Randomized Algorithms in Type Theory

Christine Paulin, LRI - Université Paris Sud
with Philippe Audebaud, ENS Lyon & Richard Lassaigne, Université Paris 7

September 26, 2001

Abstract

We study how to specify and prove randomized algorithms in type theory. Our goal is to build an environment for reasoning on randomized algorithms on top of the Coq proof assistant. This work presents a shallow embedding of a functional language with randomized constructions into type theory and a set of rules to analyse the probabilistic behavior of programs expressed in this language.

It is known that there is a natural representation of randomized programs constructions as operations on distributions on the state. Instead to interpret an imperative program as a function from state to state, it becomes a function from distribution on state to distribution on state. Pre and post conditions are generalised to real-valued functions on state, with values in the segment $[0, 1]$. The fact that a program satisfies a certain pre and post-condition is generalised to the fact that for any input distribution, the measure of the precondition is less than the measure of the post-condition for the output distribution of the program. The purpose of this talk is to

show how to apply this technics for doing an interpretation of functional randomized algorithms inside a pure functional type theory. We show that the interpretation of programs corresponds to a monadic transformation $(\tau \mapsto (\tau \to [0,1]) \to [0,1])$. We discuss a possible representation of $[0,1]$ in type theory which enjoys the expected properties. We finally propose a system for reasoning on randomized programs.

Randomized Algorithms in Type Theory

	Christine Paulin	LRI - Université Paris Sud
with	Philippe Audebaud	ENS Lyon
	Richard Lassaigne	Université Paris 7

Work in progress

Outline

Dependent types are good for program specification and certification

No fancy dependent type !!

- Motivations
- A monadic transformation for analysing randomized programs
- Examples
- Expected properties
- Conclusion

Motivations

How to specify and prove randomized algorithms in Coq?

- Programming & mathematical problems
- Certified programs $\equiv$ dependently typed functional terms

$$\frac{f : A \to B \qquad \forall x : A. P(x) \Rightarrow Q(x, f(x))}{F : \prod x.P(x) \to \{y : B | Q(x, y)\}}$$

$$\frac{f : A \to B \qquad \forall x : A. P(x) \Rightarrow \mathsf{Pr}(Q(x, f(x)) \geq 1 - \epsilon}{\textbf{?}}$$

Explicit solution

The result depends on an additional randomized input $x \in \Omega$.

$$\forall x : A.P(x) \Rightarrow \exists \Omega, \mathcal{E} \subseteq \Omega.$$

$$\mathsf{Pr}(\mathcal{E}) \leq \epsilon$$

$$\wedge$$

$$\forall \omega \in \Omega.\{y : B. \left| \begin{array}{l} Q(x,y) \vee \omega \in \mathcal{E} \\ \omega \notin \mathcal{E} \Rightarrow Q(x,y) \\ \neg Q(x,y) \Rightarrow \omega \in \mathcal{E} \end{array} \right. \right\}$$

- $\omega \in \mathcal{E}$ is a weak form of absurdity.
- Good for studying algorithmic aspects.
- Which Ω ?

Implicit solution

D. Kozen, G. Plotkin, C. Jones, C. Morgan ...

- Imperative framework with $x := \texttt{random}$ or $P +_p Q$.
- A randomized program P is non-deterministic, its semantics is a set of states.
- Analyse the distribution of this set.
- Interpret randomized algorithm P as transformers of "measures" on the set of state S.

Axiomatic semantics

f and g are functions from S to $[0, 1]$

$$f[P]g \Leftrightarrow \forall \mu. \int f \; d\mu \leq \int g \; dP(\mu)$$

Property ϕ on states : characteristic function $\mathbb{I}_\phi$.

$$\phi(s) \text{ true} \quad \leftrightarrow \int \mathbb{I}_\phi \; d\mu = 1$$

State s : Dirac measure δ_s.

Interpretation of a functional randomized language

(First-order) functional ML $+$ `random`

- A program $p : \tau$ represents a set of values of type τ
- A program $p : \tau$ corresponds to a distribution μ_p on τ
- When $Q : \tau \to \texttt{Prop}$, $\mathsf{Pr}(p \in Q) = \mu_p(Q)$

$$\frac{\mathsf{Pr}(Q\ x\ (f\ x)) \geq 1 - \epsilon}{\mu_{(f\ x)}\ \mathbb{I}_{(Q\ x)} \geq 1 - \epsilon}$$

