An introduction to Probabilistic Programming. Part 1: discrete probability

EPIT - CIRM, May 2026

Christine Tasson

isae SUPAERO
Institut Supérieur de l'Aéronautique et de l'Espace

Probabilistic programming examples in μ-PPL and byo-PPL.

Baudart 2024, *muPPL, a PPL prototype in Python, https://github.com/gbdrt/mu-ppl*

A probabilitic program is a random variable

Examples

Sum of two Dice

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b
```

```
let dice prob () : int =
  let a = sample prob (randint 1 6) in
  let b = sample prob (randint 1 6) in
  a + b
```

Sum of two Dice

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b
```

```
let dice prob () : int =
  let a = sample prob (randint 1 6) in
  let b = sample prob (randint 1 6) in
  a + b
```

dice is a random variable whose

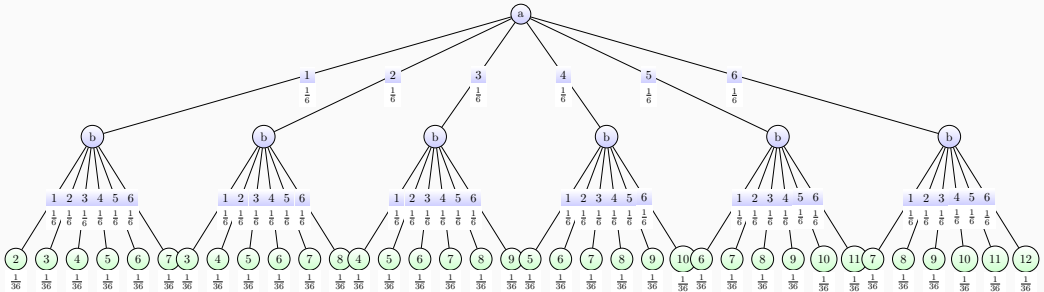
- Experiment: a through of two independent dice,
  - Universe (or potential samples): $\{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3, 4, 5, 6\}$,
  - Samples in the joint distribution: (a, b) with probability $\frac{1}{6} * \frac{1}{6}$,
- Outcome: $a + b$, the sum of samples

Sum of two Dice

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b
```

```
let dice prob () : int =
  let a = sample prob (randint 1 6) in
  let b = sample prob (randint 1 6) in
  a + b
```

The sampling tree of the random variable dice is:



Sum of two Dice

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b
with Enumeration():
    dist: Categorical[int] = infer(dice)
```

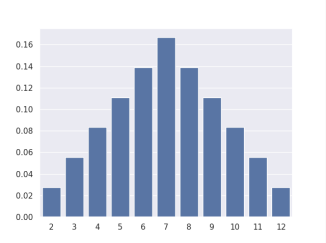
```
let dice prob () : int =
  let a = sample prob (randint 1 6) in
  let b = sample prob (randint 1 6) in
  a + b

let _ = infer dice ()
```

The discrete probability distribution (or law) of dice is the probabilistic mass function (PMF):

$$\begin{aligned} [\![\text{dice}]\!] : \quad \mathbb{N} &\to \mathbb{R}^+ \\ k &\mapsto \sum_{a=1}^{6}\sum_{b=1}^{6} \tfrac{1}{36}\delta_{a+b}(k) \end{aligned}$$

with expected value: $\mathbb{E}(\text{dice}) = \sum [\![dice]\!](k) * k$

The PMF can be represented by a histogram:



Random walk

```
def walk(n:int) → int:
  s = 0
  for k in range(n):
    s += 1 if sample(Bernoulli(0.5)) else -1
  return s
```

```
let rec walk prob n : int =
  if (n <= 0) then 0
  else
    let b = sample prob (bernoulli ~p:0.5) in
    walk prob (n-1) + (if b then 1 else -1)
```

Random walk

```
def walk(n:int) → int:
  s = 0
  for k in range(n):
    s += 1 if sample(Bernoulli(0.5)) else -1
  return s
```

```
let rec walk prob n : int =
  if (n <= 0) then 0
  else
    let b = sample prob (bernoulli ~p:0.5) in
    walk prob (n-1) + (if b then 1 else -1)
```

walk is a random variable whose

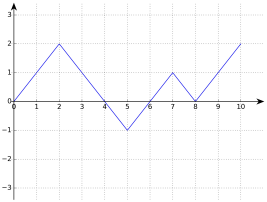
Experiment: flip n fair coins,

Universe (or potential samples): $\{\text{true}, \text{false}\}^n$,

Samples in the joint distribution: $(b_0, \ldots, b_{n-1})$ with probability 0.5^n,

Outcome: $\sum_{i=0}^{n-1} 1 * \delta^{b_i}_{\texttt{true}} - 1 * \delta^{b_i}_{\texttt{false}}$

An example of sampling trace (experiment)



Random walk

```
def walk(n:int) → int:
  s = 0
  for k in range(n):
    s += 1 if sample(Bernoulli(0.5)) else -1
  return s
with Enumeration:
  RW: Categorical[int] = infer(walk, 10)
```

```
let rec walk prob n : int =
  if (n <= 0) then 0
  else
    let b = sample prob (bernoulli ~p:0.5) in
    walk prob (n-1) + (if b then 1 else -1)

let _ = infer walk 10
```

