Verifying properties of functional programs: from the deterministic to the probabilistic case

Charles Grellois
(partly joint with Dal Lago and Melliès)

FOCUS Team – INRIA & University of Bologna

Séminaire Méthodes Formelles
March 7, 2017

Functional programs,

Higher-order models

Imperative vs. functional programs

- Imperative programs: built on finite state machines (like Turing machines).

  Notion of state, global memory.

- Functional programs: built on functions that are composed together (like in Lambda-calculus).

  No state (except in impure languages), higher-order: functions can manipulate functions.

(recall that Turing machines and λ-terms are equivalent in expressive power)

Imperative vs. functional programs

- Imperative programs: built on finite state machines (like Turing machines).

  Notion of state, global memory.

- Functional programs: built on functions that are composed together (like in Lambda-calculus).

  No state (except in impure languages), higher-order: functions can manipulate functions.

(recall that Turing machines and λ-terms are equivalent in expressive power)

Example: imperative factorial

```
int fact(int n) {
  int res = 1;
  for i from 1 to n do {
    res = res * i;
    }
  }
  return res;
}
```

Typical way of doing: using a variable (change the state).

Example: functional factorial

In OCaml:

```
let rec factorial n =
    if n <= 1 then
      1
    else
      factorial (n-1) * n;;
```

Typical way of doing: using a recursive function (don't change the state).

In practice, forbidding global variables reduces considerably the number of bugs, especially in a parallel setting (cf. Erlang).

Advantages of functional programs

- Very mathematical: calculus of functions.
- ...and thus very much studied from a mathematical point of view. This notably leads to strong typing, a marvellous feature.
- Much less error-prone: no manipulation of global state.

More and more used, from Haskell and Caml to Scala, Javascript and even Java 8 nowadays.

Also emerging for probabilistic programming.

Price to pay: analysis of higher-order constructs.

Advantages of functional programs

Price to pay: analysis of higher-order constructs.

Example of higher-order function: `map`.

`map` φ $[0, 1, 2]$ returns $[\varphi(0), \varphi(1), \varphi(2)]$.

Higher-order: `map` is a function taking a function φ as input.

Advantages of functional programs

Price to pay: analysis of higher-order constructs.

- Function calls $+$ recursivity $=$ deal with stacks of calls $\rightarrow$ approaches for verification using automata with stacks of stacks of stacks. . . or with Krivine machines that also have a stack of calls
- Based on λ-calculus with recursion and types: we will use its semantics to do verification

  That's the first goal of the talk.

(but that's only an approach among many others)

Probabilistic functional programs

Probabilistic programming languages are more and more pervasive in computer science: modeling uncertainty, robotics, cryptography, machine learning, AI...

What if we add probabilistic constructs?

In this talk: $M \oplus_p N \rightarrow_v \{ M^p, N^{1-p} \}$

Allows to simulate some random distributions, not all. In future work: add fully the two roots of probabilistic programming, drawing values at random from more probability distributions (typically on the reals), and conditioning which allows among others to do machine learning.

Probabilistic functional programs

Probabilistic programming languages are more and more pervasive in computer science: modeling uncertainty, robotics, cryptography, machine learning, AI...

What if we add probabilistic constructs?

In this talk: $M \oplus_p N \to_v \{ M^p, N^{1-p} \}$

Second goal of the talk. Go towards verification of probabilistic functional programs. We give an incomplete method for termination-checking and hints towards verification of more properties.

