Program Logic for Higher-Order Probabilistic Programs in Isabelle/HOL

Michikazu Hirata, Yasuhiko Minamide, Tetsuya Sato

Tokyo Institute of Technology

FLOPS2022
May 10, 2022

Probabilistic Programming Languages

- Programmers write a probabilistic distribution as a program.
- The languages support sampling and conditioning.

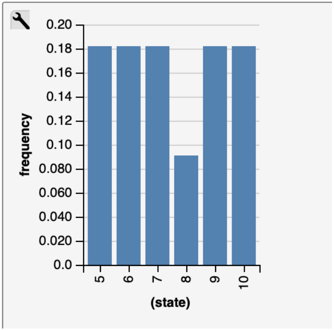
Ex.* The distribution when we roll two dice and at least one die is 4.

```
var roll = function () {
  var die1 = randomInteger(6) + 1;
  var die2 = randomInteger(6) + 1;

  // Only keep executions where at least one die is a 4.
  condition(die1 === 4 || die2 === 4);

  return die1 + die2;
}


var dist = Enumerate(roll);
viz.auto(dist);
```



*Adrian, S. Probabilistic programming.
https://www.cs.cornell.edu/courses/cs4110/2016fa/lectures/lecture33.html.

Probabilistic Programming Languages

Monte Carlo method

For a sequence $x_1, \ldots, x_n$ sampled from a distribution $\mathtt{d}$,

$$\mathbb{E}_{x \sim \mathtt{d}}[\mathtt{h}(x)] \approx \text{the average of } h(x_1), \ldots, h(x_n).$$

($\mathtt{d}$ is a probability distribution on $\mathtt{X}$ and $\mathtt{h} : \mathtt{X} \Rightarrow \mathtt{real}$ is a function.)

$$montecarlo : \mathtt{nat} \Rightarrow P[\mathtt{real}]$$

$$\begin{aligned} montecarlo\,\mathtt{n} \equiv \mathtt{if}\ (\mathtt{n} = 0)\ & \mathtt{then\ return}\ 0 \\ & \mathtt{else\ do}\ \{ \\ & \quad \mathtt{m} \leftarrow montecarlo\ (\mathtt{n} - 1); \\ & \quad \mathtt{x} \leftarrow \mathtt{d}; \\ & \quad \mathtt{return}\ ((1/\mathtt{n}) * (\mathtt{h}(\mathtt{x}) + \mathtt{m} * (\mathtt{n} - 1)))\ \} \end{aligned}$$

$P[\mathtt{real}]$ is the type of probability distributions on $\mathtt{real}$.

Expected property: The grater n is, the closer $montecarlo\ n$ is to $\mathbb{E}_{x \sim \mathtt{d}}[\mathtt{h}(x)]$.

Semantics of Probabilistic Programming Languages

Semantics based on measure theory

The probability monad (the Giry monad) G.

$$\Gamma \vdash e : P[X] \xrightarrow{\text{Interpretation}} \text{A measurable function } [\![e]\!] : [\![\Gamma]\!] \to G([\![X]\!]).$$

Problem: Function spaces (with desired properties) do not exist in general.

Semantics based on quasi-Borel spaces[Heunen+, LICS2017]

A suitable model for higher-order languages.

- Function spaces exist.
- Every probability distribution on *standard borel spaces* (e.g. $\mathbb{R}^n$, $\mathbb{N}$) is represented as a *probability distribusion* on a quasi-Borel space.
- The probability monad is commutative strong.

The Verification Framework PPV[Sato+, POPL2019]

PPV(Probabilistic Programming Verification framework) is a verification framework for higher-order probabilistic programming language.

- PPV consists of the language and three kind of logics (Assertion logic, Unary logic, Relational logic).
- The language supports sampling and conditioning.
- Its semantics is based on quasi-Borel spaces.

PPV was applied to verify

- Monte Carlo method,
- Importance sampling, and
- Gaussian mean learning.

Contributions

Goal:

Verification of **higher-order** probabilistic programs with proof assistant.

Contributions:

- Formalizing quasi-Borel spaces in Isabelle/HOL
- Formalizing a core part of PPV
  (The language, Assertion logic, Unary logic)
  - Our PPV does not support conditioning.
  - We introduced integrability in the logic because it is necessary.
- Verification of the Monte Carlo method on mechanized PPV (including the integrability)

Contributions

Goal:

Verification of **higher-order** probabilistic programs with proof assistant.

We choose **Isabelle/HOL**.