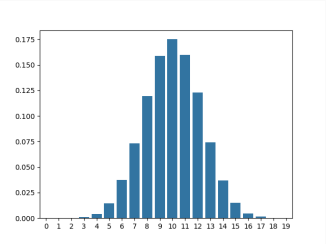
The semantics of walk n is a parameterized discrete probability distribution:
$\forall n \in \mathbb{N}\ [\![\text{walk n}]\!] : \mathbb{N} \to \mathbb{R}^+$
with expected value:

$$\mu = \mathbb{E}([\![\text{walk n}]\!]) = \sum [\![\text{walk n}]\!](k) * k$$

and standard deviation

$$\sigma^2 = \mathbb{V}([\![\text{walk n}]\!]) = \mathbb{E}(([\![\text{walk n}]\!] - \mu)^2)$$

Its law can be represented by a histogram:



Loops and recursive functions

```
def StoppingTime(d) → int:
    time = 0
    while sample(d):
        time = time + 1
    return time
```

```
let rec vonNeumann prob d : bool =
  let a = sample prob d in
  let b = sample prob d in
  if a != b then a
  else vonNeumann prob d
```

Loops and recursive functions

```
def StoppingTime(d) → int:
    time = 0
    while sample(d):
        time = time + 1
    return time
```

```
let rec vonNeumann prob d : bool =
  let a = sample prob d in
  let b = sample prob d in
  if a != b then a
  else vonNeumann prob d
```

StoppingTime is a random variable whose

Experiment: sample i.i.d. copies of distribution d with support X.

Universe (or potential samples): $X^{\mathbb{N}}$,

Samples in the joint distribution: $(b_k)_{k\in\mathbb{N}}$

Outcome: n for $\mathrm{true}^n\,\mathrm{false}$

vonNeumann is a random variable whose

Experiment: sample i.i.d. copies of distribution d with support X.

Universe (or potential samples): $(X \times X)^{\mathbb{N}}$,

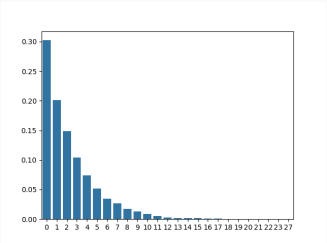
Samples in the joint distribution: $(a_k, b_k)_{k\in\mathbb{N}}$

Outcome: true after $(b,b)^k(\mathrm{true}, \mathrm{false})$ or false after $(b,b)^k(\mathrm{false}, \mathrm{true})$ for any k.

Loops and recursive functions

```
def StoppingTime(d) → int:
    time = 0
    while sample(d):
        time = time + 1
    return time
with ImportanceSampling(num_particles=100):
    ST = infer(StoppingTime, Bernoulli(0.7))
```

StoppingTime (Bernoulli(0.7)) is a discrete random variable whose law is represented by the histogram:




```
let rec vonNeumann prob d : bool =
  let a = sample prob d in
  let b = sample prob d in
  if a != b then a
  else vonNeumann prob d

let _ = infer vonNeumann (bernoulli ~p:0.2)
```

vonNeumann (bernoulli ~p:0.2) is a discrete boolean random variable whose mean is: 0.5 ! It is a fair coin.

Importance Sampling, an approximate inference algorithm

Random variable

A **Probabilistic program** is the random variable whose values are the outcome of its execution.

Independent executions

If the program has no **effect**, then its executions are independent and follow the same law.

Importance Sampling, an approximate inference algorithm

Random variable

A **Probabilistic program** is the random variable whose values are the outcome of its execution.

Independent executions

If the program has no **effect**, then its executions are independent and follow the same law.

Law of large numbers

Run n times an effect-free probabilistic program and store the outputs $x_1, \ldots, x_n$.
Then, $\frac{x_1+\cdots+x_n}{n}$ approximates $\mathbb{E}(X)$ and $\frac{g(x_1)+\cdots+g(x_n)}{n}$ approximates $\mathbb{E}(g(X))$ (if finite).

Importance Sampling, an approximate inference algorithm

Random variable

A **Probabilistic program** is the random variable whose values are the outcome of its execution.

Independent executions

If the program has no **effect**, then its executions are independent and follow the same law.

Law of large numbers

Run n times an effect-free probabilistic program and store the outputs $x_1, \ldots, x_n$.

Then, $\frac{x_1+\cdots+x_n}{n}$ approximates $\mathbb{E}(X)$ and $\frac{g(x_1)+\cdots+g(x_n)}{n}$ approximates $\mathbb{E}(g(X))$ (if finite).

Monte-Carlo Simulation: histograms approximate distributions:

For a random variable X, $\frac{1}{n}\#\{i \mid x_i = x\}$ approximates $\mathbb{E}(\mathbb{1}_{X=x}) = \mathbb{P}(X = x)$

Summary: A discrete probabilistic program is a random variable

Samples in distributions

for instance $\mathrm{RandInt}$, $\mathrm{Bernoulli}\ \mathrm{p}$, $\mathrm{Categorical}$, $\mathrm{Empirical}$

Outputs discrete values

for instance of type unit, bool, int

Its law is a probability distribution that can be computed by

Exact inference: enumeration of the tree of sampling traces

Approximate inference: importance sampling, that is Monte-Carlo simulation.

Summary: A discrete probabilistic program is a random variable

Samples in distributions

for instance $\mathrm{RandInt}$, $\mathrm{Bernoulli}\ \mathrm{p}$, $\mathrm{Categorical}$, $\mathrm{Empirical}$

Outputs discrete values

for instance of type unit, bool, int

Its law is a probability distribution that can be computed by

Exact inference: enumeration of the tree of sampling traces

Approximate inference: importance sampling, that is Monte-Carlo simulation.

We need tools to prove properties of probabilistic programs:

Operational Semantics, Denotational Semantics, Verification, Equivalence of programs.

Probabilistic semantics

Introduction

What is semantics

Operational semantics

Transition systems whose configurations are the states of an abstract machine (Turing machine, pushdown automata, Krivine machine,...) Or terms (rewriting),...

