Linearity in Higher-Order Recursion Schemes

Pierre Clairambault, Charles Grellois and Andrzej Murawski

Aix-Marseille Université

Séminaire PPS
Dec 7, 2017

Modeling functional programs using higher-order recursion schemes

Model-checking

Approximate the program $\longrightarrow$ build a model $\mathcal{M}$.

Then, formulate a logical specification φ over the model.

Aim: design a program which checks whether

$$\mathcal{M} \;\vDash\; \varphi.$$

That is, whether the model $\mathcal{M}$ meets the specification φ.

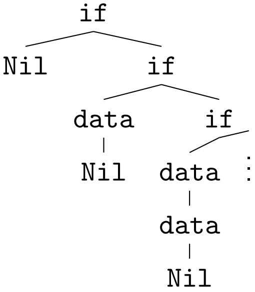
An example

$$
\begin{array}{rcl}
\texttt{Main} & = & \texttt{Listen Nil} \\
\texttt{Listen}\ x & = & \texttt{if end\_signal() then}\ x \\
 & & \texttt{else Listen received\_data()} :: x
\end{array}
$$

An example

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen}\ x & = & \texttt{if end\_signal() then}\ x \\ & & \texttt{else Listen received\_data()}::x \end{array}$$

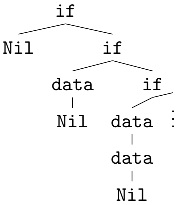
A tree model:



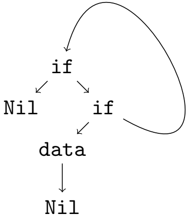
We abstracted conditionals and datatypes.

The approximation contains a non-terminating branch.

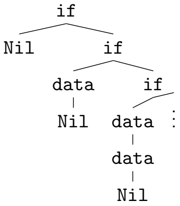
Finite representations of infinite trees



is not regular: it is not the unfolding of a finite graph as



Finite representations of infinite trees



but it is represented by a higher-order recursion scheme (HORS).

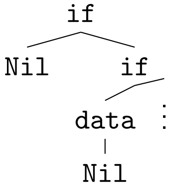
Higher-order recursion schemes

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen } x & = & \texttt{if end\_signal() then } x \\ & & \texttt{else Listen received\_data()} :: x \end{array}$$

is abstracted as

$$\mathcal{G} \;=\; \begin{cases} \texttt{S} & = \quad \texttt{L Nil} \\ \texttt{L } x & = \quad \texttt{if } x \,(\texttt{L } (\texttt{data } x\,)\,) \end{cases}$$

which represents the higher-order tree of actions



Higher-order recursion schemes

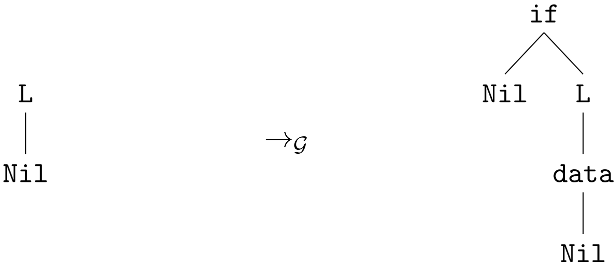
$$\mathcal{G} = \begin{cases} \texttt{S} & = \quad \texttt{L Nil} \\ \texttt{L}\ x & = \quad \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ )\ ) \end{cases}$$

Rewriting starts from the start symbol S:

$$\texttt{S} \quad \to_{\mathcal{G}} \quad \begin{array}{c} \texttt{L} \\ | \\ \texttt{Nil} \end{array}$$

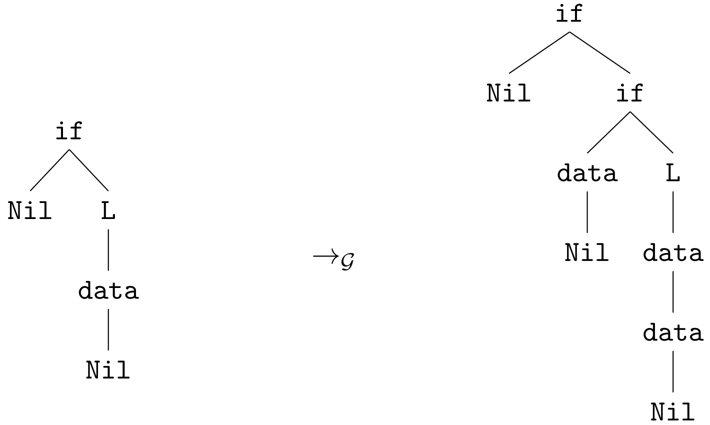
Higher-order recursion schemes

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$



Higher-order recursion schemes

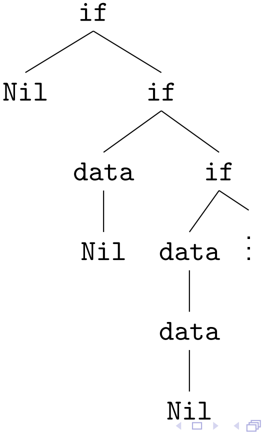
$$\mathcal{G} \;=\; \begin{cases} \mathtt{S} & = \quad \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



