Two Type-Theoretic Approaches to Probabilistic Termination

Ugo dal Lago  Charles Grellois

FOCUS Team – INRIA & University of Bologna
Université Aix-Marseille

Séminaire Logique de la Preuve, 21 septembre 2017
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 probabilistic **functional** languages?

We introduce a **monadic, affine sized type system** sound for AST (our result at ESOP 2017), and sketch a **dependent, affine** type system for AST (work in progress).
Sized Types and Termination

A sound termination check for the deterministic case
Sized Types: the Deterministic Case

Simply-typed $\lambda$-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 $\lambda$-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*. 
Sizes: \( s, r ::= i \mid \infty \mid \hat{s} \)

+ size comparison underlying subtyping. Notably \( \hat{\infty} \equiv \infty \).

Idea: \( k \) successors = at most \( k \) constructors.
- \( \hat{\text{Nat}}^i \) is 0,
- \( \hat{\text{Nat}}^i \) is 0 or \( S \ 0 \),
- \( \ldots \)
- \( \hat{\text{Nat}}^\infty \) is any natural number. Often denoted simply \( \text{Nat} \).

The same for lists,\( \ldots \)
Sized Types: the Deterministic Case

Sizes: \( s, r ::= i \mid \infty \mid \hat{s} \)

+ size comparison underlying subtyping. Notably \( \hat{\infty} \equiv \infty \).

Fixpoint rule:

\[
\Gamma, f : \text{Nat}^i \rightarrow \sigma \vdash M : \text{Nat}^{\hat{i}} \rightarrow \sigma[i/\hat{i}] \quad \text{i pos } \sigma \\
\hline 
\Gamma \vdash \text{letrec } f = M : \text{Nat}^s \rightarrow \sigma[i/s]
\]

“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: \( s, r ::= i \mid \infty \mid \tilde{s} \)

+ size comparison underlying subtyping. Notably \( \tilde{\infty} \equiv \infty \).

Fixpoint rule:

\[
\begin{align*}
\Gamma, f : \text{Nat}^i \to \sigma & \vdash M : \text{Nat}^{\tilde{i}} \to \sigma[i/\tilde{i}] \quad i \text{ pos } \sigma \\
\Gamma & \vdash \text{letrec } f = M : \text{Nat}^{s} \to \sigma[i/s]
\end{align*}
\]

Typable \( \implies \) SN. Proof using reducibility candidates.

Decidable type inference.
Sized types: example in the deterministic case

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

\[
\text{plus} \equiv \text{(letrec } \begin{align*}
\text{plus} &: \text{Nat} \rightarrow \text{Nat} 
&\rightarrow \text{Nat} = \\
\lambda x : \text{Nat}. &\lambda y : \text{Nat}. \text{ case } x \text{ of } \begin{cases}
o &\Rightarrow y \\
| s &\Rightarrow \lambda x' : \text{Nat}. s (\underbrace{\text{plus } x' y})_{\text{Nat}} 
\end{cases} 
\end{align*}
\) : \text{Nat} \rightarrow \text{Nat} 
\]

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 Lambda-Calculus and its Operational Semantics
A Probabilistic $\lambda$-calculus

$$M, N, \ldots ::= V \mid V \ V \mid \text{let } x = M \text{ in } N \mid M \oplus_p N \mid \text{case } V \text{ of } \{ S \rightarrow W \mid 0 \rightarrow Z \}$$

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

- Formulation equivalent to $\lambda$-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 $\lambda$-calculus: Operational Semantics

\[
\begin{align*}
\text{let } x &= V \text{ in } M & \rightarrow_v \{ (M[x/V])^1 \} \\
(\lambda x. M) V & \rightarrow_v \{ (M[x/V])^1 \} \\
(\text{letrec } f = V) \left( c \overrightarrow{W} \right) & \rightarrow_v \left\{ \left( V[f/(\text{letrec } f = V)] \left( c \overrightarrow{W} \right) \right)^1 \right\}
\end{align*}
\]
A Probabilistic $\lambda$-calculus: Operational Semantics

\[
\text{case } S \ V \text{ of } \{ S \rightarrow W \mid 0 \rightarrow Z \} \rightarrow_v \{ (W \ V)^1 \} \\
\]

\[
\text{case } 0 \text{ of } \{ S \rightarrow W \mid 0 \rightarrow Z \} \rightarrow_v \{ (Z)^1 \} \\
\]
A Probabilistic $\lambda$-calculus: Operational Semantics