Using higher-order functions

Bending a coin in the probabilistic functional language Church:

```
var makeCoin = function(weight) {
  return function() {
    flip(weight) ? 'h' : 't'
  }
}
var bend = function(coin) {
  return function() {
    (coin() == 'h') ? makeCoin(0.7)() : makeCoin(0.1)()
  }
}
var fairCoin = makeCoin(0.5)
var bentCoin = bend(fairCoin)
viz(repeat(100,bentCoin))
```

Roadmap

1. Semantics of linear logic for verification of deterministic functional programs

2. A type system for termination of probabilistic functional programs

3. Towards verification for the probabilistic case?

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

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

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

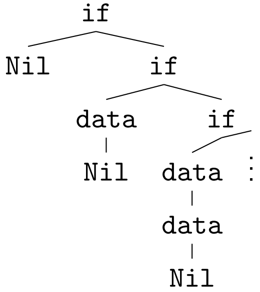
An example

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

An example

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

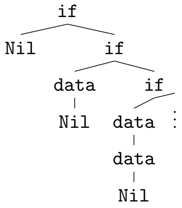
A tree model:



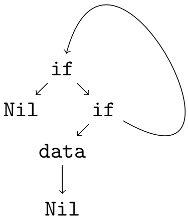
We abstracted conditionals and datatypes.

The approximation contains a non-terminating branch.

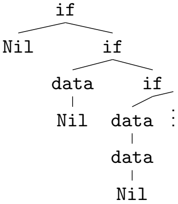
Finite representations of infinite trees



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).

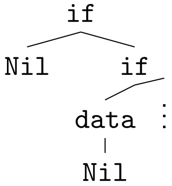
Higher-order recursion schemes

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

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

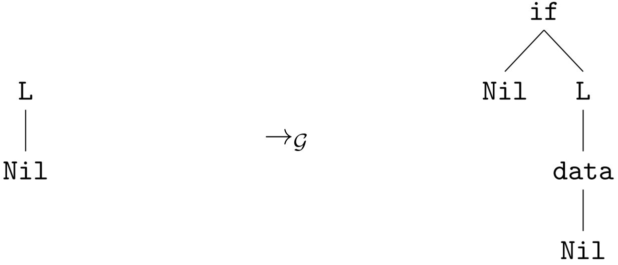
$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \mathtt{S} & = & \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = & \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{array} \right.$$

Rewriting starts from the start symbol S:

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

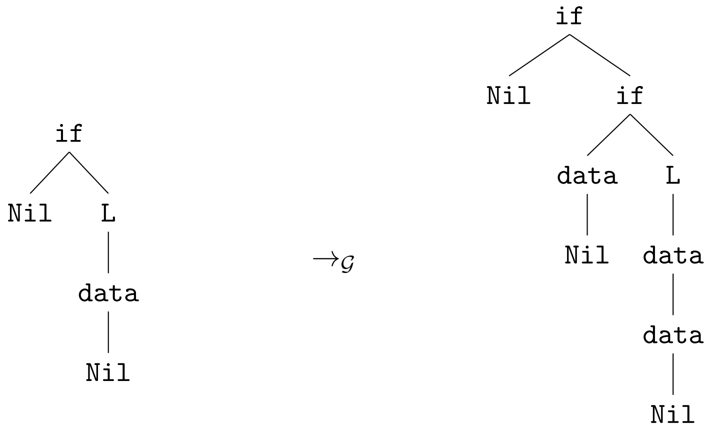
Higher-order recursion schemes

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



Higher-order recursion schemes

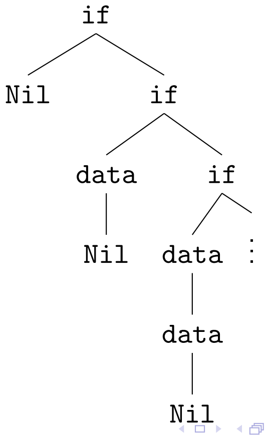
$$\mathcal{G} = \begin{cases} \mathtt{S} & = \quad \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \quad \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



Higher-order recursion schemes

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

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



Higher-order recursion schemes

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

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

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

They are also equi-expressive to pushdown automata with stacks of stacks of stacks... and a collapse operation.

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.