Higher-order recursion schemes

$$\mathcal{G} \;=\; \begin{cases} \mathtt{S} & = \quad \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$

$\langle \mathcal{G} \rangle \quad =$



Higher-order recursion schemes

$$\mathcal{G} \;=\; \begin{cases} \mathtt{S} & = \;\; \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \;\; \mathtt{if}\ x\,(\mathtt{L}\ (\mathtt{data}\ x\,)\,) \end{cases}$$

HORS can alternatively be seen as simply-typed λ-terms with

simply-typed recursion operators $Y_\sigma \;:\; (\sigma \rightarrow \sigma) \rightarrow \sigma$.

Alternating parity tree automata

Checking specifications over trees

Monadic second order logic

MSO is a common logic in verification, allowing to express properties as:

" all executions halt "

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

" every time data is added to a buffer, it is eventually processed "

Alternating parity tree automata

Checking whether a formula holds can be performed using an automaton.

For an MSO formula φ, there exists an equivalent APT $\mathcal{A}_\varphi$ s.t.

$$\langle \mathcal{G} \rangle \quad \models \quad \varphi \qquad \text{iff} \qquad \mathcal{A}_\varphi \text{ has a run over } \langle \mathcal{G} \rangle.$$

APT $\quad = \quad$ alternating tree automata (ATA) $+$ parity condition.

Alternating tree automata

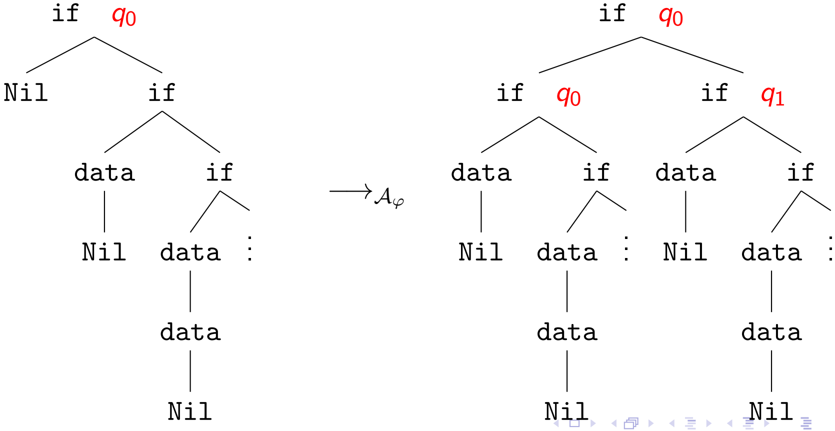
ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

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

Alternating tree automata

ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

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

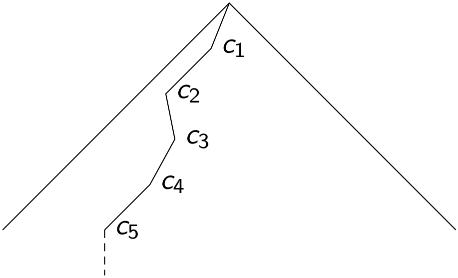


Alternating parity tree automata

Each state of an APT is attributed a color

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

An infinite branch of a run-tree is winning iff the maximal color among the ones occuring infinitely often along it is even.



Alternating parity tree automata

Each state of an APT is attributed a color

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

An infinite branch of a run-tree is winning iff the maximal color among the ones occuring infinitely often along it is even.

A run-tree is winning iff all its infinite branches are.

For a MSO formula φ:

$$\mathcal{A}_\varphi \text{ has a winning run-tree over } \langle \mathcal{G} \rangle \qquad \text{iff} \qquad \langle \mathcal{G} \rangle \vDash \varphi.$$

The higher-order model-checking problem

The (local) HOMC problem

Input: HORS $\mathcal{G}$, formula φ.

Output: `true` if and only if $\langle \mathcal{G} \rangle \vDash \varphi$.