\[
M \oplus_p N \rightarrow_v \left\{ M^p, N^{1-p} \right\}
\]

\[
M \rightarrow_v \left\{ L_i^p \mid i \in I \right\}
\]

\[
\text{let } x = M \text{ in } N \rightarrow_v \left\{ (\text{let } x = L_i \text{ in } N)^p_i \mid i \in I \right\}
\]
A Probabilistic $\lambda$-calculus: Operational Semantics

$$D \overset{VD}{=} \left\{ M_j^{p_j} \mid j \in J \right\} + D_V \quad \forall j \in J, \ M_j \rightarrow_v E_j$$

$$D \rightarrow_v \left( \sum_{j \in J} p_j \cdot E_j \right) + D_V$$

For $D$ a distribution of terms:

$$\llbracket D \rrbracket = \sup_{n \in \mathbb{N}} \left( \{ E_n \mid D \Rightarrow^n_v E_n \} \right)$$

where $\Rightarrow^n_v$ is $\rightarrow^n_v$ followed by projection on values.

We let $\llbracket M \rrbracket = \llbracket \{ M^1 \} \rrbracket$.

$M$ is AST iff $\sum \llbracket M \rrbracket = 1$. 
Random Walks as Probabilistic Terms

- **Biased** random walk:

\[ M_{bias} = \left( \text{letrec } f = \lambda x. \text{case } x \text{ of } \{ S \rightarrow \lambda y. f(y) \oplus \frac{2}{3} \left( f(S \ S \ y) \right) \ | \ 0 \rightarrow 0 \} \right)^n \]

- **Unbiased** random walk:

\[ M_{unb} = \left( \text{letrec } f = \lambda x. \text{case } x \text{ of } \{ S \rightarrow \lambda y. f(y) \oplus \frac{1}{2} \left( f(S \ S \ y) \right) \ | \ 0 \rightarrow 0 \} \right)^n \]

\[ \sum \llbracket M_{bias} \rrbracket = \sum \llbracket M_{unb} \rrbracket = 1 \]

Capture this in a sized type system?
We also want to capture terms as:

$$M_{nat} = \left( \text{letrec } f = \lambda x. x \oplus \frac{1}{2} S (f \, x) \right) 0$$

of semantics

$$\llbracket M_{nat} \rrbracket = \left\{ (0)^{\frac{1}{2}}, (S \, 0)^{\frac{1}{4}}, (S \, S \, 0)^{\frac{1}{8}}, \ldots \right\}$$

summing to 1.

(This is the geometric distribution.)
Distribution Types

A Probabilistic Counterpart to Sized Types
Beyond SN Terms, Towards Distribution Types

First idea: extend the sized type system with:

\[
\text{Choice} \quad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M \oplus_p N : \sigma}
\]

and “unify” types of \(M\) and \(N\) by subtyping.

Kind of product interpretation of \(\oplus\): we can’t capture more than SN...
First idea: extend the sized type system with:

\[
\text{Choice} \quad \frac{}{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma} \quad \Gamma \vdash M \oplus_p N : \sigma
\]

and “unify” types of \( M \) and \( N \) by subtyping.

We get at best

\[
f : \hat{\text{Nat}}^i \to \text{Nat}^\infty \vdash \lambda y. f(y) \oplus \frac{1}{2} (f(SSy))) : \hat{\text{Nat}}^i \to \text{Nat}^\infty
\]

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:

\[
\begin{align*}
&\text{Choice} & & \Gamma | \Theta \vdash M : \mu & & \Gamma | \Psi \vdash N : \nu & & \{\mu\} = \{\nu\} \\
& & & \Gamma | \Theta \oplus_p \Psi \vdash M \oplus_p N : \mu \oplus_p \nu
\end{align*}
\]

Now

\[
f : \left\{ \left( \text{Nat}^i \rightarrow \text{Nat}^\infty \right)^{\frac{1}{2}}, \left( \text{Nat}^i \rightarrow \text{Nat}^\infty \right)^{\frac{1}{2}} \right\}
\]

\[
\vdash \lambda y. f(y) \oplus^{\frac{1}{2}} (f(SSy))) : \text{Nat}^i \rightarrow \text{Nat}^\infty
\]
Designing the Fixpoint Rule

\[ f : \left\{ \left( \text{Nat}^i \rightarrow \text{Nat}^\infty \right)^{\frac{1}{2}}, \left( \text{Nat}^\hat{i} \rightarrow \text{Nat}^\infty \right)^{\frac{1}{2}} \right\} \]

\[ \vdash \lambda y. f(y) \oplus \frac{1}{2} (f(S S y))) : \text{Nat}^\hat{i} \rightarrow \text{Nat}^\infty \]