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

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

Alternating tree automata

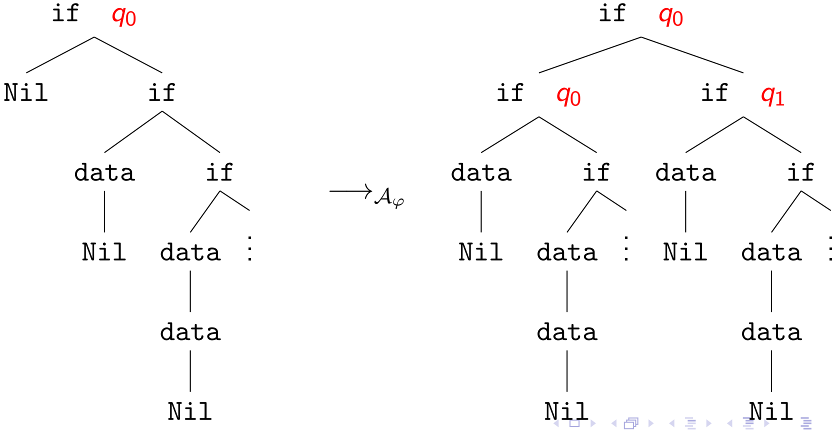
ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

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

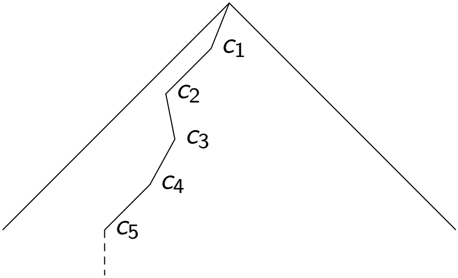


Alternating parity tree automata

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 higher-order model-checking problems

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 "

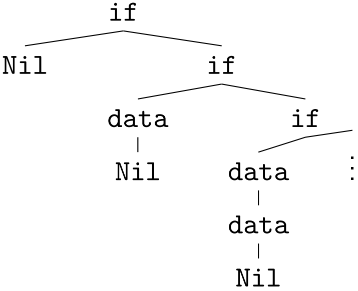
Output: `true`.

The (local) HOMC problem

Input: HORS $\mathcal{G}$, formula φ.

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

Example: φ = " there is an infinite execution "



Output: `true`.

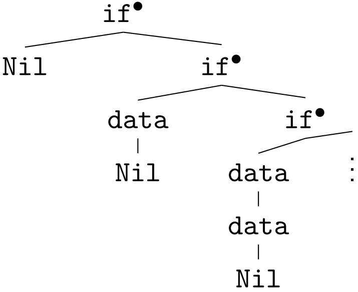
The global HOMC problem

Input: HORS $\mathcal{G}$, formula φ.

Output: a HORS $\mathcal{G}^{\bullet}$ producing a marking of $\langle \mathcal{G} \rangle$.

Example: φ = " there is an infinite execution "

Output: $\mathcal{G}^{\bullet}$ of value tree:



The selection problem

Input: HORS $\mathcal{G}$, APT $\mathcal{A}$, state $q \in Q$.

Output: `false` if there is no winning run of $\mathcal{A}$ over $\langle \mathcal{G} \rangle$.
Else, a HORS $\mathcal{G}^q$ producing a such a winning run.

Example: $\varphi \; = \;$ " there is an infinite execution ", q_0 corresponding to φ

Output: $\mathcal{G}^{q_0}$ producing

$$\begin{array}{c} \mathtt{if}^{q_0} \\ | \\ \mathtt{if}^{q_0} \\ | \\ \mathtt{if}^{q_0} \\ | \\ \vdots \end{array}$$

Our line of work (joint with Melliès)

These three problems are decidable, with elaborate proofs (often) relying on semantics.

Our contribution: an excavation of the semantic roots of HOMC, at the light of linear logic, leading to refined and clarified proofs.