Example: φ = " there is an infinite execution "

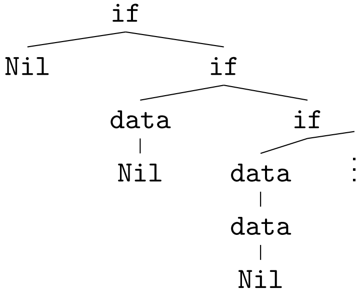
Output: `true`.

The (local) HOMC problem

Input: HORS $\mathcal{G}$, formula φ.

Output: `true` if and only if $\langle \mathcal{G} \rangle \ \vDash \ \varphi$.

Example: $\varphi \ = \ $ " there is an infinite execution "



Output: `true`.

Our line of work

This problem is decidable (Ong 2006), and its complexity is n-EXPTIME where n is the order of the HORS of interest.

But there are practical algorithms that work quite well!

Our contributions:

- Explain why it works: in fact, complexity depends on the linear order of the HORS
- For this, we introduce a linear-nonlinear version of HORS and of APT. This framework allows us to give simpler proofs of existing results of HOMC, and allows to unify these existing approaches.

Intersection types and alternation

A first connection with linear logic

Alternating tree automata and intersection types

A key remark (Kobayashi 2009):

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

can be seen as the intersection typing

$$\texttt{if} \; : \; \emptyset \rightarrow (q_0 \wedge q_1) \rightarrow q_0$$

refining the simple typing

$$\texttt{if} \; : \; o \rightarrow o \rightarrow o$$

Alternating tree automata and intersection types

In a derivation typing the tree $\texttt{if}\ \ T_1\ \ T_2$:

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \texttt{if} : \emptyset \to (q_0 \wedge q_1) \to q_0}\delta \qquad \color{red}{\emptyset}}{\emptyset \vdash \texttt{if}\ T_1 : (q_0 \wedge q_1) \to q_0}\text{App} \qquad \dfrac{\vdots}{\color{red}{\emptyset \vdash T_2 : q_0}} \qquad \dfrac{\vdots}{\color{red}{\emptyset \vdash T_2 : q_1}}}{\emptyset \vdash \texttt{if}\ T_1\ T_2 : q_0}\text{App}$$

Intersection types naturally lift to higher-order – and thus to $\mathcal{G}$, which finitely represents $\langle \mathcal{G} \rangle$.

Theorem (Kobayashi 2009)

$\vdash \mathcal{G} : q_0$ *iff* *the ATA* $\mathcal{A}_\varphi$ *has a run-tree over* $\langle \mathcal{G} \rangle$.

A closer look at the Application rule

In the intersection type system:

$$\text{App} \qquad \frac{\Delta \vdash t : (\ \theta_1\ \wedge \cdots \wedge\ \theta_n) \rightarrow \theta \qquad \Delta_i \vdash u\ :\ \theta_i}{\Delta\ ,\ \Delta_1\ ,\ \ldots\ ,\ \Delta_n\ \vdash\ t\,u\ :\ \theta}$$

This rule could be decomposed as:

$$\frac{\Delta\ \vdash\ t\ :\ (\bigwedge_{i=1}^{n}\ \theta_i\,) \rightarrow \theta' \qquad \dfrac{\Delta_i\ \vdash\ u\ :\ \theta_i \qquad \forall i \in \{1, \ldots, n\}}{\Delta_1, \ldots, \Delta_n\ \vdash\ u\ :\ \bigwedge_{i=1}^{n}\ \theta_i}\ \text{Right}\ \bigwedge}{\Delta, \Delta_1, \ldots, \Delta_n\ \vdash\ t\ u\ :\ \theta'}$$

A closer look at the Application rule

In the intersection type system:

$$\text{App} \qquad \frac{\Delta \vdash t : (\,\theta_1 \ \wedge \cdots \wedge \ \theta_n) \to \theta \qquad \Delta_i \vdash u \, : \, \theta_i}{\Delta\, ,\, \Delta_1\, ,\, \ldots\, ,\, \Delta_n \ \vdash \ t\, u \, : \, \theta}$$

This rule could be decomposed as:

$$\frac{\Delta \ \vdash \ t \ : \ (\,\bigwedge_{i=1}^{n} \ \theta_i\,) \to \theta' \qquad \dfrac{\Delta_i \ \vdash \ u \ : \ \theta_i \qquad \forall i \in \{1, \ldots, n\}}{\Delta_1, \ldots, \Delta_n \ \vdash \ u \ : \ \bigwedge_{i=1}^{n} \ \theta_i} \ \text{Right} \bigwedge}{\Delta, \Delta_1, \ldots, \Delta_n \ \vdash \ t \ u \ : \ \theta'}$$