- Rich probability theory library (including the Giry monad)

Quasi-Borel spaces	8,950
PPV	3,100
Integrability of Monte Carlo approximation	150
Verification of the Monte Carlo method	300
Total	12,500

Outline

Verification of Monte Carlo Approximation

Monte Carlo method

For a sequence $x_1, \ldots, x_n$ sampled from a distribution $\mathtt{d}$,

$$\mathbb{E}_{x\sim\mathtt{d}}[\mathtt{h}(x)] \approx \text{the average of } h(x_1), \ldots, h(x_n).$$

($\mathtt{d}$ is a probability distribution on $\mathtt{X}$ and $\mathtt{h} : \mathtt{X} \Rightarrow \mathtt{real}$ is a function.)

$$\begin{aligned}
&\mathit{montecarlo} : \mathtt{nat} \Rightarrow P[\mathtt{real}] \\
&\mathit{montecarlo}\ \mathtt{n} \equiv \mathtt{if}\ (\mathtt{n} = 0)\ \mathtt{then}\ \mathtt{return}\ 0 \\
&\qquad \mathtt{else}\ \mathtt{do}\ \{ \\
&\qquad\quad \mathtt{m} \leftarrow \mathit{montecarlo}\ (\mathtt{n} - 1); \\
&\qquad\quad \mathtt{x} \leftarrow \mathtt{d}; \\
&\qquad\quad \mathtt{return}\ ((1/\mathtt{n}) * (\mathtt{h}(\mathtt{x}) + \mathtt{m} * (\mathtt{n} - 1)))\ \}
\end{aligned}$$

Theorem (The weak law of large numbers)

Let $(X_n)_{n=1,2,\ldots}$ be *i.i.d.* random variables with the mean μ and the variance $\sigma^2 < \infty$. For $\varepsilon > 0$,

$$\lim_{n\to\infty} \Pr\left[\left|\frac{X_1 + \cdots + X_n}{n} - \mu\right| \geq \varepsilon\right] = 0.$$

Verification of Monte Carlo Approximation

$$\Gamma \equiv \varepsilon : \texttt{real}, \mu : \texttt{real}, \sigma : \texttt{real}, \texttt{d} : P[\texttt{X}], \texttt{h} : \texttt{X} \Rightarrow \texttt{real}$$

$$\Psi \equiv \{\sigma^2 = \mathbb{V}_{\texttt{x}\sim\texttt{d}}[\texttt{h x}], \mu = \mathbb{E}_{\texttt{x}\sim\texttt{d}}[\texttt{h x}], \varepsilon > \texttt{0}, \mathit{integrable}\ \texttt{d h}, \mathit{integrable}\ \texttt{d h}^2\}$$

$$\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathit{montecarlo} : \texttt{nat} \Rightarrow P[\texttt{real}] \mid \forall \texttt{n} : \texttt{nat}.\texttt{n} > \texttt{0} \rightarrow \Pr_{\texttt{y}\sim\texttt{r n}}[|\texttt{y} - \mu| \geq \varepsilon] \leq \sigma^2/\texttt{n}\varepsilon^2$$

$\mathit{integrable}\ \mu\ f$	The expected value $\mathbb{E}_{x\sim\mu}[f\ x]$ exists as a finite value (f is integrable w.r.t. μ).

Theorem (The weak law of large numbers)

Let $(X_n)_{n=1,2,\dots}$ be *i.i.d.* random variables with the mean μ and the variance $\sigma^2 < \infty$. For $\varepsilon > 0$,

$$\lim_{n\to\infty} \Pr\left[\left|\frac{X_1 + \cdots + X_n}{n} - \mu\right| \geq \varepsilon\right] = 0.$$

Verification of Monte Carlo Approximation

$$\Gamma \equiv \varepsilon : \texttt{real}, \mu : \texttt{real}, \sigma : \texttt{real}, \texttt{d} : P[\texttt{X}], \texttt{h} : \texttt{X} \Rightarrow \texttt{real}$$

$$\Psi \equiv \{\sigma^2 = \mathbb{V}_{\texttt{x}\sim\texttt{d}}[\texttt{h x}], \mu = \mathbb{E}_{\texttt{x}\sim\texttt{d}}[\texttt{h x}], \varepsilon > \texttt{0}, \mathit{integrable}\ \texttt{d h}, \mathit{integrable}\ \texttt{d h}^2\}$$

$$\Gamma \mid \Psi \vdash_{\mathrm{UPL}} \mathit{montecarlo} : \texttt{nat} \Rightarrow P[\texttt{real}] \mid \forall \texttt{n} : \texttt{nat}.\texttt{n} > \texttt{0} \to \Pr_{\texttt{y}\sim\texttt{r n}}[|\texttt{y} - \mu| \geq \varepsilon] \leq \sigma^2/\texttt{n}\varepsilon^2$$

$$\Gamma \mid \Psi \vdash_{\mathrm{PL}} \forall \texttt{n} : \texttt{nat}.\mathit{integrable}\ (\mathit{montecarlo}\ \texttt{n})(\lambda \texttt{x}.\texttt{x}) \wedge \mathit{integrable}\ (\mathit{montecarlo}\ \texttt{n})(\lambda \texttt{x}.\texttt{x}^2)$$

$\mathit{integrable}\ \mu\ f$	The expected value $\mathbb{E}_{x\sim\mu}[f\ x]$ exists as a finite value (f is integrable w.r.t. μ).