Recognition by homomorphism

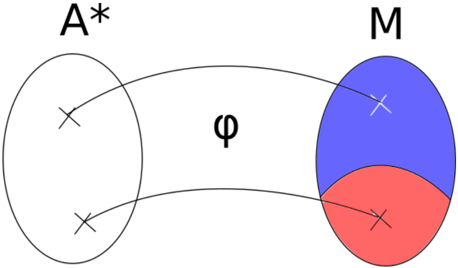
Where semantics comes into play

Automata and recognition

For the usual finite automata on words: given a regular language $L \subseteq A^*$,

there exists a finite automaton $\mathcal{A}$ recognizing L

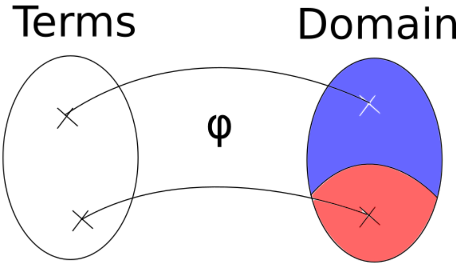
if and only if. . .



there exists a finite monoid M, a subset $K \subseteq M$
and a homomorphism $\varphi : A^* \to M$ such that $L = \varphi^{-1}(K)$.

Automata and recognition

The picture we want:



(after Aehlig 2006, Salvati 2009)

but with recursion and w.r.t. an APT.

Intersection types and alternation

A first connection with linear logic

Alternating tree automata and intersection types

A key remark (Kobayashi 2009):

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

can be seen as the intersection typing

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

refining the simple typing

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

Alternating tree automata and intersection types

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

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

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:

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

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

In the intersection type system:

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

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

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

Linear decomposition of the intuitionistic arrow:

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

Two steps: duplication / erasure, then linear use.

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

Intersection types and semantics of linear logic

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

Two interpretations of the exponential modality:

Qualitative models
(Scott semantics)

$$!\,A \quad = \quad \mathcal{P}_{fin}(A)$$

$$[\![o \Rightarrow o]\!] \;=\; \mathcal{P}_{fin}(Q) \times Q$$

$$\{q_0,\, q_0,\, q_1\} \quad = \quad \{q_0,\, q_1\}$$

Order closure

Quantitative models
(Relational semantics)

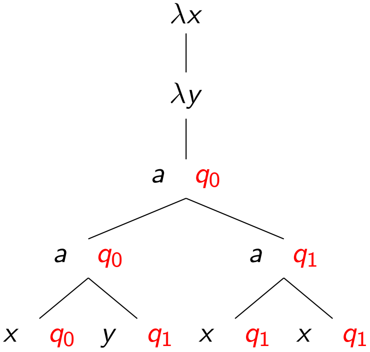
$$!\,A \quad = \quad \mathcal{M}_{fin}(A)$$

$$[\![o \Rightarrow o]\!] \;=\; \mathcal{M}_{fin}(Q) \times Q$$

$$[q_0,\, q_0,\, q_1] \quad \neq \quad [q_0,\, q_1]$$

Unbounded multiplicities

An example of interpretation



In *Rel*, one denotation:

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

In *ScottL*, a set containing the principal type

$$(\{q_0, q_1\}, \{q_1\}, q_0)$$

but also

$$(\{q_0, q_1, q_2\}, \{q_1\}, q_0)$$

and

$$(\{q_0, q_1\}, \{q_0, q_1\}, q_0)$$

and ...

Intersection types and semantics of linear logic

$$\begin{tikzcd}
Rel_! \arrow[rr, leftrightarrow, "Bucciarelli-Ehrhard", "de\ \ Carvalho"'] \arrow[d, "Ehrhard"'] & & \text{Non-idempotent types} \arrow[d, "Ehrhard"] \\
ScottL_! \arrow[rr, leftrightarrow, "Ehrhard", "Terui"'] & & \text{Idempotent types}
\end{tikzcd}$$

Let t be a term normalizing to a tree $\langle t \rangle$ and $\mathcal{A}$ be an alternating automaton.