A closer look at the Application rule

$$\dfrac{\Delta \;\vdash\; t : (\bigwedge_{i=1}^{n} \; \theta_i) \to \theta' \qquad \dfrac{\Delta_i \;\vdash\; u : \theta_i \qquad \forall i \in \{1, \ldots, n\}}{\Delta_1, \ldots, \Delta_n \;\vdash\; u : \bigwedge_{i=1}^{n} \; \theta_i} \;\text{Right } \bigwedge}{\Delta, \Delta_1, \ldots, \Delta_n \;\vdash\; t \; u : \theta'}$$

Linear decomposition of the intuitionistic arrow:

$$A \Rightarrow B \quad = \quad !\,A \multimap B$$

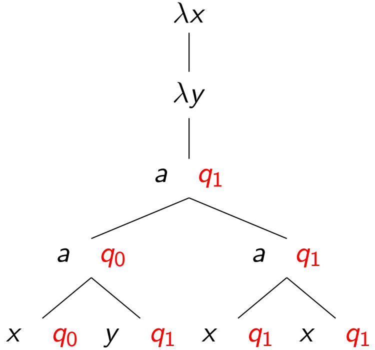
Two steps: duplication / erasure, then linear use.

Right $\bigwedge$ corresponds to the Promotion rule of indexed linear logic.
(see G.-Melliès, ITRS 2014)

Adding parity conditions to the type system

An example of colored intersection type

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



has now type

$$\Box_0\, q_0 \wedge \Box_1\, q_1 \rightarrow \Box_1\, q_1 \rightarrow q_1$$

Note the color 0 on q_0...

A type-system for verification

We devise a type system capturing all MSO:

Theorem (G.-Melliès 2014, from Kobayashi-Ong 2009)

$S : q_0 \vdash S : q_0$ admits a winning typing derivation iff the alternating parity automaton $\mathcal{A}$ has a winning run-tree over $\langle \mathcal{G} \rangle$.

By considering idempotent types, the problem is decidable and n-EXPTIME complete where n is the order of the HORS.

Why n-EXPTIME?

- The types refining o are states. There are $|Q|$ of them.
- The types refining $o \Rightarrow o$ are of the shape $\bigwedge_{i \in I} \square_{c_i} q_i \rightarrow q'$. There are $|Q| \times 2^{|Col| \times |Q|}$ of them.
- The types refining $(o \Rightarrow o) \Rightarrow o$ involve one more powerset construction, and are doubly exponential in size, and so on.

There is a semantic counterpart to this result, in the Scott model of linear logic: every exponential one crosses makes the interpretation of the type exponentially bigger.

Our idea: sometimes, the exponential is not used. Thus, it is not useful to make the search space exponentially bigger! Let's use linear typing to refine all that.

Linear HORS

Linear-Nonlinear Kinds

Kinds are generated by either of φ or ϖ in the following grammar.

$$\begin{array}{rcl} \varphi, \psi, \ldots & ::= & o \mid \varpi \multimap \psi \mid \varphi \to \psi \\ \varpi, \kappa, \iota, \ldots & ::= & \varphi \mid \&_{i \in I} \varphi_i \end{array}$$

(Kind is a word that means "simple type", to distinguish from "intersection type")

Linear Order

The **linear order** $\ell o(\kappa)$ of a kind κ is defined inductively:

$$\begin{array}{rcl}
\ell o(o) & = & 0 \\
\ell o(\varpi \multimap \varphi) & = & \max(\ell o(\varpi), \ell o(\varphi)) \\
\ell o(\varphi \rightarrow \psi) & = & \max(\ell o(\varphi) + 1, \ell o(\psi)) \\
\ell o(\&_{i \in I} \varphi_i) & = & \max_{i \in I} \ell o(\varphi_i)
\end{array}$$

Applicative Terms

A term $t \in \mathrm{KT}_{\Gamma|\Delta}(\kappa)$ is called **applicative** if one can derive $\Gamma \mid \Delta \vdash_{\mathrm{ap}} t :: \kappa$ using:

$$\frac{}{\Gamma, x :: \varphi \mid \Delta \vdash_{\mathrm{ap}} x :: \varphi} \qquad \frac{}{\Gamma \mid \Delta, x :: \&_{i \in I} \varphi_i \vdash_{\mathrm{ap}} \pi_i\, x :: \varphi_i}$$

$$\frac{\Gamma \mid \Delta \vdash_{\mathrm{ap}} t_i :: \varphi_i \qquad (i \in I)}{\Gamma \mid \Delta \vdash_{\mathrm{ap}} \langle t_i \mid i \in I \rangle :: \&_{i \in I} \varphi_i}$$

$$\frac{\Gamma \mid \Delta_1 \vdash_{\mathrm{ap}} t :: \varpi \multimap \varphi \qquad \Gamma \mid \Delta_2 \vdash_{\mathrm{ap}} u :: \varpi}{\Gamma \mid \Delta_1, \Delta_2 \vdash_{\mathrm{ap}} t\, u :: \varphi}$$

$$\frac{\Gamma \mid \Delta \vdash_{\mathrm{ap}} t :: \varphi_1 \to \varphi_2 \qquad \Gamma \mid \_ \vdash_{\mathrm{ap}} u :: \varphi_1}{\Gamma \mid \Delta \vdash_{\mathrm{ap}} t\, u :: \varphi_2}$$