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

\[
\{ \Gamma \} = \text{Nat} \\
i \notin \Gamma \text{ and } i \text{ positive in } \nu \\
\{ (\text{Nat}^{s_j} \rightarrow \nu[i/s_j])^{p_j} \mid j \in J \} \text{ induces an AST sized walk}
\]

\[
\begin{align*}
\text{LetRec} & \quad \frac{\Gamma \mid f : \{ (\text{Nat}^{s_j} \rightarrow \nu[i/s_j])^{p_j} \mid j \in J \} \vdash V : \text{Nat}^{\hat{i}} \rightarrow \nu[i/\hat{i}]}{\Gamma \mid \emptyset \vdash \text{letrec } f = V : \text{Nat}^{\tau} \rightarrow \nu[i/\tau]}
\end{align*}
\]

Sized walk: AST is checked by an external PTIME procedure.
A crucial feature: our type system is affine.

Higher-order symbols occur at most once. Consider:

\[ M_{naff} = \text{letrec } f = \lambda x. \text{case } x \text{ of } \{ S \rightarrow \lambda y. f(y) \oplus \frac{2}{3} (f(S S y); f(S S y)) \mid 0 \rightarrow 0 \} \]

The induced sized walk is AST, but \( M_{naff} \) is not.
Generalized Random Walks and the Necessity of Affinity

Tree of recursive calls, starting from 1:

```
[0]   [1]
     /   /
   [2 2] [2 2]
      /   /
  [2 1] [2 3 3]
     /   /
 [2]   [2 2 2]
    /   /
[1]   [3 3]
     /   /
[0]   [2 2]
```

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 Property I: Subject Reduction

Main idea: reduction of

$$\emptyset \mid \emptyset \vdash 0 \oplus 0 : \left\{ \left( \text{Nat}^\hat{s} \right)^{\frac{1}{2}}, \left( \text{Nat}^\hat{t} \right)^{\frac{1}{2}} \right\}$$

is to

$$\left\{ \left( 0 : \text{Nat}^\hat{s} \right)^{\frac{1}{2}}, \left( 0 : \text{Nat}^\hat{t} \right)^{\frac{1}{2}} \right\}$$

1. Same expectation type: \( \frac{1}{2} \cdot \text{Nat}^\hat{s} + \frac{1}{2} \cdot \text{Nat}^\hat{t} \)
2. Splitting of \([0 \oplus 0]\) in a typed representation \(\rightarrow\) notion of pseudo-representation
Theorem

Let $M \in \Lambda \oplus$ be such that $\emptyset \mid \emptyset \vdash M : \mu$. Then there exists a closed typed distribution $\left\{ (W_j : \sigma_j)^{p'_j} \mid j \in J \right\}$ such that

- $\mathbb{E} \left( (W_j : \sigma_j)^{p'_j} \right) \preceq \mu$,
- and that $\left[ (W_j)^{p'_j} \mid j \in J \right]$ is a pseudo-representation of $\llbracket M \rrbracket$.

By the soundness theorem of next slide, this inequality is in fact an equality.
Key Property II: Typing Soundness

**Theorem (Typing soundness)**

\[ \text{If } \Gamma \mid \Theta \vdash M : \mu, \text{ then } M \text{ is AST.} \]

Proof by reducibility, using set of candidates parametrized by probabilities.
Reducibility, the Probabilistic Case

Usual reducibility proof:

\[ M \text{ closed of type } \sigma \implies M \in Red_{\sigma} \implies M \text{ is SN} \]

In our setting:
Reducibility, the Probabilistic Case

Usual reducibility proof:

\[ M \text{ closed of type } \sigma \implies M \in \text{Red}_\sigma \implies M \text{ is SN} \]

In our setting:

\[ M \in T\text{Red}^p_\sigma \implies \sum \| M \| \geq p \]
Reducibility, the Probabilistic Case

Usual reducibility proof:

\[ M \text{ closed of type } \sigma \Rightarrow M \in \text{Red}_\sigma \Rightarrow M \text{ is SN} \]

In our setting:

\[ M \text{ closed of type } \sigma \Rightarrow \forall p < 1, M \in \text{TRed}_\sigma^p \Rightarrow \forall p < 1, \sum \llbracket M \rrbracket \geq p \]