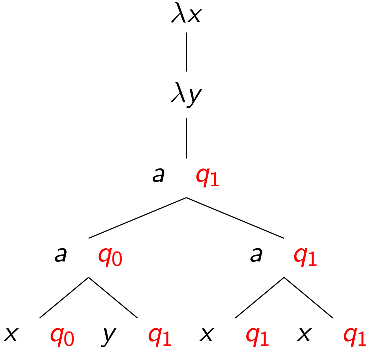
$$\mathcal{A} \text{ accepts } \langle t \rangle \text{ from } q \quad \Leftrightarrow \quad q \in [\![t]\!] \quad \Leftrightarrow \quad \emptyset \vdash t : q :: o$$

Extension with recursion and parity condition?

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 $\langle \mathcal{G} \rangle$.

We obtain decidability by considering idempotent types.

Our reformulation

- shows the modal nature of $\Box$ (in the sense of S4),
- internalizes the parity condition,
- paves the way for semantic constructions.

Colored semantics

We extend:

- *Rel* with countable multiplicities, coloring and an inductive-coinductive fixpoint
- *ScottL* with coloring and an inductive-coinductive fixpoint.

Methodology: think in the relational semantics, and adapt to the Scott semantics using Ehrhard's 2012 result:

the finitary model *ScottL* is the extensional collapse of *Rel*.

Finitary semantics

In ScottL, we define $\Box$, λ and $\mathbf{Y}$ using downward-closures.
$ScottL_{\not{\downarrow}}$ 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).

We could also obtain global model-checking and selection.

Similar model-theoretic results were obtained by Salvati and Walukiewicz the same year.

Probabilistic Termination

Checking a first property on probabilistic program

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

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

Fixpoint rule:

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

Typable $\implies$ SN. Proof using reducibility candidates.

Decidable type inference.

Sized types: example in the deterministic case

From Barthe et al. (op. cit.):

$$
\begin{array}{lll}
\mathrm{plus} \equiv (\texttt{letrec} & \mathit{plus}_{:\mathrm{Nat}^{\imath} \to \mathrm{Nat} \to \mathrm{Nat}} = & \\
 & \lambda x_{:\mathrm{Nat}^{\widehat{\imath}}}.\ \lambda y_{:\mathrm{Nat}}.\ \mathsf{case}\ x\ \mathsf{of}\ \{\mathsf{o} \Rightarrow y & \\
 & \qquad\qquad\qquad\qquad \mid \mathsf{s} \Rightarrow \lambda x'_{:\mathrm{Nat}^{\imath}}.\ \mathsf{s}\ \underbrace{(\mathit{plus}\ x'\ y)}_{:\mathrm{Nat}} & \\
 & \qquad\qquad\qquad\qquad \} & \\
) : & \mathrm{Nat}^{s} \to \mathrm{Nat} \to \mathrm{Nat} &
\end{array}
$$

The case rule ensures that the size of x' is lesser than the one of x.
Size decreases during recursive calls $\Rightarrow$ SN.

A probabilistic λ-calculus

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

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

- 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

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

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

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

A probabilistic λ-calculus: operational semantics

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

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

A probabilistic λ-calculus: operational semantics

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

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

A probabilistic λ-calculus: operational semantics

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

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 $\rightarrow_v^n$ 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:

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

- Unbiased random walk:

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

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

Capture this in a sized type system?

Another term

We also want to capture terms as:

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

of semantics

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

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:

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

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

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

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

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

$$\vdash$$

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

Designing the fixpoint rule

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

$$\vdash$$

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

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

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

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.

