Verification of (probabilistic) functional programs

Charles Grellois

LIS, Aix-Marseille Université

April 9, 2020
Online seminar at PPS

In this talk. . .

- Verification of deterministic functional programs by model-checking, the model being higher-order recursion schemes (HORS)
- Probabilistic functional programs: termination analysis as a first step towards verification:
  - using a type system
  - using a model, *probabilistic* HORS (abbreviated as PHORS)

Higher-order programs, probabilistic programs

- Higher-order (HO) : a function can take functions as inputs, which can themselves take functions as inputs, and so on.

$$\texttt{map}\ \varphi\ [0,1,2] \qquad \text{returns} \qquad [\varphi(0),\varphi(1),\varphi(2)].$$

- Probabilistic : a program's behavior will depend on a probability (a coin toss for example)

$$\begin{array}{rcll} M \oplus_p N & \rightarrow & M & \text{with prob. } p \\ & \rightarrow & N & \text{with prob. } 1-p \end{array}$$

Verifying HO programs

Several approaches. Among them:

- Model-checking : approximate the program as a model, and check whether this model satisfies a given specification using a systematic algorithm
- Type theory : we do not approximate the program, but we annotate it, if we can, by informations allowing the verification of the program

We will have a look at both approaches for probabilistic analysis of termination (a first step towards "full" verification).

Before that, let's see we can do for the deterministic case.

Modeling (deterministic) 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

$$\mathcal{M} \models \varphi.$$

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

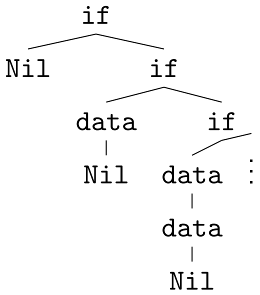
An example

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

An example

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

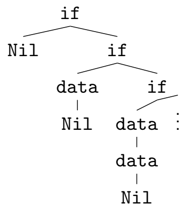
A tree model:



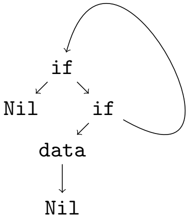
We abstracted conditionals and datatypes.

The approximation contains a non-terminating branch.

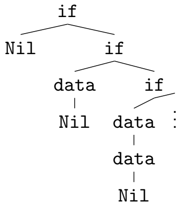
Finite representations of infinite trees



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).

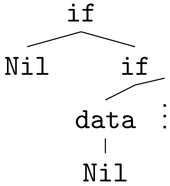
Higher-order recursion schemes

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

is abstracted as

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

which represents the higher-order tree of actions



Higher-order recursion schemes

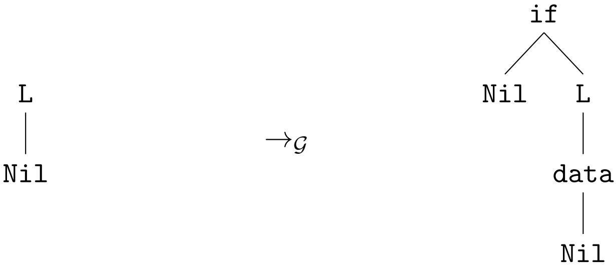
$$\mathcal{G} = \begin{cases} \texttt{S} & = \quad \texttt{L Nil} \\ \texttt{L}\ x & = \quad \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ )\ ) \end{cases}$$

Rewriting starts from the start symbol S:

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

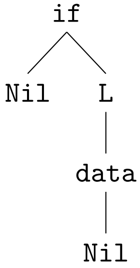
Higher-order recursion schemes

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

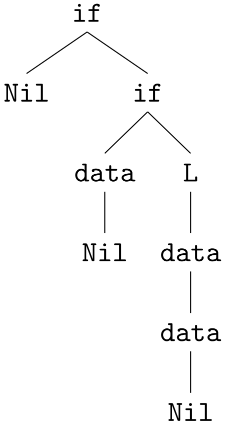


Higher-order recursion schemes

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



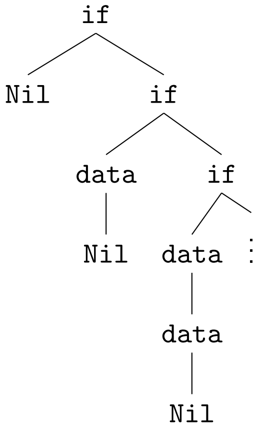
$\rightarrow_{\mathcal{G}}$



Higher-order recursion schemes

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

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



Higher-order recursion schemes

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

can be rewritten in λ-calculus style as

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

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

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

Note that, in general, arguments may be functions of functions of functions. . .

