Two Type-Theoretic Approaches to Probabilistic Termination

Ugo dal Lago  Charles Grellois

FOCUS Team – INRIA & University of Bologna

DICE-FOPARA 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: 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*. 
Sized Types: the Deterministic Case

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 \( \text{S} \ 0 \),
- \( \ldots \)
- \( \hat{\text{Nat}}^{\infty} \) 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:

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

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

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

Fixpoint rule:

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

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

Decidable type inference.
A Probabilistic $\lambda$-calculus

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

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

- 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

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

$$\text{(}\lambda x . M) V \rightarrow_v \left\{ (M[x/V])^1 \right\}$$

$$\text{(letrec } f = V) \left( \text{c } W \right) \rightarrow_v \left\{ \left( V[f/(\text{letrec } f = V)] \left( \text{c } W \right) \right)^1 \right\}$$
A Probabilistic $\lambda$-calculus: Operational Semantics

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

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

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

\[
M \rightarrow_v \{ L_i^{p_i} \mid i \in I \}
\]

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

\[
\mathcal{D} \overset{V\mathcal{D}}{=} \left\{ M_j^{p_j} \mid j \in J \right\} + \mathcal{D}_V \quad \forall j \in J, \quad M_j \rightarrow_v E_j
\]

\[
\mathcal{D} \rightarrow_v \left( \sum_{j \in J} p_j \cdot E_j \right) + \mathcal{D}_V
\]

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

\[
\begin{array}{c}
\left[ \mathcal{D} \right] = \sup_{n \in \mathbb{N}} \left( \left\{ E_n \mid \mathcal{D} \Rightarrow^n_v E_n \right\} \right)
\end{array}
\]

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

We let $\left[ M \right] = \left[ \left\{ M^1 \right\} \right]$.

$M$ is AST iff $\sum \left[ M \right] = 1$. 
Random Walks as Probabilistic Terms

- **Biased** random walk:

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

- **Unbiased** random walk:

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

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

Capture this in a sized type system?
Another Term

We also want to capture terms as:

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

of semantics

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

summing to 1.

(This is the geometric distribution.)
Beyond SN Terms, Towards Distribution Types

**First idea:** extend the sized type system with:

\[
\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma
\]

Choice

\[
\frac{\Gamma \vdash M \oplus p N : \sigma}{\Gamma \vdash \Gamma \vdash M \oplus p N : \sigma}
\]

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:

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

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

We get at best

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

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:

\[
\frac{\Gamma \mid \Theta \vdash M : \mu \quad \Gamma \mid \Psi \vdash N : \nu \quad \{\mu\} = \{\nu\}}{
\Gamma \mid \Theta \oplus_p \Psi \vdash M \oplus_p N : \mu \oplus_p \nu}
\]

Now

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

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

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

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

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 (\(\Rightarrow\) reaches 0 with proba 1) \(\Rightarrow\) termination.
Designing the Fixpoint Rule

\[ \{ \Gamma \} = \text{Nat} \]

\[ i \not\in \Gamma \text{ and } i \text{ positive in } \nu \]

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

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

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:

$$M_{naff} = \text{letrec } f = \lambda x.\text{case } x \text{ of } \begin{cases} S \to \lambda y.f(y) \oplus_2 (f(S S y); f(S S y)) & 0 \to 0 \end{cases} \text{ of }$$

The induced sized walk is AST, but $M_{naff}$ is not.
Key Property I: Subject Reduction

Main idea: reduction of

\[ \emptyset | \emptyset \vdash 0 \oplus 0 : \left\{ \left( \text{Nat}^\ddagger \right)^{\frac{1}{2}}, \left( \text{Nat}^\ddag \right)^{\frac{1}{2}} \right\} \]

is to

\[ \left\{ \left( 0 : \text{Nat}^\ddagger \right)^{\frac{1}{2}}, \left( 0 : \text{Nat}^\ddag \right)^{\frac{1}{2}} \right\} \]

1. Same expectation type: \( \frac{1}{2} \cdot \text{Nat}^\ddagger + \frac{1}{2} \cdot \text{Nat}^\ddag \)
2. Splitting of \( [0 \oplus 0] \) in a typed representation → notion of pseudo-representation
Key Property I: Subject Reduction

**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

- $\mathbb{E} \left( (W_j : \sigma_j)^{p_j'} \right) \preccurlyeq \mu$,

- 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)

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

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

In our setting:

\[ M \in T\text{Red}_\sigma^P \Rightarrow \sum [M] \geq p \]
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:

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

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

Usual reducibility proof:

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

In our setting:

\[ M \text{ closed of type } \sigma \quad \Rightarrow \quad M \in \text{TRed}^1_\sigma \quad \Rightarrow \quad \sum \mathbb{[} M \mathbb{]} = 1 \text{ i.e. } M \text{ AST} \]

by a continuity lemma.
Usual case: $\overline{x} : \overline{\sigma} \vdash M : \tau \quad \Rightarrow \quad \forall \overrightarrow{V} \in V\text{Red}_\overline{\sigma}, \quad M[\overrightarrow{x}/\overrightarrow{V}] \in \text{Red}_\tau$
Reducibility, the Probabilistic Case – Open Terms

Usual case: \( \vec{x} : \sigma \vdash M : \tau \Rightarrow \forall \vec{V} \in \mathcal{VRed}_\sigma, \; M[\vec{x}/\vec{V}] \in \mathit{Red}_\tau \)

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

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

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

Alternative approach to sized types: dependent types.

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

Examples of dependent types à la Xi:

- \( \varphi \mid \Gamma \vdash 2 : \text{int} \ (2) \)

- \( \varphi \mid \Gamma \vdash \langle 2 \mid 2 \rangle : \Sigma a : \text{int}. \text{int} \ (a) \)

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:

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

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

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

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

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

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

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

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

PTS = probabilistic transition system
Examples of PTS

Example 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

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

\[ \varphi \mid \Gamma \mid f : \prod \vec{a} : \vec{\gamma} . \sigma \models \mathcal{P} \left( f \left[ \vec{l} \right] \right) : \sigma[\vec{a}/\vec{l}] \]

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

\[
\varphi | \Gamma | \emptyset \vdash M : \text{bool}(I)
\]
\[
\varphi, I = 1 | \Delta | \Theta \vdash_{P} N : \sigma
\]
\[
\varphi, I = 0 | \Delta | \Theta \vdash_{Q} L : \sigma
\]
\[
\varphi | \Gamma, \Delta | \Theta \vdash_{\text{if}(I, P, Q)} \text{if } M \text{ then } N \text{ else } L : \sigma
\]

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

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

\[
\varphi \mid \Gamma \mid \Theta \vdash_{P \oplus_p Q} M \oplus_q N : \sigma
\]

\( P \oplus_p Q \) 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

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

\[
\varphi, \overrightarrow{a} : \overrightarrow{\gamma} \mid \Gamma \vdash f : \prod \overrightarrow{a} : \overrightarrow{\gamma}.\sigma \quad \vdash_{P} \quad V : \sigma
\]

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

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

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