Linear HORS

An applicative term is necessarily $\rhd_{\beta\eta\delta}$-normal.

It does not contain any fixpoints or abstractions, and only consists of pairing and applying (projections of) variables from the contexts.

We write $\mathsf{App}_{\Gamma|\Delta}(\kappa)$ for the set of applicative terms t such that $\Gamma \mid \Delta \vdash_{\mathsf{ap}} t \;::\; \kappa$.

A linear HORS will associate to every non-terminal a term of the form

$$t = \mathsf{l}_1 x_1. \ldots \mathsf{l}_n x_n.\, t' \in \mathsf{KT}_{\Gamma|\_}(\varphi)$$

where $t' \in \mathsf{App}_{\Gamma, V_\lambda | V_\ell}(o)$, $\mathsf{l}_i \in \{\lambda, \ell\}$ and $V_\mathsf{l} = \{x_i \,|\, \mathsf{l}_i = \mathsf{l}\}$.

We call such terms **function definitions of kind φ in context Γ**, and write $\mathsf{Def}_\Gamma(\varphi)$ for the corresponding set.

Linear HORS

A **linear HORS (LHORS)** is a 4-tuple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ where:

- Σ is a tree signature,
- $\mathcal{N}$ is a finite set of kinded **non-terminals**, with a functional kind; we use upper-case letters $F, G, H, \ldots$ to range over them. We denote $\mathcal{N}(F)$ the functional kind of F and write $F :: \mathcal{N}(F)$.
- $S \in \mathcal{N}$ is a distinguished **start symbol** of kind o,
- $\mathcal{R}$ is a function associating to each F in $\mathcal{N}$ a kinded term $\mathcal{R}(F) \in \mathrm{Def}_{\Sigma,\mathcal{N}}(\mathcal{N}(F))$.

Linear Order of a Linear HORS

The **linear order** (*resp.* **linear depth**) of a LHORS $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$, written $\ell o(\mathcal{G})$ (*resp.* $\ell d(\mathcal{G})$), is the maximal linear order (*resp.* linear depth) of the kinds of its non-terminals in $\mathcal{N}$.

Linear HORS: Example

$$\Sigma = \{\mathrm{b} :: o \,\&\, o \multimap o, \mathrm{c} :: o \multimap o, \mathrm{d} :: o \multimap o, \mathrm{e} :: o\}$$

$$\mathcal{N} = \{S :: o, F :: (o \multimap o) \multimap o, G :: o \multimap o, H :: (o \multimap o) \multimap o \multimap o\}$$

$$\begin{array}{rclcl}
\mathcal{R}(S) & = & F\,G & :: & o \\
\mathcal{R}(F) & = & \ell f^{o \multimap o}.\, \mathrm{b}\, \langle f\, \mathrm{e}, F\,(H\,f) \rangle & :: & (o \multimap o) \multimap o \\
\mathcal{R}(G) & = & \ell x^{o}.\, \mathrm{c}\,(\mathrm{d}\,x) & :: & o \multimap o \\
\mathcal{R}(H) & = & \ell f^{o \multimap o}.\, \ell x^{o}.\, \mathrm{c}\,(f\,(\mathrm{d}\,x)) & :: & (o \multimap o) \multimap o \multimap o
\end{array}$$

Linear order: 0

Value Tree of a Linear HORS

Tree contexts:

$$T[-] ::= [-] \mid \mathsf{a}\ t_1\ \ldots\ t_{i-1}\ T[-]\ t_{i+1}\ \ldots\ t_n \mid \langle t_1, \ldots, t_{i-1}, T[-], t_{i+1}, \ldots, t_n \rangle$$

where a is a terminal symbol in Σ. We give a reduction on applicative terms $t \in \mathrm{App}_{\Sigma,\mathcal{N}|\_}(o)$ by:

$$\begin{array}{rcl} T[F\ t_1\ \ldots\ t_n] & \rhd & T[t[t_i/x_i]] \\ T[(\pi_j\ \langle t_i \mid i \in I \rangle)\ u_1\ \ldots\ u_p] & \rhd & T[t_j\ u_1\ \ldots\ u_p] \end{array}$$