Alternating parity tree automata

Checking specifications over trees

A connection with linear logic

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 ”

MSO notably contains LTL, CTL, PDL. It is equivalent to the modal μ-calculus over trees.

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.

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

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

Alternating tree automata

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

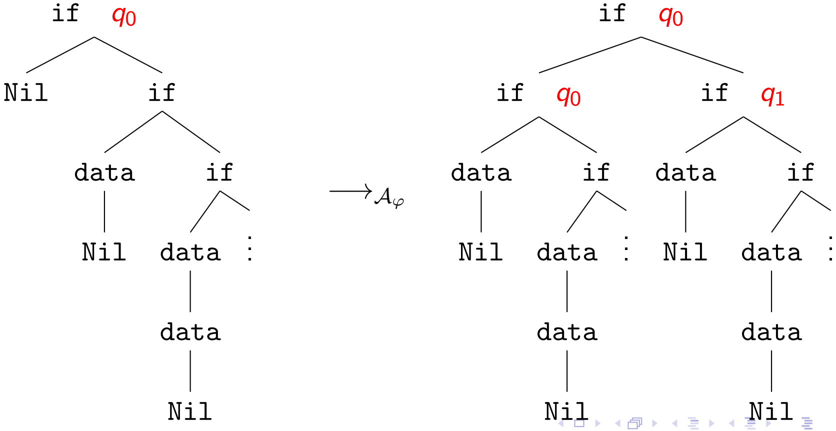
Typically: $\delta(q_0, \texttt{if}) \; = \; (2, q_0) \wedge (2, q_1)$.

A connection with the exponential of linear logic. . .

Alternating tree automata

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

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

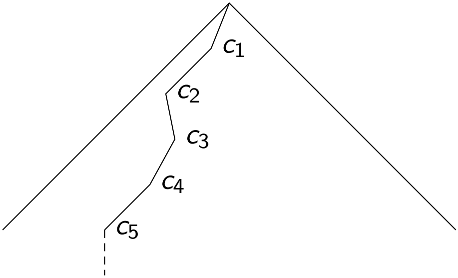


Alternating parity tree automata

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 coloring information will be interpreted using a modality added to linear logic.

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 ”

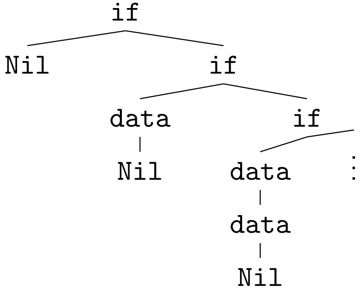
Output: `true`. Note that here we can notably investigate termination properties.

The (local) HOMC problem

Input: HORS $\mathcal{G}$, formula φ.

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

Example: φ = " there is an infinite execution "



Output: `true`. Note that here we can notably investigate termination properties.

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 (with Melliès, Clairambault and Murawski):

- A connection with linear logic and its models, based on a refinment of an intersection type system and on a connection between intersection types and linear logic
- 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.

Overview of our results

Finitary semantics of linear logic

In ScottL (a finitary model of linear logic), we define $\Box$, λ (distributive law) and $\mathbf{Y}$ in an appropriate way.
$ScottL_{\lightning}$ is a model of the λY-calculus.

Theorem

An APT $\mathcal{A}$ has a winning run from q_0 over $\langle \mathcal{G} \rangle$ if and only if

$$q_0 \in [\![\lambda(\mathcal{G})]\!].$$

Corollary

The local higher-order model-checking problem is decidable (and is n-EXPTIME complete).

See Grellois-Melliès: CSL 2015, Fossacs 2015, MFCS 2015, and my thesis.

Linear order and the true complexity of HOMC

Clairambault, G., Murawski, POPL 2018: order isn't the good measure for complexity. We can use the linear order.