Monadic transformation

Computations	Functional values
$p : \tau$	$\widehat{p} : \widehat{\tau}$
	$?\mathsf{unit} : \tau \to \widehat{\tau}$ $?\mathsf{star} : \widehat{\tau} \to (\tau \to \widehat{\sigma}) \to \widehat{\sigma}$ $?\widehat{\mathtt{random}} : \widehat{\tau}$
$\mathbf{let}\ x\ = a\ \mathbf{in}\ b$ $[x : \tau]t$ $(t\ u)$ $\mathbf{if}\ b\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$	$(\mathsf{star}\ \widehat{a}\ [x : \tau]\widehat{b})$ $[x : \tau]\widehat{t}$ $(\mathsf{star}\ \widehat{u}\ \widehat{t})$ $(\mathsf{star}\ \widehat{b}\ [x : \mathtt{bool}]\mathbf{if}\ x\ \mathbf{then}\ \widehat{e_1}\ \mathbf{else}\ \widehat{e_2})$

Which monad ?

Transform $p : \tau$ into a measure μ_p on τ of type $\widehat{\tau} \equiv (\tau \to [0,1]) \to [0,1]$
Kind of double negation, usual monadic transformation :

$$\begin{aligned}
\mathsf{unit} \quad &: \quad \tau \to \widehat{\tau} \\
&= \quad [x:\tau][f:\tau \to [0,1]](f\ x) \\
\mathsf{star} \quad &: \quad \widehat{\tau} \to (\tau \to \widehat{\sigma}) \to \widehat{\sigma} \\
&= \quad [\mu_a : \widehat{\tau}][\mu_b : \tau \to \widehat{\sigma}][f : \sigma \to [0,1]](\mu_a\ [x:\tau](\mu_b\ x\ f))
\end{aligned}$$

Satisfy the expected properties

Interpretation of randomized constructions

$$
\begin{array}{lcl}
\texttt{random}(n) & : & \widehat{\texttt{int}} \\
 & = & [f : \texttt{int} \to [0,1]]\Sigma_{i=1}^{n}(f\ i)/n \\
\texttt{flip}() & : & \widehat{\texttt{bool}} \\
 & = & [f : \texttt{bool} \to [0,1]]\frac{1}{2}(f\ \texttt{true}) + \frac{1}{2}(f\ \texttt{false}) \\
e_1 +_p e_2 & : & \widehat{\tau} \\
 & = & [f : \tau \to [0,1]]p \times (\widehat{e_1}\ f) + (1-p) \times (\widehat{e_2}\ f)
\end{array}
$$

Example : checking primality of p

$p - 1 = 2^k.u \quad 1 < a < p$

$(\texttt{test}\ p\ a) \stackrel{\text{def}}{=} a^u \equiv 1(p) \vee \exists j.0 \leq j < k.a^{2^j u} \equiv -1(k)$

Properties :

$$p \text{ prime} \Rightarrow \forall a.(\texttt{test}\ p\ a)$$

$$p \text{ not prime \& odd} \Rightarrow |\{a | (\texttt{test}\ p\ a)\}| \leq \frac{(p-1)}{4}$$

Program :

let rec random_test p n =
if n = 0 **then** true
else if test p (random (p-1)) **then** random_test p (n-1)
else false

Functional interpretation

Transform a functional program with `random` into a measure that can be executed to compute the probability for the output to satisfy a decidable property.

```
let rec random_test_fun p n =
if n = 0 then (unit true)
else star  (star (random (p-1)) [a](unit (test p a)))
           [b]if b then random_test_fun p (n-1) else (unit false)
```

Functional computation

let evaluate p n =
  let test_correct b =
   if b **then if** exact_prime p **then** 1. **else** 0. **else** 1.
in random_test_fun p n test_correct

```
# evaluate 23 1;;
- : float = 1
# evaluate 9 1;;
- : float = 0.75
# evaluate 9 2;;
- : float = 0.9375
# evaluate 9 3;;
- : float = 0.984375
```

Inefficient : test all the cases for the value of `random`!!