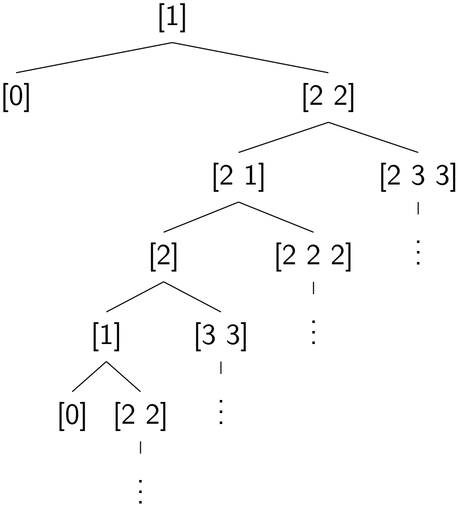
Higher-order symbols occur at most once. Consider:

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

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 $\Gamma \,|\, \Theta \vdash M \;:\; \mu$, then M is AST.

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

Conclusion of this part

Main features of the type system:

- Affine type system with distributions of types
- Sized walks induced by the letrec rule and solved by an external PTIME procedure
- Subject reduction + soundness for AST

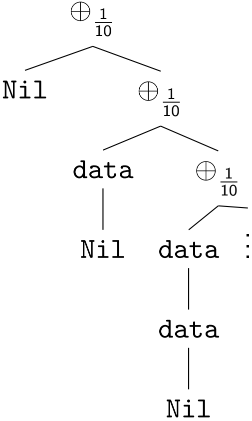
Next steps:

- type inference (decidable again??)
- extensions with refinement types, non-affine terms

Towards Higher-Order Probabilistic Verification

Probabilistic HOMC

```
IntList random_list() {
  IntList list = Nil;
  while(rand() > 0.1) {
    list := rand_int()::list;
  }
  return l;
}
```

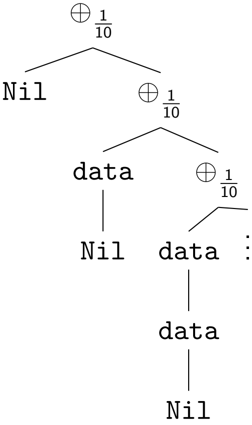


Probabilistic HOMC

Allows to represent probabilistic programs.

And to define higher-order regular Markov Decision Processes: those bisimilar to their encoding represented by a HORS.

(encoding of probabilities + payoffs in symbols)



Probabilistic automata

Idea: no longer verify φ but $Pr_{\geq p}\ \varphi$.

- Step one: quantitative ATA.
- Step two: deal with colors and parity condition.

Probabilistic automata (PATA):

- ATA on non-probabilistic symbols
- $+$ probabilistic behavior on choice symbol $\oplus_p$

Run-tree: labels $(q,\ p_n,\ p_f)$.

The root of a run-tree of probability p is labeled $(q_0, 1, p)$, where p is the probability with which we want the tree to satisfy the formula.

Probabilistic alternating tree automata

Probabilistic behavior:

$$\oplus_p \quad (q,\ p_n,\ p_f)$$

$$b \qquad\qquad c$$

is labeled as

$$\oplus_p \quad (q,\ p_n,\ p_f)$$

$$b \quad (q,\ p \times p_n,\ p'_f) \qquad\qquad c \quad (q,\ (1-p) \times p_n,\ p_f - p'_f)$$

for some $p'_f \in [0, p_f]$ such that $p'_f \leq p \times p_n$ and $p_f - p'_f \leq (1-p) \times p_n$.

Example of PATA run

$\varphi \quad = \quad$ "all the branches of the tree contain data"

is modeled by the PATA:

- $\delta_1(q_0, \mathtt{data}) \; = \; (1, q_1)$,
- $\delta_1(q_1, \mathtt{data}) \; = \; (1, q_1)$,
- $\delta_1(q_0, \mathtt{Nil}) \; = \; \perp$,
- $\delta_1(q_1, \mathtt{Nil}) \; = \; \top$.