Idea: when the automaton doesn't use alternation, complexity doesn't increase that much...

We need to define extensions of HORS and APT: their linear versions.

A big advantage of this framework: allows to reprove several results on HOMC in a much simpler way!

Linear Order

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

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

while the standard notion of order over kinds $\kappa ::= o \mid \kappa \to \kappa$ is:

$$\begin{array}{rcl} ord(o) & = & 0 \\ ord(\varphi \to \psi) & = & \max(ord(\varphi) + 1, ord(\psi)) \end{array}$$

Linear Order

Theorem

Assume $n \geq 1$. The time complexity of checking whether a LNAPTA $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle = \langle \Sigma, \mathcal{Q}, \delta, q_0 \rangle$ 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 call-by-value 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), where depth is an adaptation of order to CBV.

They do not use a CPS to encode into usual HORS, because it would have made the order (and thus 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.

Our result is in fact slightly more general (resource verification in the spirit of (Kobayashi 2009)).

Probabilistic Termination I:

Using Type Theory

Motivations

- Probabilistic programming languages are more and more pervasive in computer science: modeling uncertainty, robotics, cryptography, machine learning, AI...
- Quantitative notion of termination: almost-sure termination (AST)
- AST has been studied for imperative programs in the last years...
- ...but what about the functional probabilistic languages?

We introduce a monadic, affine sized type system sound for AST.

Sized types: the deterministic case

Simply-typed λ-calculus is strongly normalizing (SN).

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}$$

where $\sigma, \tau \ ::= \ o \ \Big| \ \sigma \to \tau$.

Forbids the looping term $\Omega \ = \ (\lambda x.x\ x)(\lambda x.x\ x)$.

Strong normalization: all computations terminate.

Sized types: the deterministic case

Simply-typed λ-calculus is strongly normalizing (SN).

No longer true with the letrec construction...

Sized types: a decidable extension of the simple type system ensuring SN for λ-terms with letrec.

See notably:

- Hughes-Pareto-Sabry 1996, *Proving the correctness of reactive systems using sized types*,
- Barthe-Frade-Giménez-Pinto-Uustalu 2004, *Type-based termination of recursive definitions*.

Sized types: the deterministic case

Sizes: $\mathfrak{s}, \mathfrak{r} \quad ::= \quad \mathfrak{i} \mid \infty \mid \widehat{\mathfrak{s}}$

$+$ size comparison underlying subtyping. Notably $\widehat{\infty} \equiv \infty$.

Idea: k successors $=$ at most k constructors.

- $\mathsf{Nat}^{\widehat{\mathfrak{i}}}$ is 0,
- $\mathsf{Nat}^{\widehat{\widehat{\mathfrak{i}}}}$ is 0 or S 0,
- ...
- Nat^{∞} is any natural number. Often denoted simply Nat.

The same for lists,...

Sized types: the deterministic case

Sizes: $\mathfrak{s}, \mathfrak{r} \quad ::= \quad \mathfrak{i} \mid \infty \mid \widehat{\mathfrak{s}}$

$+$ size comparison underlying subtyping. Notably $\widehat{\infty} \equiv \infty$.

Fixpoint rule:

$$\frac{\Gamma, f \,:\, \mathsf{Nat}^{\mathfrak{i}} \to \sigma \vdash M \,:\, \mathsf{Nat}^{\widehat{\mathfrak{i}}} \to \sigma[\mathfrak{i}/\widehat{\mathfrak{i}}] \qquad \mathfrak{i} \text{ pos } \sigma}{\Gamma \vdash \mathsf{letrec}\ f \ = \ M \,:\, \mathsf{Nat}^{\mathfrak{s}} \to \sigma[\mathfrak{i}/\mathfrak{s}]}$$

"To define the action of f on size $n+1$,
we only call recursively f on size at most n"

Sized types: the deterministic case

Sizes: $\qquad \mathfrak{s}, \mathfrak{r} \quad ::= \quad \mathfrak{i} \mid \infty \mid \widehat{\mathfrak{s}}$

$+$ size comparison underlying subtyping. Notably $\widehat{\infty} \equiv \infty$.