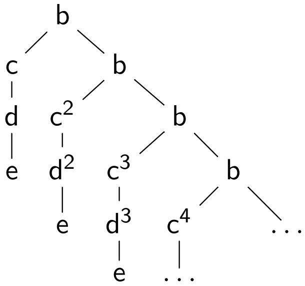
(where $\mathcal{R}(F) \; = \; l_1 x_1. \ldots l_n x_n. t$)

Definition

A linear HORS $\mathcal{G}$ is productive when the limit of any potentially infinite sequence of reductions $S \; \rhd \; \mathcal{R}(S) \; \rhd \; t_2 \; \rhd \; \cdots$ which is fair, *i.e.* which eventually rewrites everything that can be rewritten, exists. This limit is then called the value tree $\langle \mathcal{G} \rangle$ of $\mathcal{G}$.

Value Tree of a Linear HORS

We represent tuples as branching. Tree generated by the previous example:



Generated by a linear HORS of linear order 0.
With usual HORS, order 2 is necessary!

HORS and linear HORS generate the same trees, but with this difference on (linear) order.

An Equivalent Linear-Nonlinear λY-calculus

There is a linear-nonlinear extension of the λY-calculus which is equivalent to linear-nonlinear HORS: there are mutual translations preserving the linear order.

These translations are much simpler than the ones of Salvati and Walukiewicz for the traditional case (mainly because we use products).

Linear-Nonlinear APT

Linear-Nonlinear APT

Recall the correspondence between APT transitions and intersection types.

We will build on it, and consider from now on that an APT is a way to give intersection types to tree constructors. Now we use more refined types:

$$\begin{array}{lcll}
\sigma & ::= & q \mid P \multimap \sigma \mid E \multimap \sigma \mid A \rightarrow \sigma & (q \in Q) \\
A & ::= & \bigwedge_{i \in I} \ \square_{c_i} \ \sigma_i & (c_i \in Col) \\
P & ::= & \langle\, \emptyset, \ldots, \emptyset, \square_c \ \sigma, \emptyset, \ldots, \emptyset \,\rangle & (c \in Col) \\
E & ::= & \langle\, \emptyset, \ldots, \emptyset, \ldots, \emptyset \,\rangle &
\end{array}$$

Idea: a linear-nonlinear APT (LNAPTA) will explore at most one of the components of a product.

Refinement Relation

The **refinement relation** between intersection types and kinds is defined by the following rules.

$$\frac{q \in Q}{q :: o} \qquad \frac{P :: \varpi \qquad \sigma :: \varphi}{P \multimap \sigma :: \varpi \multimap \varphi} \qquad \frac{E :: \varpi \qquad \sigma :: \varphi}{E \multimap \sigma :: \varpi \multimap \varphi}$$

$$\frac{A :: \varphi \qquad \sigma :: \varphi'}{A \rightarrow \sigma :: \varphi \rightarrow \varphi'}$$

$$\frac{\forall i \in I,\ \sigma_i :: \varphi}{\bigwedge_{i \in I} \Box_{c_i}\ \sigma_i :: \varphi} \qquad \frac{\sigma :: \varphi_j}{\left\langle \emptyset, \ldots, \emptyset, \underbrace{\Box_c\ \sigma}_{\text{position } j}, \emptyset, \ldots, \emptyset \right\rangle :: \&_{i \in I} \varphi_i} \qquad \frac{}{E :: \varpi}$$

LNAPTA

A *linear-nonlinear APTA* (LNAPTA) is a tuple $\langle \Sigma, Q, \delta, q_0 \rangle$, where

- Σ is a tree signature,
- Q is a finite set of states,
- $q_0 \in Q$ is the inital state,
- and δ is a map from Σ to sets of intersection types over Q and Col such that $\sigma :: \Sigma(\mathrm{a})$ for any $\mathrm{a} \in \Sigma$ and $\sigma \in \delta(\mathrm{a})$.

LNAPTA: Example

$\Sigma = \{\mathrm{b} :: o \,\&\, o \multimap o, \mathrm{c} :: o \multimap o, \mathrm{d} :: o \multimap o, \mathrm{e} :: o\}$

Check that, on any branch, after a c is encountered, we never encounter b again and we eventually encounter e (call this property P)?