The integrability of the program is required to prove the UPL-judgment.

The Proof of Monte Carlo Method

- We proved according to the proof outline shown in [Sato+, POPL2019].
- We additionally need to prove the integrability of the program.
- Our proof requires a large number of steps of equational reasoning because we have not implemented automation.

Pen and paper proof: 7 lines

$$\mathbb{E}_{y\sim(t_n \gg\!= (\lambda m.\mathtt{d} \gg\!= (\lambda x.\mathtt{return}\ ((\mathtt{h}\ x + m * n)/(S\ n)))))}[y]$$

$$= \mathbb{E}_{y\sim(t_n \otimes \mathtt{d} \gg\!= (\lambda (m,x).\mathtt{return}\ ((\mathtt{h}\ x + m * n)/(S\ n))))}[y]$$

$$= \mathbb{E}_{(m,x)\sim t_n \otimes \mathtt{d}}\left[\frac{\mathtt{h}\ x + m * n}{S\ n}\right]$$

$$= \frac{1}{S\ n} * \mathbb{E}_{(m,x)\sim t_n \otimes \mathtt{d}}[\mathtt{h}\ x] + \frac{n}{S\ n} * \mathbb{E}_{(m,x)\sim t_n \otimes \mathtt{d}}[m]$$

$$= \frac{1}{S\ n} * \mathbb{E}_{x\sim \mathtt{d}}[\mathtt{h}\ x] + \frac{n}{S\ n} * \mathbb{E}_{m\sim t_n}[m]$$

$$= \frac{1}{S\ n} * \mu + \frac{n}{S\ n} * \mathbb{E}_{m\sim t_n}[m]$$

$$= \mu$$

In Isabelle: around 100 lines

```
(* expectation *)
 (* subst with IH *)
  apply(rule pl_subst[where u="hp_expect var1 hp_id" and t="var6"],simp add
   apply(rule pl_eq_sym)
   apply(rule pl_andEl[where ψ="hp_var var1 hp_id =PL var5 ^t 2 /t hp_real v
   apply(rule pl_impE[where ψ="hp_const 0 <PL var2"])
    apply(rule pl_ax,simp)
   apply(rule pl_ax,simp)
 (* nemerical transformation *)
  apply(rule pl_subst[of "λt. (var6 *t hp_real var2 +t t) /t hp_real (hp_suc
  apply(rule pl_subst[OF _ pl_times_distrib[of _ _ var6]],simp add: hp_defi
  apply(rule pl_subst[OF _ pl_eq_sym[OF pl_div_timesr[of _ _ var6]]],simp a
  apply(rule pl_subst[where u="hp_const (1::real)" and t="hp_real (hp_const
   apply(rule pl_subst[OF _ pl_real_const[where n=1,simplified]],simp add: 
   apply(rule pl_eq_refl)

   (* var2 + 1 / var2 + 1 = 1 *)
  apply(rule pl_subst[OF _ pl_eq_sym[OF pl_suc]],simp add: hp_definitions)
  apply(rule pl_subst[OF _ pl_eq_sym[OF pl_real_plus]],simp add: hp_definit
  apply(rule pl_subst[OF _ pl_plus_com[of _ _ "hp_real var2"]],simp add: hp_
  apply(rule pl_subst[where u="(hp_real var2 +t hp_real (hp_const 1)) /t (hp
   apply(rule pl_eq_sym)
   apply(rule pl_impE[OF pl_div_div[where t="hp_real var2 +t hp_real (hp_cor
   apply(rule pl_impE[OF pl_rorder_plust[of _ _ "hp_const 0" "hp_real var2 
   apply(rule pl_subst[OF _ pl_real_const[where n=0,simplified]],simp add: 
   apply(rule pl_andI)
    apply(rule pl_impE[OF pl_order_nat_real])
    apply(rule pl_ax,simp)
   apply(rule pl_impE[OF pl_order_nat_real])
   apply(rule pl_order_const,simp)

  apply(rule pl_times_right1)
```

Outline

PPV: Syntax and Typing System

The programming language: **HPProg**

$$\begin{array}{lcl} T & ::= & \mathsf{unit} \mid \mathsf{nat} \mid \mathsf{bool} \mid \mathsf{real} \mid \mathsf{preal} \mid T \times T \mid T \Rightarrow T \mid P[T], \\ e & ::= & x \mid c \mid f \mid e\ e \mid \lambda x.e \mid \langle e, e \rangle \mid \pi_i(e) \mid \text{rec\_nat}\ e\ e \\ & & \mid \text{return}\ e \mid \text{bind}\ e\ e \mid \text{Bernoulli}(e) \mid \text{Gauss}(e, e). \end{array}$$

$P[T]$ is the type of probability distributions on T.