Fixpoint rule:

$$\frac{\Gamma, f : \mathsf{Nat}^{\mathfrak{i}} \to \sigma \vdash M : \mathsf{Nat}^{\widehat{\mathfrak{i}}} \to \sigma[\mathfrak{i}/\widehat{\mathfrak{i}}] \qquad \mathfrak{i} \text{ pos } \sigma}{\Gamma \vdash \mathsf{letrec}\ f = M : \mathsf{Nat}^{\mathfrak{s}} \to \sigma[\mathfrak{i}/\mathfrak{s}]}$$

Sound for SN: typable $\Rightarrow$ SN.

Decidable type inference (implies incompleteness).

A probabilistic λ-calculus

$$
\begin{array}{rcl}
M, N, \ldots & ::= & V \mid V\ V \mid \text{let } x = M \text{ in } N \mid M \oplus_p N \\
& & \mid \text{case } V \text{ of } \{ \mathsf{S} \to W \mid 0 \to Z \}
\end{array}
$$

$$
V, W, Z, \ldots \quad ::= \quad x \mid 0 \mid \mathsf{S}\ V \mid \lambda x.M \mid \text{letrec } f = V
$$

- Formulation equivalent to λ-calculus with $\oplus_p$, but constrained for technical reasons (A-normal form)
- Restriction to base type Nat for simplicity, but can be extended to general inductive datatypes (as in sized types)

A probabilistic λ-calculus: operational semantics

$$\frac{}{\text{let } x \;=\; V \text{ in } M \;\; \to_v \;\; \left\{ (M[x/V])^1 \right\}}$$

$$\frac{}{(\lambda x.M)\;\; V \;\; \to_v \;\; \left\{ (M[x/V])^1 \right\}}$$

$$\frac{}{(\text{letrec } f \;=\; V)\; \left(c \; \overrightarrow{W}\right) \;\; \to_v \;\; \left\{ \left(V[f/\,(\text{letrec } f \;=\; V)] \; \left(c \; \overrightarrow{W}\right)\right)^1 \right\}}$$

A probabilistic λ-calculus: operational semantics

$$\frac{}{\mathsf{case}\ \mathsf{S}\ V\ \mathsf{of}\ \{\,\mathsf{S} \to W \ \mid\ 0 \to Z\,\} \ \to_v\ \left\{ (W\ \ V)^1 \right\}}$$

$$\frac{}{\mathsf{case}\ 0\ \mathsf{of}\ \{\,\mathsf{S} \to W \ \mid\ 0 \to Z\,\} \ \to_v\ \left\{ (Z)^1 \right\}}$$

A probabilistic λ-calculus: operational semantics

$$\frac{}{M \;\oplus_p\; N \quad \rightarrow_v \quad \left\{\, M^p,\, N^{1-p} \,\right\}}$$

$$\frac{M \quad \rightarrow_v \quad \left\{\, L_i^{p_i} \;\middle|\; i \in I \,\right\}}{\text{let } x \;=\; M \text{ in } N \quad \rightarrow_v \quad \left\{\, (\text{let } x \;=\; L_i \text{ in } N)^{p_i} \;\middle|\; i \in I \,\right\}}$$

A probabilistic λ-calculus: operational semantics

$$\frac{\mathscr{D} \stackrel{VD}{=} \left\{ M_j^{p_j} \;\middle|\; j \in J \right\} + \mathscr{D}_V \qquad \forall j \in J,\quad M_j \;\to_v\; \mathscr{E}_j}{\mathscr{D} \;\to_v\; \left(\sum_{j \in J} p_j \cdot \mathscr{E}_j\right) + \mathscr{D}_V}$$

For $\mathscr{D}$ a distribution of terms:

$$[\![\, \mathscr{D} \,]\!] \;=\; \sup_{n \in \mathbb{N}} \left(\left\{ \mathscr{D}_n \;\middle|\; \mathscr{D} \Rrightarrow_v^n \mathscr{D}_n \right\} \right)$$

where $\Rrightarrow_v^n$ is $\to_v^n$ followed by projection on values.

We let $[\![\, M \,]\!] \;=\; [\![\, \{ M^1 \} \,]\!]$.

M is AST iff $\sum [\![\, M \,]\!] = 1$.