General Recursion

A program that only terminates with probability one :

let rec fix_test x =

if flip () **then** fix_test (x+1) **else** x

```
# fix_test 1;;
- : int = 1
# fix_test 1;;
- : int = 2
# fix_test 1;;
- : int = 2
# fix_test 1;;
- : int = 3
# fix_test 1;;
- : int = 1
```

Functional interpretation

let rec fix_test_fun x =

star flip() [b] **if** b **then** fix_test_fun (x+1) **else** (unit x)

```
# fix_test_fun 1 (fun n -> 1.);;
Stack overflow during evaluation (looping recursion?).
```

Needs to compute limits

General fixpoints

let rec f x = F f x

$$F : (\tau \to \widehat{\sigma}) \to \tau \to \widehat{\sigma}$$

Definitions :

$$f_0 \equiv [x][f]0 \qquad f_{n+1} \equiv (F\ f_n) \qquad \mathbf{fix}\ F \equiv \sqcup_{n\in\mathbb{N}}\, f_n$$

Properties :

$$f \leq g \Rightarrow (F\ f) \leq (F\ g) \qquad \Longrightarrow \quad f_n \leq f_{n+1}$$

$$(F\ \sqcup_{n\in\mathbb{N}}\, f_n) = \sqcup_{n\in\mathbb{N}}(F\ f_n) \quad \Longrightarrow \quad (F\ (\mathbf{fix}\ F)) = (\mathbf{fix}\ F)$$

Properties of measures

$$\mu(f_1 + f_2) = \mu(f_1) + \mu(f_2) \qquad \mu(\lambda f) = \lambda\mu(f)$$

$$f \leq g \Rightarrow \mu(f) \leq \mu(g)$$

$$\mu(\mathbb{O}) = 0 \qquad \mu(\mathbb{I}) \leq 1$$

$$\mu(\sqcup(f_n)_{n\in\mathbb{N}}) = \sqcup(\mu(f_n))_{n\in\mathbb{N}}$$

Specification of [0,1]

Primitives

$$0 \text{ (false)} \qquad 1 \text{ (true)}$$

$$i_1 \leq i_2 \ (i_1 \Rightarrow i_2) \qquad i_1 \sqcup i_2 \ (i_1 \vee i_2) \qquad i_1 \sqcap i_2 \ (i_1 \wedge i_2)$$

$$i_1 + i_2 \qquad i_1 \times i_2$$

$$\sqcup_{n \in \mathbb{N}} i_n$$

Properties

$$i_1 = 0 \vee i_1 = 1 \Rightarrow i_1 \sqcup i_2 = i_1 + i_2$$

$$i_1 = 0 \vee i_1 = 1 \Rightarrow i_1 \sqcap i_2 = i_1 \times i_2$$

Implementing $[0,1]$

- A subset of positive reals : $[0,1]\equiv\{x:\mathbb{R}^+|0\leq x\leq 1\}$
  - Effective computation of $\mu_p(f)$
  - How to encode $(\tau\to\texttt{Prop})\to(\tau\to[0,1])$?
  - How to encode $\sqcup i_n$?
  - How to encode `random` for a uniform distribution ?
    $(f:[0,1]\to[0,1])\leadsto\int f\ d\mu$
- Intensional vs extensional functions from τ to $[0,1]$

$$f:\tau\to[0,1]\qquad f:\tau\to[0,1]\to\texttt{Prop}$$

Solution :

$$[0,1]\equiv\{\mathbb{R}^+\to\texttt{Prop}|\ldots\}$$

Two solutions

$I : \mathbb{R}^+ \to \texttt{Prop}$

- Singletons sets : $x \in I \Rightarrow y \in I \Rightarrow x = y$
- Closed segments : $\begin{cases} 0 \in I \\ x \in I \Rightarrow y \leq x \Rightarrow y \in I \\ (\forall y < x . y \in I) \Rightarrow x \in I \end{cases}$

To relate $I \in [0, 1]$ and $d \in \mathbb{R}$ we have to do a proof.

$d \leq I$

- singleton $\forall x . x \in I \Rightarrow d \leq x$
- segment $d \in I$

Operations on segments (U)

$x : \mathbb{R}^+, u : U$	$x \in u$	$\equiv (u\ x)$
$u_1, u_2 : U$	$u_1 \leq u_2$	$\equiv \forall x. x \in u_1 \Rightarrow x \in u_2$
$P : \texttt{Prop}$	$\widetilde{P}$	$\equiv [x] x = 0 \vee P$
$d : \mathbb{R}^+, 0 \leq d \leq 1$	$\widetilde{d}$	$\equiv [x] x \leq d$
$b : \texttt{bool}$	$\widetilde{b}$	$\equiv [x] \textbf{if } b \textbf{ then } \texttt{True} \textbf{ else } x = 0$
$u : I \rightarrow U$	$\sqcap_{i \in I}(u\ i)$	$\equiv [x] \forall i. (x \in (u\ i))$
$u : I \rightarrow U$	$\sqcup_{i \in I}(u\ i)$	$\equiv [x] (\forall y < x. \exists i. (y \in (u\ i)))$