\( p \) increases with the number of fixpoint unfoldings we do, and we prove that \( M \) is in \( \text{TRed}_\sigma^p \) iff its \( n \)-unfolding is.
Reducibility, the Probabilistic Case

Usual reducibility proof:

\[ M \text{ closed of type } \sigma \implies M \in \text{Red}_\sigma \implies M \text{ is SN} \]

In our setting:

\[ M \text{ closed of type } \sigma \implies M \in T\text{Red}^1_\sigma \implies \sum \llbracket M \rrbracket = 1 \text{ i.e. } M \text{ AST} \]

by a continuity lemma.
Reducibility, the Probabilistic Case – Open Terms

Usual case: \( \boxed{\overrightarrow{x} : \sigma \vdash M : \tau} \Rightarrow \forall \overrightarrow{V} \in \overrightarrow{V\text{Red}}_\sigma, \ M[\overrightarrow{x}/\overrightarrow{V}] \in \overrightarrow{Red}_\tau \)
Reducibility, the Probabilistic Case – Open Terms

Usual case:  \[ \overrightarrow{x} : \overrightarrow{\sigma} \vdash M : \tau \implies \forall \overrightarrow{V} \in \overrightarrow{\text{VRed}}_\sigma, \ M[\overrightarrow{x}/\overrightarrow{V}] \in \text{Red}_\tau \]

In our setting: if \( \Gamma \mid y : \{\tau_j^{p_j}\}_{j \in J} \vdash M : \mu \) then

- \( \forall (q_i)_i \in [0, 1]^n, \ \forall \overrightarrow{V} \in \prod_{i=1}^n \text{VRed}_{\sigma_i}^{q_i}, \)
- \( \forall \left( q_j' \right)_j \in [0, 1]^J, \ \forall W \in \bigcap_{j \in J} \text{VRed}_{\tau_j}^{q_j'}, \)
- we have \( M[\overrightarrow{x}, y/\overrightarrow{V}, W] \in \text{TRed}_\mu^\alpha \)

where \( \alpha = (\prod_{i=1}^n q_i) \left( \left( \sum_{j \in J} p_j q_j' \right) + 1 - \left( \sum_{j \in J} p_j \right) \right) \).
Another Approach Using Dependent Types
Another Approach Using Dependent Types

Alternative approach to sized types: dependent types.

See Xi (2002), \textit{Dependent Types for Program Termination Verification}.

Examples of dependent types à la Xi:

\begin{itemize}
\item $\varphi \mid \Gamma \vdash 2 : \text{int} \ (2)$
\item $\varphi \mid \Gamma \vdash \langle 2 \mid 2 \rangle : \Sigma a : \text{int. int} \ (a)$
\end{itemize}

Terms of base type: annotated with size information which can be packed in the term (annotation by a size expression). Produces a sum type (existential).

$\varphi$: context of constraints on free size variables, like $a \in \{ a \in \text{int} \mid a > 2 \}$. 

Another Approach Using Dependent Types

Alternative approach to sized types: dependent types.

See Xi (2002), *Dependent Types for Program Termination Verification*.

Examples of dependent types à la Xi:

- \( \varphi \mid \Gamma \vdash + : \Pi \{a : int, b : int\}. \text{int}(a) \times \text{int}(b) \rightarrow \text{int}(a + b) \)
- \( \varphi \mid \Gamma \vdash \times : \Pi \{a : int, b : int\}. \text{int}(a) \times \text{int}(b) \rightarrow \text{int}(a \times b) \)

Functions typically have universally quantified arguments (product type). Note that we could derive terms from + and \( \times \) which use sum types for return types.
Another Approach Using Dependent Types

Sum types allow to get a uniform Choice rule:

\[ \varphi \mid \Gamma \mid \Theta \vdash M : \sigma \quad \varphi \mid \Gamma \mid \Theta \vdash N : \sigma \]

\[ \varphi \mid \Gamma \mid \Theta \vdash M \oplus_p N : \sigma \]

No longer need for distribution types!
Various sizes are annotated in the term.
Another Approach Using Dependent Types

$$\{ \Gamma \} \subseteq \{ \text{bool, int} \}$$

$$\varphi, \vec{\alpha} : \vec{\gamma} \mid \Gamma \mid f : \prod \vec{\alpha} : \vec{\gamma} . \sigma \models_P V : \sigma$$

letrec $$(\mathcal{P}, \rho)$$ is AST for every $$\rho \models \varphi$$

$$\varphi \mid \Gamma \mid \Theta \vdash \text{letrec } f[\vec{\alpha} : \vec{\gamma}] : \sigma = V : \prod \vec{\alpha} : \vec{\gamma} . \sigma$$