Random walks as probabilistic terms

- Biased random walk:

$$M_{bias} \;=\; \left(\text{letrec } f \;=\; \lambda x.\text{case } x \text{ of } \left\{ \mathsf{S} \to \lambda y.f(y) \oplus_{\frac{2}{3}} (f(\mathsf{S}\,\mathsf{S}\,y))) \;\middle|\; 0 \to 0 \right\}\right)\; \underline{n}$$

- Unbiased random walk:

$$M_{unb} \;=\; \left(\text{letrec } f \;=\; \lambda x.\text{case } x \text{ of } \left\{ \mathsf{S} \to \lambda y.f(y) \oplus_{\frac{1}{2}} (f(\mathsf{S}\,\mathsf{S}\,y))) \;\middle|\; 0 \to 0 \right\}\right)\; \underline{n}$$

$$\sum [\![M_{bias}]\!] \;=\; \sum [\![M_{unb}]\!] \;=\; 1$$

Capture this in a sized type system?

Another term

We also want to capture terms as:

$$M_{nat} \;=\; \Big(\text{letrec } f \;=\; \lambda x.x \oplus_{\frac{1}{2}} \mathsf{S}\ (f\ x)\Big)\ 0$$

of semantics

$$[\![\, M_{nat} \,]\!] \;=\; \Big\{ (0)^{\frac{1}{2}}, (\mathsf{S}\ 0)^{\frac{1}{4}}, (\mathsf{S}\ \mathsf{S}\ 0)^{\frac{1}{8}}, \ldots \Big\}$$

summing to 1.

Remark that this recursive function generates the geometric distribution.

Beyond SN terms, towards distribution types

First idea: extend the sized type system with:

$$\text{Choice} \qquad \frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \oplus_p N : \sigma}$$

and "unify" types of M and N by subtyping.

Kind of product interpretation of $\oplus$: we can't capture more than SN...

Beyond SN terms, towards distribution types

First idea: extend the sized type system with:

$$\text{Choice} \qquad \frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M \oplus_p N : \sigma}$$

and "unify" types of M and N by subtyping.

We get at best

$$f \ : \ \mathrm{Nat}^{\widehat{\widehat{i}}} \to \mathrm{Nat}^{\infty} \ \vdash \ \lambda y. f(y) \oplus_{\frac{1}{2}} (f(\mathrm{S}\,\mathrm{S}\,y))) \ : \ \mathrm{Nat}^{\widehat{i}} \to \mathrm{Nat}^{\infty}$$

and can't use a variation of the letrec rule on that.

Beyond SN terms, towards distribution types

We will use distribution types, built as follows:

$$\text{Choice} \quad \frac{\Gamma \mid \Theta \ \vdash \ M : \mu \qquad \Gamma \mid \Psi \ \vdash \ N : \nu \qquad \{\!| \mu |\!\} = \{\!| \nu |\!\}}{\Gamma \mid \Theta \oplus_p \Psi \ \vdash \ M \oplus_p N : \mu \oplus_p \nu}$$

Now

$$f \ : \ \left\{ \left(\mathsf{Nat}^{\mathfrak{i}} \to \mathsf{Nat}^{\infty}\right)^{\frac{1}{2}}, \ \left(\mathsf{Nat}^{\widehat{\widehat{\mathfrak{i}}}} \to \mathsf{Nat}^{\infty}\right)^{\frac{1}{2}} \right\}$$

$$\vdash$$

$$\lambda y. f(y) \oplus_{\frac{1}{2}} (f(\mathsf{S}\,\mathsf{S}\,y))) \ : \ \mathsf{Nat}^{\widehat{\mathfrak{i}}} \to \mathsf{Nat}^{\infty}$$

Designing the fixpoint rule

$$f \ : \ \left\{ \left(\mathsf{Nat}^{\mathfrak{i}} \to \mathsf{Nat}^{\infty}\right)^{\frac{1}{2}},\ \left(\mathsf{Nat}^{\widehat{\widehat{\mathfrak{i}}}} \to \mathsf{Nat}^{\infty}\right)^{\frac{1}{2}} \right\}$$

$$\vdash$$

$$\lambda y.f(y) \oplus_{\frac{1}{2}} (f(\mathsf{S}\,\mathsf{S}\,y))) \ : \ {\color{red}\mathsf{Nat}^{\widehat{\mathfrak{i}}}} \to \mathsf{Nat}^{\infty}$$

induces a random walk on $\mathbb{N}$:

- on $n+1$, move to n with probability $\frac{1}{2}$, on $n+2$ with probability $\frac{1}{2}$,
- on 0, loop.