Doing proofs of programs

A well-typed randomized term : $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash p : \tau$

Specifications : $P : \tau_1 \to \cdots \to \tau_n \to [0,1], Q : \tau \to [0,1]$

A correct program : $(x_1 : \tau_1, \ldots, x_n : \tau_n)$ **pr** $P \vdash p$ **pr** Q

Interpretation :

$$\forall \mu_i : \widehat{\tau_i}.(\mu_1\ [x_1](\mu_2\ \ldots(\mu_n[x_n](P\ x_1\ \cdots\ x_n))\ldots)) \leq (\widehat{p}\ Q)$$

Functional specifications

$$\nu : \tau \to \widehat{\sigma}$$

Specification of the input/output behavior :

$$P : \tau \to [0,1] \quad Q : \sigma \to [0,1]$$

Definition of ν **pr** $(P \to Q)$:

$$\forall \mu : \widehat{\tau}.(\mu\ P) \leq (\mu\ [x](\nu\ x\ Q))$$

Exactly what is needed for application.

Some rules

$$\frac{\Gamma\ \mathbf{pr}\ P \vdash e\ \mathbf{pr}\ Q \quad P' \leq P \quad Q \leq Q'}{\Gamma\ \mathbf{pr}\ P' \vdash e\ \mathbf{pr}\ Q'}$$

$$\frac{\Gamma\ \mathbf{pr}\ P \vdash e\ \mathbf{pr}\ Q \quad \Gamma\ \mathbf{pr}\ P' \vdash e'\ \mathbf{pr}\ Q}{\Gamma\ \mathbf{pr}\ (\widehat{b}\ \widetilde{\mathtt{true}})P + (\widehat{b}\ \widetilde{\mathtt{false}})P' \vdash \mathbf{if}\ b\ \mathbf{then}\ e\ \mathbf{else}\ e'\ \mathbf{pr}\ Q}$$

$$\frac{f\ \mathbf{pr}\ P_i \to Q \vdash t\ \mathbf{pr}\ P_{i+1} \to Q \quad P_0 \equiv \widetilde{0}}{\mathbf{fix}\ [f]t\ \mathbf{pr}\ \sqcup_{i \in \mathbb{N}} P_i \to Q}$$

Example

let rec f x = **if** `flip` **then** f (x+1) **else** x

Proof that the program terminates with probability one :

Takes $P_{i+1} \equiv \frac{1}{2} P_i + \frac{1}{2} \quad Q \equiv \mathbb{I}$

Assume f **pr** $P_i \to \mathbb{I} \equiv \forall \mu.(\mu\ P_i) \leq (\mu[x](f\ x\ \mathbb{I}))$

Prove **if** `flip` **then** $\widehat{f(x}+1)$ **else** x **pr** $P_{i+1} \to \mathbb{I}$

Justification

$$
\begin{array}{rcl}
(\mu\ P_{i+1}) & = & (\mu'\ P_{i+1}) \qquad\qquad (\mu'\ f) \equiv (\mu'\ [x](f\ x+1)) \\
& = & \frac{1}{2}(\mu'\ P_i) + \frac{1}{2} \\
& \leq & \frac{1}{2}(\mu'\ [x](f\ x\ \mathbb{I})) + \frac{1}{2} \\
& = & \frac{1}{2}(\mu\ [x](f\ (x+1)\ \mathbb{I})) + \frac{1}{2} \\
& = & (\mu\ [x]\frac{1}{2}(f\ (x+1)\ \mathbb{I}) + \frac{1}{2})
\end{array}
$$

Example 2

Improve the precision :

$p_1, p_2 : \tau$

$Q : \tau \to \texttt{Prop}$

Γ **pr** $P_i \vdash p_i$ **pr** $\widetilde{Q}$

$$q \equiv \textbf{let } x \ = p_1 \textbf{ in let } y \ = p_2 \textbf{ in } (\texttt{choice}\ x\ y)$$

Hypothesis : $(Q\ x) \vee (Q\ y) \Rightarrow (Q\ (\texttt{choice}\ x\ y))$

Result : Γ **pr** $P_1 + P_2 - P_1 P_2 \vdash q$ **pr** $\widetilde{Q}$

Conclusion

- Mainly reformulation of known technics at a syntactic level
- More suitable for mechanical manipulation (?)

Future work

Theory

- Justify formally this theory
- Precise comparison with other approaches
- Analyse the complexity of programs

Coq

- Library for measure manipulation
- Non trivial examples
- Partial automation of the proofs