Peter Landin (1930-2009) *The Mechanical Evaluation of Expressions (1966)*

Denotational semantics

Programs denote or represent functions acting on values or memory states.

Christopher Strachey (1916-1975) *Towards a formal semantics (1964)*

Dana Scott (1932–) and Christopher Strachey *Towards a mathematical Semantics of Computer Languages (1971)*

Probabilistic semantics

Probabilistic programs

Describe statistical models

Compute exactly or approximately characteristics of the random variable associated to the program

Formal methods make it possible to prove that

a probabilistic program describes the correct statistical model,

the implementation of an inference algorithm is correct,

program transformations are correct.

Dexter Kozen, *Semantics of probabilistic programs, 1979*

Probabilistic semantics

Imperative programming

Syntax of a simplified μ-PPL

$$
\begin{aligned}
t &::= \mathrm{None} \mid \mathrm{bool} \mid \mathrm{int} \mid t \times t \mid \mathrm{dist}(\mathrm{bool}) \\
e &::= c \mid x \mid (e,\ e) \mid op(e) \\
s &::= \mathrm{pass} \mid x = e \mid x = f(e) \mid \mathrm{if}\ e\mathrm{:}\ s\ \mathrm{else}\mathrm{:}\ s \mid s;\ s \\
&\quad \mid x = \mathrm{sample}(e) \\
d &::= \mathrm{def}\ f(x)\mathrm{:}\ s\ \mathrm{return}\ e \mid d\ d
\end{aligned}
$$

The only distribution we consider is $\mathrm{Bernoulli}(\mathrm{p})$, there are many other presentations:

$$
\begin{aligned}
\mathrm{coin} &= \mathrm{Bernoulli}\,(0.5) \\
\mathrm{flip}\ p &= \mathrm{sample}(\mathrm{Bernoulli}\,(\mathrm{p})) \\
\mathrm{e} \oplus_p \mathrm{e'} &= x = \mathrm{sample}(\mathrm{Bernoulli}\,(\mathrm{p}));\ \mathrm{if}\ x\mathrm{:}\ e\ \mathrm{else}\mathrm{:}\ e'
\end{aligned}
$$

Discrete Random Variable

Definition
A random variable $X : \Omega \to \mathbb{R}$ is discrete if it take a countable number of values.

The probability mass function (PMF) of a discrete random variable X is a function $f : \mathbb{R} \to [0, 1]$ such that: $f(x) = \mathbb{P}(X = x) = \mathbb{P}(\{\omega \in \Omega \mid X(\omega) = x\})$.

Discrete Random Variable

Definition
A random variable $X : \Omega \to \mathbb{R}$ is discrete if it take a countable number of values.

The probability mass function (PMF) of a discrete random variable X is a function $f : \mathbb{R} \to [0, 1]$ such that: $f(x) = \mathbb{P}(X = x) = \mathbb{P}(\{\omega \in \Omega \mid X(\omega) = x\})$.

The mean (expected value), written $\mathbb{E}(X)$ of a discrete random variable X is:

$$\mathbb{E}(X) = \sum_x x \, \mathbb{P}(X = x) = \sum_x x \, f(x)$$

when the sum exists (it is always the case when $X > 0$).

Discrete Random Variable

Definition
A random variable $X : \Omega \to \mathbb{R}$ is discrete if it take a countable number of values.

The probability mass function (PMF) of a discrete random variable X is a function $f : \mathbb{R} \to [0, 1]$ such that: $f(x) = \mathbb{P}(X = x) = \mathbb{P}(\{\omega \in \Omega \mid X(\omega) = x\})$.

The mean (expected value), written $\mathbb{E}(X)$ of a discrete random variable X is:

$$\mathbb{E}(X) = \sum_x x \, \mathbb{P}(X = x) = \sum_x x \, f(x)$$

when the sum exists (it is always the case when $X > 0$).

If $g : \mathbb{R} \to \mathbb{R}$ measurable and X is a discrete random variable, then $g(X)$ is a discrete random variable

$$\mathbb{E}(g(X)) = \sum_x g(x) \, \mathbb{P}(X = x)$$

How to represent Discrete Probability Distributions

Data Structure:

- **Support** : output values
- **Mass functions** : probability associated to each value
- **Mean** : when there is an analytic value

Operations:

- **Random generator** i.e. sampling
- **Approximated expected value** given or computed by Monte-Carlo simulation

Example of discrete random variables

Bernoulli $X \sim \mathcal{B}(p)$
Toss of a coin of bias p.

Binomial $X \sim \mathrm{Bin}(n, p)$
Sum of tosses of n coins of bias p.

Empirical/Uniform $X \sim \mathrm{Uniform}(xs)$
Every value in xs is equally likely.

Categorical $X \sim \mathcal{C}(xs, ps)$
Value $xs[i]$ has probability $ps[i]$.

Semantics of expressions

An **environnement**[1] γ maps each typed variable in Γ to a value of matching type.

An **expression** $\Gamma \vdash e : t$ computed in an environment $\gamma \in \Gamma$ is evaluated to a value of type t: its Semantics is a function $[\![e]\!] : \Gamma \to t$

$$
\begin{array}{lcl}
[\![c]\!]^{\varphi}_{\gamma} & = & c \\
[\![x]\!]^{\varphi}_{\gamma} & = & \gamma(x) \\
[\![op(e)]\!]^{\varphi}_{\gamma} & = & op([\![e]\!]^{\varphi}_{\gamma}) \\
[\![(e_1,\, e_2)]\!]^{\varphi}_{\gamma} & = & ([\![e_1]\!]^{\varphi}_{\gamma}\,, [\![e_2]\!]^{\varphi}_{\gamma})
\end{array}
$$

[1] Environement is sometimes called memory, state,...