The type system ensures that there is no recursive call from size 0.

Random walk AST ($=$ reaches 0 with proba 1) $\Rightarrow$ termination.

Designing the fixpoint rule

$$\text{LetRec}\quad \dfrac{\begin{array}{c}\{\!|\,\Gamma\,|\!\} \;=\; \mathsf{Nat} \\ \mathfrak{i} \notin \Gamma \text{ and } \mathfrak{i} \text{ positive in } \nu \\ \color{red}{\left\{\, (\mathsf{Nat}^{\mathfrak{s}_j} \to \nu[\mathfrak{i}/\mathfrak{s}_j])^{p_j} \;\;\middle|\;\; j \in J \,\right\} \text{ induces an AST sized walk}} \\ \Gamma \,|\, f \;:\; \left\{\, (\mathsf{Nat}^{\mathfrak{s}_j} \to \nu[\mathfrak{i}/\mathfrak{s}_j])^{p_j} \;\;\middle|\;\; j \in J \,\right\} \vdash V \;:\; \mathsf{Nat}^{\widehat{\mathfrak{i}}} \to \nu[\mathfrak{i}/\widehat{\mathfrak{i}}]\end{array}}{\Gamma \,|\, \emptyset \vdash \mathsf{letrec}\ f \;=\; V \;:\; \mathsf{Nat}^{\mathfrak{r}} \to \nu[\mathfrak{i}/\mathfrak{r}]}$$

Sized walk: AST is checked by an external PTIME procedure.

Generalized random walks and the necessity of affinity

A crucial feature: our type system is affine.

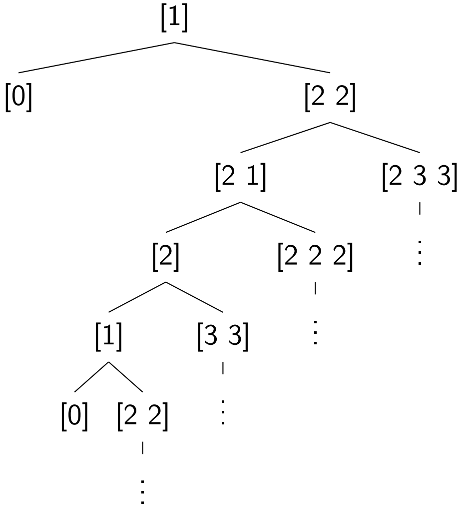
Higher-order symbols occur at most once. Consider:

$$M_{naff} \;=\; \text{letrec } f \;=\; \lambda x.\text{case } x \text{ of } \left\{ \mathsf{S} \to \lambda y.f(y) \oplus_{\frac{2}{3}} (f(\mathsf{S}\,\mathsf{S}\,y)\,;\, f(\mathsf{S}\,\mathsf{S}\,y)) \;\middle|\; 0 \to 0 \right\}$$

The induced sized walk is AST.

Generalized random walks and the necessity of affinity

Tree of recursive calls, starting from 1:



Leftmost edges have probability $\frac{2}{3}$; rightmost ones $\frac{1}{3}$.

This random process is not AST.

Problem: modelisation by sized walk only makes sense for affine programs.

Key properties

A nice subject reduction property, and:

Theorem (Typing soundness)

If $\Gamma \mid \Theta \vdash M : \mu$, then M is AST.

Proof by reducibility, using set of candidates parametrized by probabilities.

See Dal Lago-Grellois ESOP 2017 and ACM TOPLAS 2019 (long version).