Typing rules are standard.

$$\frac{\Gamma \vdash e : T}{\Gamma \vdash \text{return}\ e : P[T]} \qquad \frac{\Gamma \vdash e : \mathsf{real}}{\Gamma \vdash \text{Bernoulli}(e) : P[\mathsf{bool}]}$$

$$\frac{\Gamma \vdash e : P[T] \qquad \Gamma \vdash f : T \Rightarrow P[T']}{\Gamma \vdash \text{bind}\ e\ f : P[T']} \qquad \frac{\Gamma \vdash e : \mathsf{real} \qquad \Gamma \vdash e' : \mathsf{real}}{\Gamma \vdash \text{Gauss}(e, e') : P[\mathsf{real}]}$$

Formalization of PPV

We shallowly embed PPV.

$$\text{Type } T \xrightarrow{\text{Interpretation}} \text{An object } [\![T]\!] \text{ of } \mathbf{QBS}$$
$$\text{Typed term } \Gamma \vdash e : T \xrightarrow{\text{Interpretation}} \text{A morphism } [\![e]\!] : [\![\Gamma]\!] \to [\![T]\!]$$

In Isabelle/HOL

definition *"hpprog_typing* Γ *e T* $\equiv$ *e* $\in \Gamma \to_Q$ *T"*

QBS

- Objects $\cdots$ Quasi-Borel spaces.
- Morphisms $\cdots$ Structure-preserving functions.

$\Gamma \to_Q T =$ the set of all morphisms from Γ to T.

Similarly, logics of PPV are defined according to its semantics.

Formalization of PPV

De Bruijn index

definition "*var1* $\equiv$ *snd*"
lemma *hpt_var1*:
"Γ,,Z $\vdash_t$ *var1* ;; Z"

definition "*var2* $\equiv$ *snd* $\circ$ *fst*"
lemma *hpt_var2*:
"Γ,,Z,,Y $\vdash_t$ *var2* ;; Z"
*Γ,,Z,,Y = (Γ $\bigotimes_Q$ Z) $\bigotimes_Q$ Y

definition "λ_t $\equiv$ *curry*"
lemma *hpt_abs*:
assumes "Γ,,X $\vdash_t$ t ;; T"
shows "Γ $\vdash_t$ λ_t t ;; X $\Rightarrow_Q$ T"

Ex. $\Gamma, y : Y \vdash (\lambda x.x)\ y : Y$

lemma "Γ,,Y $\vdash_t$ (λ_t *var1*) \$$_t$ *var1* ;; Y"

De Bruijn index makes reasoning cumbersome.

The Original PPV vs Our Mechanized PPV

Conditioning

Our mechanized PPV does not support the conditioning.

- We use the probability monad[Heunen+, LICS2017] on **QBS**.
- The original PPV uses the σ-finite measure monad[Scibior+, POPL2018] on **QBS**.

The probability monad is constructed from the Giry monad which is included in the standard library HOL-Probability.

The Original PPV vs Our Mechanized PPV

Integrability

We use the following Eqs. in the proof of the Monte Carlo approximation.

$$\mathbb{E}_{x \sim d}[f\ x + g\ x] = \mathbb{E}_{x \sim d}[f\ x] + \mathbb{E}_{x \sim d}[g\ x]. \tag{1}$$

$$\mathbb{V}_{(x,y) \sim d_1 \otimes d_2}[f\ x + g\ y] = \mathbb{V}_{x \sim d_1}[f\ x] + \mathbb{V}_{y \sim d_2}[g\ y]. \tag{2}$$

(1) holds if

- f and g are non-negative, or
- f and g are integrable w.r.t. d.

In the proof of (2), we use (1) with functions which might be negative.

Integrability is necessary!

Conclusion

- Formalizing quasi-Borel spaces in Isabelle/HOL
- Formalizing a core part of PPV.
  (The language, Assertion logic, Unary logic)
  - Our PPV does not support conditioning.
  - We added integrability in the logic because it is necessary.
- Verification of the Monte Carlo method on mechanized PPV (including the integrability).

The formalization of quasi-Borel spaces is available at AFP*.
The entire formalization is available at author's repository**.

* Quasi-Borel Spaces, Archive of formal proofs, 2022.
** `https://github.com/HirataMichi/PPV`

Future Works

- Conditioning
  We need to formalize the σ-finite measure monad to support conditioning.
- Proof automation
  It may reduce cost of verification to prove simple Eqs. semantically, rather than apply rules manually.
- Relational program logic
  We expect no major difficulties.

  Applications
  - Sample size required in importance sampling
  - Differential privacy