Sampling semantics (definitions)

Assume that in a statement, each sample is associated to a unique key κ. Let $\mathscr{K}$ be the set of those keys.

Sampling trace[2]:

a trace σ of samplings is a map that associates to each key a sample in the support of the sampled distribution. Let Ω be the set of sampling traces.

Sampling semantics of statements: $[\![s]\!] : \Gamma \times \Omega \times \mathbb{R}^+ \to \Gamma \times \Omega \times \mathbb{R}^+$

$[\![s]\!](\gamma, \sigma, \rho) = (\gamma', \sigma', \rho')$ means that the statement s may consume samples from the sampling trace σ with induced probability p and use values of variables stored in the environment γ to generate the trace σ' of remaining samples, the updated probability $\rho' = p * \rho$, and the updated environment γ'.

[2] A sampling trace may also be named entropy, oracle, probabilistic scheduling, random data base...

Sampling semantics of statements (selected rules)

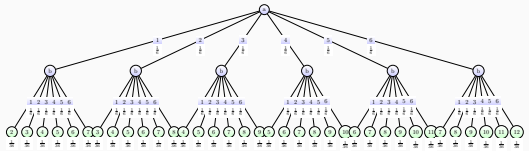
$$\frac{}{[\![x = \text{sample}(\mu,\ \kappa)]\!]\,(\gamma, \sigma + [\kappa \leftarrow v], \rho) = (\gamma + [x \leftarrow v], \sigma', \rho * \mu(v))}$$

$$\frac{[\![s1]\!]\,(\gamma, \sigma, \rho) = (\gamma', \sigma', \rho') \qquad [\![s2]\!]\,(\gamma', \sigma', \rho') = (\gamma'', \sigma'', \rho'')}{[\![s1;\ s2]\!]\,(\gamma, \sigma, \rho) = (\gamma'', \sigma'', \rho'')}$$

$$\frac{\text{def } f(x)\text{: } s \text{ return } e \qquad [\![s]\!]\,([x \leftarrow v], \sigma, 1) = (\gamma, \sigma', \rho)}{f : \lambda v \lambda \sigma.([\![e]\!]_\gamma, \rho)}$$

Example

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b
```



Sampling semantics (exercises)

```
def Binomial(n: int, d) → int:
    s = 0
    for k in range(n):
        s += sample(d)
    return s
```

```
def FlipSum(n: int, d) → int:
    x = sample(d)
    sum = 0
    for k in range(n):
        sum += x
    return sum
```

Stochastic semantics (definitions)

A **stochastic matrix** $k : X \rightsquigarrow Y$, is a probability distribution on Y parameterized by X

$k : X \times Y \to \mathbb{R}^+$ is a matrix

$\forall b \in Y,\ k(-, b) : X \to \mathbb{R}^+$ is a function

$\forall x \in X,\ k(x, -) : Y \to \mathbb{R}^+$ is a probability distribution over Y

A **statement** s transforms an environnement into a probability distribution of updated environments.

It is interpreted by a stochastic matrix:

$$[\![s]\!] : \Gamma \rightsquigarrow \Gamma$$

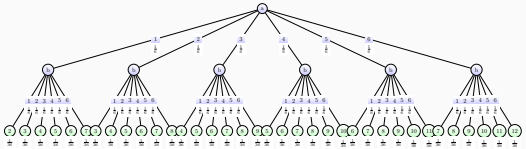
Stochastic semantics (selected rules)

$[\![s]\!] : \Gamma \rightsquigarrow \Gamma$

$$
\begin{aligned}
[\![x = \text{sample}(e)]\!]^{\varphi}_{\gamma,\gamma'} &= \sum_{v \in \text{supp}(\mu)} \mu(v)\ \delta^{\gamma'}_{\gamma + [x \leftarrow v]} \quad \textit{with } \mu = [\![e]\!]^{\varphi}_{\gamma} \\
[\![s_1;\ s_2]\!]^{\varphi}_{\gamma,\gamma''} &= \sum_{\gamma' \in \Gamma} [\![s_1]\!]^{\varphi}_{\gamma,\gamma'}\ \ [\![s_2]\!]^{\varphi}_{\gamma',\gamma''} \\
[\![\text{def } f(x)\text{: } s \text{ return } e]\!]^{\varphi} &= \varphi + [f \leftarrow \lambda v.\ \lambda v'.\ k(v,v')] \\
&\quad \textit{with } k(v,v') = \sum_{\gamma} [\![s]\!]^{\varphi}_{[x \leftarrow v],\gamma}\ \ \delta^{v'}_{[\![e]\!]^{\varphi}_{\gamma}}
\end{aligned}
$$

Example

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b
```



Correspondence between sampling and stochastic semantics

Theorem (soundness) For any statement s, any environments γ and γ',

$$[\![s]\!]_{\gamma,\gamma'} = \sum_{\sigma \in \Omega} \delta^{\gamma'}_{\gamma'(\sigma)} * \rho(\sigma) \quad \text{where } (\gamma'(\sigma), \_, \rho(\sigma)) = [\![s]\!](\gamma, \sigma, 1)$$

Example

For each line, prove that the two sides of the equivalence have the same semantics.

```
def coin():
  x = sample Bernoulli(0.5)
  return x
```

```
def coin():
    x = sample(RandInt(1, 2n))
    return x > n
```

```
x = sample(μx); y = sample(μy)
```

```
y = sample(μy); x = sample(μx)
```