Probabilistic Termination II: Using Probabilistic HORS

Termination analysis for PHORS

We extend HORS to probabilistic HORS (PHORS).
Example: random walk:

$$\mathcal{G} \;=\; \begin{cases} \mathtt{S} & = \quad \mathtt{F\ e} \\ \mathtt{F}\ x & = \quad x \oplus_p \mathtt{F\ (\ F}\ x\ \mathtt{)} \end{cases}$$

Probabilistic reduction:

$$S \xrightarrow{1} F\ e \xrightarrow{R,1-p} F(F\ e) \xrightarrow{L,p} F\ e \xrightarrow{L,p} e$$

has probability $(1-p) \times p^2$.

Termination probability: sum of the probabilites of the reductions from S ending in e (after finitely many steps).

Contributions

Kobayashi, Dal Lago, Grellois, LICS 2019:

- Definition of PHORS, of their operational semantics, relation with recursive Markov chains. . .
- (Un)Decidability results: several results among which the undecidability of AST for order ≥ 2
- A fixpoint characterization of the termination probability giving the semi-decidability of the lower bound problem
- A sound procedure (is it complete?) for computing an upper bound of the termination probability for order-2 PHORS.

(Un)decidability results

Unsolvability of Diophantine equations in terms of polynomials with non-negative coefficients:

Given two polynomials $P(x_1; \ldots ; x_k)$ and $Q(x_1; \ldots ; x_k)$ with non-negative integer coefficients, whether $P(x_1; \ldots ; x_k) < Q(x_1; \ldots ; x_k)$ for some $x_1; \ldots ; x_k \in \mathbb{N}$ is undecidable.

Idea: show that for every P and Q as above, one can effectively construct an order-2 PHORS that does not almost surely terminate if and only if $P(x_1; \ldots ; x_k) < Q(x_1; \ldots ; x_k)$ for some $x_1; \ldots ; x_k$.

Start from order 3 (easier) then reduce to order 2 replacing Church numerals with appropriate probabilistic functions.

(Un)decidability results

It is also undecidable:

- whether a given order-2 PHORS $\mathcal{G}$ satisfies $Pr(\mathcal{G}) \geq r$
- whether a given order-2 PHORS $\mathcal{G}$ satisfies $Pr(\mathcal{G}) = r$

Whether a given order-n PHORS $\mathcal{G}$ satisfies $Pr(\mathcal{G}) > r$ is semi-decidable.

Fixpoint characterization

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{F}\ \mathtt{e} \\ \mathtt{F}\ x & = & x \oplus_p \mathtt{F}\ (\ \mathtt{F}\ x\ ) \end{array} \right.$$

becomes

$$\left\{ \begin{array}{lcl} S & = & 1 \times F(1) \\ F(x) & = & px + (1-p) \times\ F(F(x)) \end{array} \right.$$

and

$$\left\{ \begin{array}{lcl} S & = & \frac{p}{1-p} \text{ if } 0 \leq p \leq \frac{1}{2} \\ & = & 1 \text{ if } \frac{1}{2} \leq p \leq 1 \end{array} \right.$$

Fixpoint characterization

Order-n systems can be reduced to order-(n-1) systems (but no more).

The previous system can be reduced to ordinary (order-0) equations, see the paper (Example 4.5).

The probability of termination can be expressed as a least fixpoint over such systems of equations.

So, for the lower bound, we can have approximations by iteration.

Upper bound of a lfp?? roughly, approximate the function as a piecewise affine function.

Also in the paper: experiments that work well, and two exemples of incompleteness to be adressed in future work.

Conclusions and perspectives

- Finer measure of HOMC complexity
- A (very incomplete) type system for checking whether a probabilistic functional program is AST
- First steps on PHORS termination

Some perspectives:

- LTL model-checking for PHORS
- A probabilistic modal mu-calculus for model-checking PHORS?
- Extension of the approximation algorithm for PHORS termination probabilites
- A more complete type system for checking whether a probabilistic program is AST

Thank you for your attention!

Conclusions and perspectives

- Finer measure of HOMC complexity
- A (very incomplete) type system for checking whether a probabilistic functional program is AST
- First steps on PHORS termination

Some perspectives:

- LTL model-checking for PHORS
- A probabilistic modal mu-calculus for model-checking PHORS?
- Extension of the approximation algorithm for PHORS termination probabilites
- A more complete type system for checking whether a probabilistic program is AST

Thank you for your attention!