We can only check $\neg P$ over this signature, by $\mathcal{A} = \langle \Sigma,\ \{q_0, q_1\},\ \delta,\ q_0 \rangle$, where:

$$\begin{array}{rcl}
\delta(\mathrm{b}) & = & \{\langle \Box_1 q_0, \emptyset \rangle \multimap q_0, \langle \emptyset, \Box_1 q_0 \rangle \multimap q_0, \langle \emptyset, \emptyset \rangle \multimap q_1\} \\
\delta(\mathrm{c}) & = & \{\langle \Box_2 q_1 \rangle \multimap q_0, \langle \Box_2 q_1 \rangle \multimap q_1\} \\
\delta(\mathrm{d}) & = & \{\langle \Box_1 q_0 \rangle \multimap q_0, \langle \Box_2 q_1 \rangle \multimap q_1\}
\end{array}$$

LNAPTA Run-Tree

We keep informal here. Idea:

- Usual alternating APT on inputs that are nonlinearly typed (i.e. under an exponential)
- On inputs typed with $\&_{i \in I} \varphi_i$, choose either not to explore anything, or to explore a single direction.

The formal definition is by delinearization (idea: remove all the linear information to get back to the usual case).

As such, LNAPTA allow to check for MSO properties over nonlinear signatures, for disjunctive properties over linear signatures,. . . and we do not know a logical characterization of the intermediate classes.

Typing and Model-Checking

We give an intersection type system in which we type the rules of HORS.

$$\frac{\sigma \in \delta(\mathrm{a})}{\_ \mid \_ \vdash_{\mathcal{A}} \mathrm{a} \,:\, \sigma \,::\, \Sigma(\mathrm{a})} \qquad \frac{x \notin \Sigma}{x \,:\, \bigwedge_{\{\star\}} \Box_{\varepsilon}\, \sigma \,::\, \varphi \mid \_ \vdash_{\mathcal{A}} x \,:\, \sigma \,::\, \varphi}$$

$$\frac{x \notin \Sigma \cup \mathcal{N}}{\_ \mid x \,:\, \langle \emptyset, \ldots, \emptyset, \Box_{\varepsilon}\, \sigma, \emptyset, \ldots, \emptyset \rangle \,::\, \&_{i \in I} \varphi_i \vdash_{\mathcal{A}} \pi_i\, x \,:\, \sigma \,::\, \varphi_i}$$

Typing and Model-Checking

$$\frac{\Gamma, x : \bigwedge_{i \in I} \Box_{c_i}\ \sigma_i :: \varphi_1 \mid \Delta \vdash_{\mathcal{A}} t : \tau :: \varphi_2 \qquad I \subseteq J}{\Gamma \mid \Delta \vdash_{\mathcal{A}} \lambda x.\, t : \bigwedge_{j \in J} \Box_{c_j}\ \sigma_j \rightarrow \tau :: \varphi_1 \rightarrow \varphi_2}$$

$$\frac{\Gamma \mid \Delta \vdash_{\mathcal{A}} t : \tau :: \varphi_2 \qquad x \notin dom(()\Gamma, \Delta)}{\Gamma \mid \Delta \vdash_{\mathcal{A}} \lambda x.\, t : \bigwedge \emptyset \rightarrow \tau :: \varphi_1 \rightarrow \varphi_2}$$

$$\frac{\Gamma \mid \Delta, x : P :: \varpi \vdash_{\mathcal{A}} t : \sigma :: \varphi}{\Gamma \mid \Delta \vdash_{\mathcal{A}} \ell x.\, t : P \multimap \sigma :: \varpi \multimap \varphi}$$

$$\frac{\Gamma \mid \Delta \vdash_{\mathcal{A}} t : \sigma :: \varphi \qquad x \notin dom(()\Gamma, \Delta)}{\Gamma \mid \Delta \vdash_{\mathcal{A}} \ell x.\, t : E \multimap \sigma :: \varpi \multimap \varphi}$$

Typing and Model-Checking

$$\frac{\Gamma_1 \mid \Delta_1 \vdash_{\mathcal{A}} t \,:\, \langle \emptyset, \ldots, \emptyset, \Box_c\ \sigma, \emptyset, \ldots, \emptyset \rangle \multimap \tau \ :: \ \varpi \multimap \varphi \qquad \Gamma_2 \mid \Delta_2 \vdash_{\mathcal{A}} u_j \,:\, \sigma \ :: \ \varpi}{\Gamma_1 \wedge \Box_c\ \Gamma_2 \mid \Delta_1, \Box_c\ \Delta_2 \vdash_{\mathcal{A}} t \ \langle\, u_1, \ldots,\ u_j,\ \ldots, u_n \rangle \,:\, \tau \ :: \ \varphi}$$