Probabilistic semantics

Functional programming

Syntax of a simple byo-PPL

Terms: $\lambda_{\oplus}$

$$t ::= \text{None} \mid \text{bool} \mid \text{int} \mid t \times t \mid \text{dist}(\text{bool}) \mid \text{dist}(\text{int})$$

$$e ::= c \mid x \mid (e, e) \mid op(e) \mid \text{if } e \text{ then } e \text{ else } e \mid \text{rec } f\ x\ e \mid f\ e \mid \text{let } x = \text{sample}(e) \text{ in } e$$

Values V

$$v ::= n \in \mathbb{N} \mid b \in \mathbb{B} \mid (v, w) \mid \text{rec } f\ x\ e \mid \text{coin}$$

The only distribution we consider is $\text{Bernoulli}(\text{p})$, there are many other presentations:

$$\begin{aligned} \text{coin} &= \text{Bernoulli}(0.5) \\ \text{flip } p &= \text{sample}(\text{Bernoulli}(\text{p})) \\ \text{e} \oplus_p \text{e'} &= \text{let } x = \text{sample}(\text{Bernoulli}(\text{p})) \text{ in if } x \text{ then } e \text{ else } e' \end{aligned}$$

Semantics of expressions

An **environnement**[3] γ maps each typed variable in Γ to a value of matching type.

A deterministic **expression** $\Gamma \vdash e : t$ computed in an environment $\gamma \in \Gamma$ is evaluated to a value of type t: its semantics is a function $[\![e]\!] : \Gamma \to V$

$$\begin{array}{lcl} [\![c]\!]^{\varphi}_{\gamma} & = & c \\ [\![x]\!]^{\varphi}_{\gamma} & = & \gamma(x) \\ [\![op(e)]\!]^{\varphi}_{\gamma} & = & op([\![e]\!]^{\varphi}_{\gamma}) \\ [\![(e_1, e_2)]\!]^{\varphi}_{\gamma} & = & ([\![e_1]\!]^{\varphi}_{\gamma}, [\![e_2]\!]^{\varphi}_{\gamma}) \end{array}$$

We denote as $\gamma \vdash e \downarrow v$ when $[\![e]\!]_{\gamma} = v$.

[3] Environement is sometimes called memory, state,...

Sampling semantics (definitions)

Assume that in a statement, each sample is associated to a unique key κ. Let $\mathscr{K}$ be the set of those keys.

Sampling trace[4]:

a trace σ of samplings is a map that associates to each key a sample in the support of the sampled distribution. Let Ω be the set of sampling traces.

Sampling semantics of probabilistic expressions: $(\!|e|\!) : \Gamma \times \Omega \to V \times \mathbb{R}^+$

$(\!|e|\!)(\gamma, \sigma) = (v, \rho)$ means that the expression e evaluates to value v (in potentially several steps) and is denoted as $\gamma, \sigma \vdash e \Downarrow v, \rho$

[4] A sampling trace may also be named entropy, oracle, probabilistic scheduling, random data base...

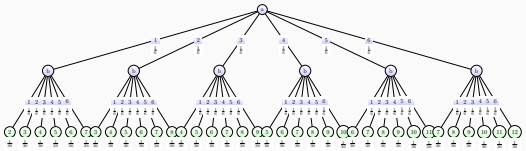
Sampling semantics of statements (selected rules)

$$\frac{\gamma \vdash e_1 \downarrow \mu \qquad \sigma(\kappa) = v \qquad \gamma + [x \leftarrow v_1], \sigma - \kappa \vdash e_2 \Downarrow v_2, \rho}{\gamma, \sigma \vdash \text{let } x = \text{sample}(e_1, \kappa) \text{ in } e_2 \Downarrow v_2, \mu(v) * \rho}$$

$$\frac{\text{rec } f \; x \;\; e \qquad [x \leftarrow v], \sigma \vdash e \Downarrow v', \rho}{f : \lambda\sigma\lambda v \, (\!| e |\!) \, ([x \leftarrow v], \sigma)}$$

Example

```
let dice prob () : int =
  let a = sample prob (randint 1 6) in
  let b = sample prob (randint 1 6) in
  a + b
```



Stochastic semantics (definitions)

Probabilistic expressions transform environments into (sub)probability distributions over expressions.

The discrete **subprobability distribution monad**[5] on a countable set X is denoted $\mathcal{D}(X)$ and its structure is given for $\mu \in \mathcal{D}(X)$ and $f : X \to \mathcal{D}Y$ by

$$\mathrm{ret}(x)(x') = \delta_x^{x'} \qquad \text{and} \qquad \mathrm{bind}(f, \mu)(y) = (\mu \gg\!= f)(y) = \sum_{x \in X} \mu(x) f(x)(y)$$

A **stochastic matrix** $k : X \rightsquigarrow Y$, is a probability distribution on Y parameterized by X

- $k : X \times Y \to \mathbb{R}^+$ is a matrix
- $\forall x \in X,\ k(x, -) : Y \to \mathbb{R}^+$ is a probability distribution over Y

Proposition: Stochastic matrices $X \rightsquigarrow Y$ are in 1:1 correspondence with functions $X \to \mathcal{D}(Y)$.

[5] Sometimes called the Panangaden or the Giry monad, sometimes it is restricted to finite support distributions.

