $q_{n_0} = q_I$.

- We say that <u>π satisfies the parity condition</u>, if the maximal priority that occurs ∞-often in $\Omega(q_{\pi_0})\,\Omega(q_{\pi_1})\ldots$ is even.

- A run tree is <u>accepting</u>, if every infinite path in it satisfies the priority condition.

- APTA A <u>accepts</u> t, if there is an accepting run tree of A on t.

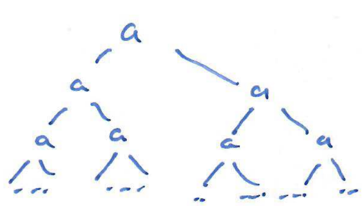
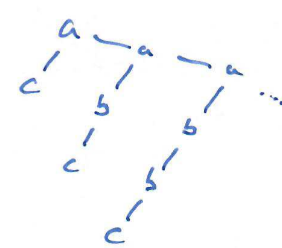
<u>Example (cont.)</u>:

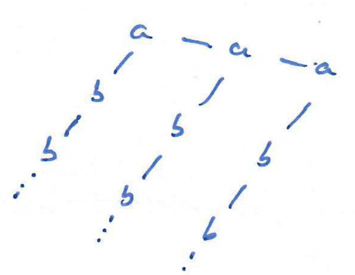
Let Σ be the alphabet in our running example.

- Let $A_1 = (\Sigma, \{q_0, q_1\}, \delta_1, q_0, \{q_0 \mapsto 2, q_1 \mapsto 1\})$, where for each $q \in \{q_0, q_1\}$:

$$\delta_1(q,a) := (1,q) \wedge (2,q)$$
$$\delta_1(q,b) := (1,q_1)$$
$$\delta_1(q,c) := \text{true}.$$

Then A_1 accepts a Σ-labelled tree t iff in every path of t where b occurs c eventually occurs.

↳  accepted,  accepted,

 not accepted.

- Let $A_2 = (\Sigma, \{q_0, q_1\}, \delta_2, q_0, \{q_0 \mapsto 2, q_1 \mapsto 1\})$,
where for all $q \in \{q_0, q_1\}$:

$$\delta_2(q, a) := (1, q_1) \wedge (2, q)$$

$$\delta_2(q, b) := (1, q)$$

$$\delta_2(q, c) := \text{true}.$$

Now A_2 accepts a Σ-labelled tree t iff
for every path of t, if the path takes a left branch
of a node labelled a,
then the path contains a c.

1.3 Higher-Order Model Checking

Goal: Define the algorithmic problem of interest.

HOMC:

Given: Scheme G and APTA A.

Question: Does A accept $[\![G]\!]$?

Theorem (Ong, LICS '06):

HOMC is n-EXPTIME-complete,
for order-n recursion schemes.

- Our goal is to reprove the result in different ways.
- We only consider recursion schemes whose value trees do not contain $\bot$.

Given a scheme G and an APTA A
one can construct G' and A' so that (1) $[\![G']\!]$ does not contain $\bot$
and (2) A accepts $[\![G]\!]$ iff A' accepts $[\![G']\!]$.