$$\frac{\Gamma \mid \Delta \vdash_{\mathcal{A}} t \,:\, E \multimap \sigma \ :: \ \varpi \multimap \varphi}{\Gamma \mid \Delta \vdash_{\mathcal{A}} t \ \langle\, u_1, \ldots,\ u_j,\ \ldots, u_n \rangle \,:\, \sigma \ :: \ \varphi}$$

$$\frac{\Gamma \mid \Delta \vdash_{\mathcal{A}} u_j \,:\, \sigma \ :: \ \varphi \qquad j \in I}{\Gamma \mid \Delta \vdash_{\mathcal{A}} \pi_j \ \langle\, u_i \mid i \in I \,\rangle \,:\, \sigma \ :: \ \varphi}$$

$$\frac{\Gamma_0 \mid \Delta \vdash_{\mathcal{A}} t \,:\, \bigwedge_{i \in I} \Box_{c_i}\ \sigma_i \to \tau \ :: \ \varphi_1 \to \varphi_2 \qquad \forall i \in I,\ \ \Gamma_i \mid \_ \vdash_{\mathcal{A}} u \,:\, \sigma_i \ :: \ \varphi_1}{\Gamma_0 \wedge \Box_{c_1}\Gamma_1 \wedge \cdots \wedge \Box_{c_n}\Gamma_n \mid \Delta \vdash_{\mathcal{A}} t\, u \,:\, \tau \ :: \ \varphi_2}$$

Typing and Model-Checking

A parity game accounts for the recursive behaviour.

The idea is that two players, Adam (who owns the vertices from $V_\forall$) and Eve (who owns those from $V_\exists$), build incrementally a typing as follows:

- Eve starts from $(S,\ q_0,\ \varepsilon)$, and must answer with a context Γ such that $\Gamma \mid \_ \vdash_{\mathcal{A}} \mathcal{R}(S)\ :\ q_0\ ::\ o$. Γ contains typings for the nonterminals introduced when rewriting S to $\mathcal{R}(S)$.
- If Γ is empty, Eve wins. Otherwise, Adam picks a typed nonterminal $F\ :\ \square_c \sigma\ ::\ \mathcal{N}(F) \in \Gamma$ and outputs the colour c.
- Then Eve provides a context Γ' such that $\Gamma' \mid \_ \vdash_{\mathcal{A}} \mathcal{R}(F)\ :\ \sigma\ ::\ \mathcal{N}(F)$, and so on.

The interaction stops if Eve can answer with the empty context (she wins), if she cannot answer (she loses) or if the play is infinite (Eve wins iff the parity condition is satisfied).

Typing and Model-Checking

Theorem (Soundness and Completeness)

Let $\mathcal{G}$ be a linear HORS and $\mathcal{A}$ be a LNAPTA. Eve has a winning strategy in the typing game $Typ(\mathcal{G}, \mathcal{A})$ if and only if there is an accepting run-tree of $\mathcal{A}$ over the tree $\langle \mathcal{G} \rangle$ produced by $\mathcal{G}$.

Theorem

Assume $n \geq 1$. The time complexity of checking whether a LNAPTA $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \delta, q_0 \rangle$ accepts the value tree of a D-deep LHORS $\mathcal{G}$ of linear order n is $\exp_n(O(poly(|\mathcal{Q}||\mathcal{G}|)))$. In particular, the problem is n-EXPTIME complete.

Three Applications

Recursive schemes over finite data domains (RSFD) extend HORS with a finite data domain over which pattern-matching can be done.

A direct and elaborate proof exists (Kobayashi et al. 2010) that their MSO model-checking is n-EXPTIME complete. The point is to embed RSFD in usual HORS, but then the complexity becomes too high...

With our framework: a very simple translation to linear-nonlinear λY-calculus, mapping a HORS of order n to a term of linear order n, allows to obtain the result!

Three Applications

Higher-Order Recursion Schemes with Cases (Neatherway et al. 2012) are similar to RSFD, but a bit more general.

Again, by a simple translation, we obtain the (previously known) result that the MSO model-checking problem is n-EXPTIME complete. And we are not impacted by increases of complexity coming from the translation.

Three Applications

What about CBV programs? A 2014 analysis by Tsukada and Kobayashi showed that reachability is n-EXPTIME complete for depth n CBV programs (with recursion and non-determinism).

They do not use a CPS to encode into usual HORS, because it would have make the complexity explode.

We use linear CPS to encode the problem into linear-nonlinear λY-calculus and obtain again the n-EXPTIME completeness result directly from our analysis of HOMC using linearity.

Conclusion

- We refine usual HOMC by introducing linear HORS, linear-nonlinear λY-calculus and LNAPTA
- We show that the complexity of HOMC actually comes from the notion of linear order
- We make three elaborate proofs considerably simpler thanks to the linear framework.