Stochastic semantics (selected rules)

$[\![e]\!] : \Gamma \rightsquigarrow V$

Step $\lambda_p \times \Gamma \rightsquigarrow \lambda_p \times \Gamma$

$$\frac{\gamma \vdash e_1 \downarrow \mu}{\mathrm{step}(\mathrm{let}\ x = \mathrm{sample}(e_1)\ \mathrm{in}\ e_2, \gamma) = \sum_{v \in \mathrm{supp}(\mu)} \mu(v)\, \delta_{(e_2, \gamma + [x \leftarrow v])}}$$

Execution

$$\mathrm{exec}^n(e, \gamma) = \begin{cases} 0 & \text{if } n = 0 \text{ and } e \text{ is not a value} \\ \mathrm{ret}(e) & \text{if } e \text{ is a value} \\ \mathrm{step}(e) \gg\!= \mathrm{exec}^{n-1} & \end{cases}$$

Semantics

$$[\![e]\!]_\gamma = \sup_n \mathrm{exec}^n(e, \gamma)$$

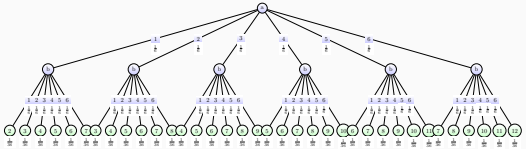
Correspondence between sampling and stochastic semantics

Theorem (soundness) For any expression e, any environments γ,

$$[\![e]\!]_\gamma(v') = \sum_{\sigma \in \Omega} \delta^{v'}_{v(\sigma)} * \rho(\sigma) \quad \text{where } (v(\sigma), \rho(\sigma)) = (\!(e)\!)(\gamma, \sigma)$$

Example

```
let a = sample prob (randint 1 6) in
let b = sample prob (randint 1 6) in
a + b
```



Bayesian inference

conditioning

Marijuana and the army

During Vietnam war in 1975, how to evaluate the proportion of smokers.

Experiment: To the question "do you smoke ?", a soldier toss a coin:

> If the coin is HEADS, she answers yes (even if she does not smoke),
> If the coin is TAIL, she answers the truth.

Thus, the answer can be yes either if the output is HEADS or if she smokes.

Observed Results: 200 answers, 160 yes and 40 no

What is the probability that a soldier smoke ?

Marijuana and the army

During Vietnam war in 1975, how to evaluate the proportion of smokers.

Experiment: To the question "do you smoke ?", a soldier toss a coin:

> If the coin is HEADS, she answers yes (even if she does not smoke),
> If the coin is TAIL, she answers the truth.

Thus, the answer can be yes either if the output is HEADS or if she smokes.

Observed Results: 200 answers, 160 yes and 40 no

What is the probability that a soldier smoke ? 60%

Marijuana and the army - privacy

During Vietnam war in 1975, how to evaluate the proportion of smokers.

Experiment: To the question "do you smoke ?", a soldier toss a coin:

> If the coin is HEADS, she answers yes (even if she does not smoke),
> If the coin is TAIL, she answers the truth.

Thus, the answer can be yes either if the output is HEADS or if she smokes.

Observed Results: 200 answers, 160 yes and 40 no

What is the probability that a soldier smoke ? 60%

What is the probability that a soldier smokes if she answered yes ?

Marijuana and the army - privacy

During Vietnam war in 1975, how to evaluate the proportion of smokers.

Experiment: To the question "do you smoke ?", a soldier toss a coin:

> If the coin is HEADS, she answers yes (even if she does not smoke),
> If the coin is TAIL, she answers the truth.

Thus, the answer can be yes either if the output is HEADS or if she smokes.

Observed Results: 200 answers, 160 yes and 40 no

What is the probability that a soldier smoke ? 60%

What is the probability that a soldier smokes if she answered yes ?

$$\mathbb{P}\left(\text{🌿} \mid \text{YES}\right) = \frac{\mathbb{P}\left(\text{YES} \mid \text{🌿}\right)\ \mathbb{P}\left(\text{🌿}\right)}{\mathbb{P}\left(\text{YES}\right)}$$

Conditional Probability

Definition
The conditional probability is defined when $\mathbb{P}(B) > 0$, then

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

Conditional Probability

Definition
The conditional probability is defined when $\mathbb{P}(B) > 0$, then

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

Two events A and B are independent when one hence all equivalent properties are satisfied $\mathbb{P}(A) = \mathbb{P}(A|B)$ or $\mathbb{P}(B) = \mathbb{P}(B|A)$ or $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$.

Conditional Probability

Definition
The conditional probability is defined when $\mathbb{P}(B) > 0$, then

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

Two events A and B are independent when one hence all equivalent properties are satisfied $\mathbb{P}(A) = \mathbb{P}(A|B)$ or $\mathbb{P}(B) = \mathbb{P}(B|A)$ or $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$.

Properties

if $\mathbb{P}(B) > 0$, then $\mathbb{P}(A) = \mathbb{P}(A|B)P(B) + \mathbb{P}(A|B^c)P(B^c)$.

if $B_1, B_2, \ldots, B_n$ is a partition Ω, then

$$\mathbb{P}(A) = \sum_{i=1}^{n} \mathbb{P}(A|B_i)\mathbb{P}(B_i)$$

Interlude theoretic - Bayes Law

Bayes Law

$$\mathbb{P}(B|A) = \frac{\mathbb{P}(A|B)\,\mathbb{P}(B)}{\mathbb{P}(A)}$$

Bayes Formula

$$\mathbb{P}(B|A) = \frac{\mathbb{P}(A|B)\,\mathbb{P}(B)}{\mathbb{P}(A|B)\,\mathbb{P}(B) + \mathbb{P}(A|B^c)\,\mathbb{P}(B^c)}$$

Generalization, if $\mathbb{P}(A) > 0$ and $B_1, B_2, \ldots, B_n$ is a partition of Ω such that $\forall i\, \mathbb{P}(B_i) > 0$, then

$$\mathbb{P}(B_j|A) = \frac{\mathbb{P}(A|B_j)P(B_j)}{\sum_{i=1}^{n} \mathbb{P}(A|B_i)\mathbb{P}(B_i)}$$

Bayes Law - Programs tests and bugs

Probabilistic Programming and Bayesian Methods for Hackers, C. Davidson-Pilon.