Example of PATA run

$\oplus_{\frac{1}{10}} \quad (q_0,\ 1,\ \frac{9}{10})$

`Nil` $\quad (q_0,\ \frac{1}{10},\ 0)$

$\oplus_{\frac{1}{10}} \quad (q_0,\ \frac{9}{10},\ \frac{9}{10})$

`data` $\quad (q_0,\ \frac{9}{100},\ \frac{9}{100})$

`Nil` $\quad (q_1,\ \frac{9}{100},\ \frac{9}{100})$

$\oplus_{\frac{1}{10}} \quad (q_0,\ \frac{81}{100},\ \frac{81}{100})$

`data` $\quad (q_0,\ \frac{81}{1000},\ \frac{81}{1000})$

`data` $\quad (q_1,\ \frac{81}{1000},\ \frac{81}{1000})$

`Nil` $\quad (q_1,\ \frac{81}{1000},\ \frac{81}{1000})$

$\vdots$

Another example

φ = all the branches of the tree contain an even amount of `data`.

Associated automaton:

- $\delta_2(q_0, \mathtt{data}) = (1, q_1)$,
- $\delta_2(q_1, \mathtt{data}) = (1, q_0)$,
- $\delta_2(q_0, \mathtt{Nil}) = \top$,
- $\delta_2(q_1, \mathtt{Nil}) = \perp$.

Another example

$\oplus_{\frac{1}{10}} \quad (q_0,\ 1,\ \frac{10}{19})$

`Nil` $\quad (q_0,\ \frac{1}{10},\ \frac{1}{10})$

$\oplus_{\frac{1}{10}} \quad (q_0,\ \frac{9}{10},\ \frac{81}{190})$

`data` $\quad (q_0,\ \frac{9}{100},\ 0)$

$\oplus_{\frac{1}{10}} \quad (q_0,\ \frac{81}{100},\ \frac{81}{190})$

`data` $\quad (q_0,\ \frac{81}{1000},\ \frac{81}{1000})$

`data` $\quad (q_1,\ \frac{81}{1000},\ \frac{81}{1000})$

`Nil` $\quad (q_0,\ \frac{81}{1000},\ \frac{81}{1000})$

$\oplus_{\frac{1}{10}} \quad (q_0,\ \frac{729}{1000},\ \frac{6561}{19000})$

$\vdots$

Intersection types for PATA

As for ATA, except for tree constructors:

$$\frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\} \quad \text{satisfies} \quad \delta_A(q, a)}{\emptyset \vdash a \,:\, \bigwedge_{j=1}^{k_1} (q_{1j}, p_n, p_f) \to \ldots \to \bigwedge_{j=1}^{k_n} (q_{nj}, p_n, p_f) \to (q, p_n, p_f)}$$

$$\frac{p'_f \in]0, p_f[\quad \text{and} \quad p'_f \leq p \times p_n \quad \text{and} \quad p_f - p'_f \leq (1-p) \times p_n}{\emptyset \vdash \oplus_p \,:\, (q, p \times p_n, p'_f) \to (q, (1-p) \times p_n, p_f - p'_f) \to (q, p_n, p_f)}$$

$$\frac{q \in Q \quad \text{and} \quad p \times p_n \geq p_f}{\emptyset \vdash \oplus_p \,:\, (q, p \times p_n, p_f) \to \emptyset \to (q, p_n, p_f)}$$

$$\frac{q \in Q \quad \text{and} \quad (1-p) \times p_n \geq p_f}{\emptyset \vdash \oplus_p \,:\, \emptyset \to (q, (1-p) \times p_n,\, p_f) \to (q, p_n, p_f)}$$

Intersection types for PATA

Theorem

$$\emptyset \vdash S : (q_0, 1, p)$$

iff

the PATA $\mathcal{A}$ has a run-tree of probability p over the tree $\langle \mathcal{G} \rangle$ generated by $\mathcal{G}$.

Under connection Rel/non-idempotent types, we obtain a similar denotational theorem.

Note that $[\![o]\!] \;=\; Q \times [0,1] \times [0,1]$.