- Rely on PTS analysis (more general than random walks)
- letrec enters a new mode: typing relation $$\models_P$$ indexed by a PTS

$$\text{PTS} = \text{probabilistic transition system}$$
Examples of PTS

```c
int x := 0;
while (flip (0.5))
    x ++;
// end
```

Examples from Chakarov and Sankaranarayanan (2013), *Probabilistic Program Analysis with Martingales*
Examples of PTS

Examples from Chakarov and Sankaranarayanan (2013), *Probabilistic Program Analysis with Martingales*

Our point: replaced sized walks by these processes modeling the flow of recursive calls. The process is built on-the-fly by the type system.
Building the PTS

\[ \varphi \vdash \overrightarrow{f} : \overrightarrow{\gamma} \quad \mathcal{P} = \text{leaf} \left( f \left[ \overrightarrow{a} \mapsto \overrightarrow{I} \right] \right) \]

\[ \varphi \mid \Gamma \mid f : \prod \overrightarrow{a} : \overrightarrow{\gamma} \cdot \sigma \parallel_{\mathcal{P}} f \left[ \overrightarrow{I} \right] : \sigma[\overrightarrow{a} / \overrightarrow{I}] \]

**leaf** \( f \left[ \overrightarrow{a} \mapsto \overrightarrow{I} \right] \) is a PTS with just one node, looping on itself and updating \( \overrightarrow{a} \) with \( \overrightarrow{I} \).
Building the PTS

\[
\varphi \mid \Gamma \mid \emptyset \vdash M : \text{bool}(I)
\]

\[
\varphi, \ I = 1 \mid \Delta \mid \Theta \ \Vdash_P N : \sigma
\]

\[
\varphi, \ I = 0 \mid \Delta \mid \Theta \ \Vdash_Q L : \sigma
\]

\[
\varphi \mid \Gamma, \Delta \mid \Theta \ \Vdash_{\text{if}(I,P,Q)} \ \text{if } M \text{ then } N \text{ else } L : \sigma
\]

if \((I, P, Q)\) is a PTS containing \(P\) and \(Q\) and with one new node branching to the root of \(P\) or of \(Q\) depending on \([[I]]_\rho\).
Building the PTS

\[
\begin{align*}
\phi \mid \Gamma \mid \Theta & \vdash_P M : \sigma \\
\phi \mid \Gamma \mid \Theta & \vdash_Q N : \sigma \\
\phi \mid \Gamma \mid \Theta & \vdash_{P \oplus p Q} M \oplus_q N : \sigma
\end{align*}
\]

\(P \oplus_p Q\) is a PTS containing \(P\) and \(Q\) and with one new node branching to the root of \(P\) or of \(Q\) depending on a biased coin flip of probability \(q\).
Building the PTS

\[
\{ \Gamma \} \subseteq \{ \text{bool, int} \}
\]

\[
\varphi, \overrightarrow{a} : \overrightarrow{\gamma} | \Gamma | f : \prod \overrightarrow{a} : \overrightarrow{\gamma}.\sigma \vdash_P V : \sigma
\]

letrec \((\mathcal{P}, \rho)\) is AST for every \(\rho \models \varphi\)

\[
\varphi | \Gamma | \Theta \vdash \text{letrec } f[\overrightarrow{a} : \overrightarrow{\gamma}] : \sigma = V : \prod \overrightarrow{a} : \overrightarrow{\gamma}.\sigma
\]

letrec \((\mathcal{P}, \rho)\) is a PTS obtained from \(\mathcal{P}\) by making the loops on the leaves pointing to the root of \(\mathcal{P}\).
Conjecture

We have strong hints that:

\[ M \text{ has type } \sigma \implies M \text{ is AST}. \]

(we have a proof sketch based on the previous realizability argument).

Note that the system is again affine.
Conclusion

First 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**

Second type system:

- **Finer analysis**: more expressive sizes, modelization by PTS
- No need for distribution types thanks to sum types
- Still **affine**
- Soundness is work in progress

Thank you for your attention!
Conclusion

First 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

Second type system:

- **Finer analysis**: more expressive sizes, modelization by PTS
- No need for distribution types thanks to sum types
- Still **affine**
- Soundness is work in progress

Thank you for your attention!