A programmer writes code and tests and wonders: Are there bugs in my code?

A is the event there is NO bug and $\mathbb{P}(A) = p$.
X is the event all tests passed.
assume that there is a 50% chance that the tests passed given that there is a bug.

What is the probability $\mathbb{P}(A|X)$ of the event there is no bug given that all the tests are passed ?

Bayes Law - Programs tests and bugs

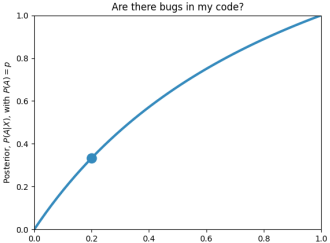
Probabilistic Programming and Bayesian Methods for Hackers, C. Davidson-Pilon.

A programmer writes code and tests and wonders: Are there bugs in my code?

A is the event there is NO bug and $\mathbb{P}(A) = p$.
X is the event all tests passed.
assume that there is a 50% chance that the tests passed given that there is a bug.

What is the probability $\mathbb{P}(A|X)$ of the event there is no bug given that all the tests are passed ?

$$\begin{aligned}
\mathbb{P}(X) &= \mathbb{P}(X, A) + \mathbb{P}(X, \overline{A}) \\
&= \mathbb{P}(X|A)\mathbb{P}(A) + \mathbb{P}(X|\overline{A})\mathbb{P}(\overline{A}) \\
&= 1 \cdot p + 0.5 \cdot (1 - p) = \frac{p + 1}{2} \\
\mathbb{P}(A|X) &= \frac{\mathbb{P}(X|A)\mathbb{P}(A)}{\mathbb{P}(X)} = \frac{2p}{1 + p}
\end{aligned}$$



Conditional Probability and random variables

Discrete

Bayes Law

$$\mathbb{P}(Y = y|X = x) = \frac{\mathbb{P}(X = x|Y = y)\,\mathbb{P}(Y = y)}{\mathbb{P}(X = x)}$$

Conditional Probability and random variables

Discrete

Bayes Law

$$\mathbb{P}(Y = y|X = x) = \frac{\mathbb{P}(X = x|Y = y)\,\mathbb{P}(Y = y)}{\mathbb{P}(X = x)}$$

Bayes Formula

$$\mathbb{P}(Y = y|X = x) = \frac{\mathbb{P}(X = x|Y = y)\,\mathbb{P}(Y = y)}{\sum_b \mathbb{P}(X = x|Y = b)\,\mathbb{P}(Y = b)}$$

Bayesian inference

Exact inference

What is this program doing?

```
def hard_dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    assume (a != b)
    return a + b
```

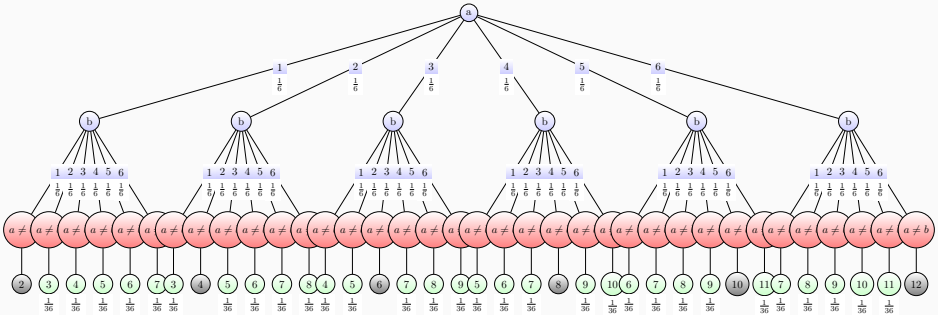
What is this program doing?

```
def hard_dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    assume (a != b)
    return a + b
```

It simulates the conditional **random variable**:

"The sum of two independent dice, given that their values are distinct."

The construction **assume**[6] rejects samples that do not satisfy the property.



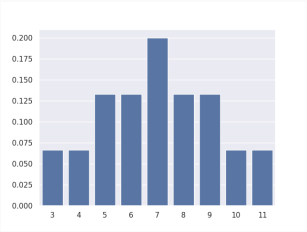
What is the law associated to the program ?

```
def hard_dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    assume (a != b)
    return a + b

with Enumeration():
    dist: Categorical[int] = infer(hard_dice)
    print(dist.stats())
    viz(dist)
    plt.show()
```

The conditional distribution:

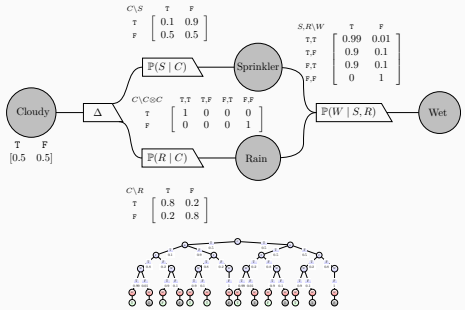
$$[\![\text{hard\_dice}]\!](k) = \mathbb{P}(a+b = k \mid a \neq b)$$

$$= \sum_{a \neq b \in \{1,\dots,6\}^2} \frac{\frac{1}{36}\mathbb{1}_{\{a+b=k\}}}{\sum_{a \neq b \in \{1,\dots,6\}^2} \frac{1}{36}}$$