PATA and quantitative μ-calculus

The probabilistic μ-calculi zoo

- qmμ = quantitative interpretation of μ-calculus [HK97,MM97]
  - $\cup$ = max, $\cap$ = min, no PCTL, game characterization on finite models
- GPL = extension with finite nesting of $[\cdot]_{\succ p}$ quantifications [CPN99]
  - expresses PCTL* but neither $\exists\Box a$ nor $L\mu$ over Kripke structures
  - no game characterization, alternation-free fragment
- $pL\mu^{\odot}_{\oplus}$ is $L\mu$+ Lukasiewicz-operators + more [MS13]
  - probabilistic quantification = fixed point and multiplication
  - (tree) game characterization over all models, encodes PCTL
- μ^p and μPCTL [CKP15]
  - distinguishes between qualitative and quantitative formulas
  - model checking μ^p-calculus is as hard as solving parity games
  - poly-time model checking of μPCTL for bounded alternation depth
- PμTL = Lμ + $[\cdot]_{\succ p}$ for next-modalities [LSWZ15]
  - satisfiability by emptiness in prob. alt. parity automata (in 2EXPTIME)

PATA and quantitative μ-calculus

What we seem to capture: $[\![\, \varphi \,]\!]_\emptyset(\varepsilon) \geq p$ for safety formulas, with:

- $[\![\, \underline{a} \,]\!]_\rho(s) \;=\; 1$ iff label$(s)= a$, 0 else
- $[\![\, X \,]\!]_\rho(s) \;=\; \rho(X)(s)$
- $[\![\, \varphi \wedge \psi \,]\!]_\rho(s) \;=\; \min([\![\, \varphi \,]\!]_\rho(s), [\![\, \psi \,]\!]_\rho(s))$
- $[\![\, \varphi \vee \psi \,]\!]_\rho(s) \;=\; \max([\![\, \varphi \,]\!]_\rho(s), [\![\, \psi \,]\!]_\rho(s))$
- $[\![\, \Box\varphi \,]\!]_\rho(s) \;=\; \min\{[\![\, \varphi \,]\!]_\rho(s') \mid s' \text{ successor of } s\}$
- $[\![\, \diamond\varphi \,]\!]_\rho(s) \;=\; \max\{[\![\, \varphi \,]\!]_\rho(s') \mid s' \text{ successor of } s\}$
- $[\![\, \nu X.\varphi \,]\!]_\rho(s) \;=\; \mathrm{gfp}(f \mapsto [\![\, \varphi \,]\!]_{\rho[f/X]})(s)$

We did not consider the quantitative operator $\circledcirc\varphi$ but could add it, with

$$[\![\, \circledcirc\varphi \,]\!]_\rho(s) \;=\; \sum_{s' \text{ succ } s} Pr(s,s')[\![\, \varphi \,]\!]_\rho(s')$$

Why only safety?

Safety conditions $\rightarrow$ all infinite branches are accepted.

Problem with automata: can not detect *a priori* sets of loosing branches.

That's why there is an *a posteriori* parity condition.

To capture it: a colored run-tree of probability

$$p - p_{bad}$$

is

- a run-tree of probability p,
- where p_{bad} is the measure of the set of rejecting (= odd-colored) branches in the run-tree.

But how to reflect that size in the typing?

Current directions

- Try to connect to the more general obligation games (Chatterjee-Piterman) and the probabilistic μ-calculus of Castro-Kilmurray-Piterman
- Dual approach: look for safety/reachability properties using probabilistic extensions of Kobayashi's type system

Conclusions

- Multiple approaches for higher-order model-checking, from theory to practice. Here, using semantics of linear logic to make the theory clearer.
- A type system for checking termination of affine probabilistic programs.
- Some preliminary hints to check for more than just termination properties.

Thank you for your attention!

Conclusions

- Multiple approaches for higher-order model-checking, from theory to practice. Here, using semantics of linear logic to make the theory clearer.
- A type system for checking termination of affine probabilistic programs.
- Some preliminary hints to check for more than just termination properties.

Thank you for your attention!