Bayesian Network: inference through sums

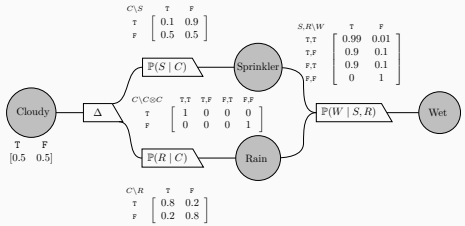
```
def bn() → bool:
    cloudy = sample(Bernoulli(0.5))

    p_s, p_r = (0.1, 0.8) if cloudy else (0.5, 0.2)
    sprinkle = sample(Bernoulli(p_s))
    rain = sample(Bernoulli(p_r))

    p_w = 0.99 if (sprinkle and rain) else 0.9 if
          (sprinkle != rain) else 0
    wet = sample(Bernoulli(p_w))
    assume(wet)
    return rain

with Enumeration():
    dist: Categorical[bool] = infer(bn)
```

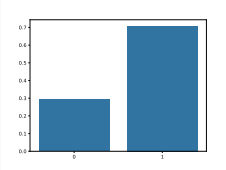


$$\mathbb{P}(\text{Rain} \mid \text{Wet=true}) = \frac{\sum_{c,s,r} \mathbb{P}(\text{Rain} = r \mid c, s, w = \text{T})}{\sum_{c,s,w,r} \mathbb{P}(\text{Rain} = r \mid c, s, r)}$$

Bayesian Network: inference through sums

```
def bn() → bool:
    cloudy = sample(Bernoulli(0.5))

    p_s, p_r = (0.1, 0.8) if cloudy else (0.5, 0.2)
    sprinkle = sample(Bernoulli(p_s))
    rain = sample(Bernoulli(p_r))

    p_w = 0.99 if (sprinkle and rain) else 0.9 if
          (sprinkle != rain) else 0
    wet = sample(Bernoulli(p_w))
    assume(wet)
    return rain

with Enumeration():
    dist: Categorical[bool] = infer(bn)
```





Exact Mass Function

Bayesian inference

Inference is NP-Hard

Inference is NP-Hard

Conditional Probability Task: Given a Bayesian Network and a variable X with value set $|X|$, decide wether $\mathbb{P}(X = x) > 0$, for an $x \in |X|$.

Complexity: The conditional Probability Task is NP-Hard.

Proof:

The problem is NP: given all samples, computing $\mathbb{P}(X = x)$ is linear (multiply all the concerned factors).

Reduction to 3-SAT: given a 3-SAT formula φ, construct in polynomial time a Bayesian Network B_φ with a distinguished random variable X such that $\mathbb{P}(X = x) > 0$ if and only if φ is satisfiable.

Conclusion: inference in PPL is untractable.

Bayesian inference

Approximate inference

What is this program doing ?

```
def success(s:int) → int:
    n = sample(RandInt(10, 20))
    d_success = Binomial(n, 0.5)
    observe(d_success, s)
    return n
```

What is this program doing ?

```
def success(s:int) → int:
    n = sample(RandInt(10, 20))
    d_success = Binomial(n, 0.5)
    observe(d_success, s)
    return n
```

Prior: X uniform between 10 and 20

Conditional law: S the number of success among X toss of a fair coin.

Likelihood:
$\mathbb{P}(S = s|X = n) = \text{Binomial}(\text{n}, 0.5)(s)$

The program success simulates the conditional random variable whose **posterior** is the conditional law of X given $S = s$

"How many times (between 10 *and* 20*) have we toss the Coin given we had s success."*

How to compute the law associated to this program ?

```
def success(s:int) → int:
    n = sample(RandInt(10, 20))
    observe(Binomial(n, 0.5), s)
    return n

with ImportanceSampling(num_particles=100):
    dist: Categorical = infer(success, 10)
```

Monte-Carlo simulation:

N-sample $(X_1, \dots, X_N)$ independent identically distributed with law X uniform on 10 to 20.

Weight: likelihood

$w_i = \mathbb{P}(S = s | X_i = n_i)$

Bayes Formula:

$$\mathbb{P}(X = n | S = s) = \frac{\mathbb{P}(S = s | X = n)\ \mathbb{P}(X = n)}{\mathbb{P}(S = s)} = \frac{\mathbb{P}(S = s | X = n)\ \mathbb{P}(X = n)}{\sum_{k=10}^{20} \mathbb{P}(S = s | X = k) \mathbb{P}(X = k)}$$

Law of Large Number: $\frac{\sum_{i=1}^{N} g(X_i) w_i}{\sum_{i=1}^{N} w_i} \xrightarrow[N \to \infty]{} \mathbb{E}(g(X) | S = s)$.

Mean approximation

Law of Large Number: $h(X_i) = w_i = \mathbb{P}(S = s|X_i)$

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} h(X_i) = \mathbb{E}(h(X)) = \sum_{k=10}^{20} \mathbb{P}(S = s|X = k)\mathbb{P}(X = k) = \mathbb{P}(S = s)$$

Law of Large Number: $\rho(X_i) = \frac{w_i g(X_i)}{\mathbb{P}(S=s)}$

$$\begin{aligned} \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \rho(X_i) &= \mathbb{E}(\rho(X)) = \sum_{k=10}^{20} \frac{\mathbb{P}(S = s|X = k)\mathbb{P}(X = k)g(k)}{\mathbb{P}(S = s)} \\ &= \sum_{k=10}^{20} \mathbb{P}(X = k|S = s)g(k) = \mathbb{E}(g(X)|S = s) \end{aligned}$$

Conclusion:

$$\frac{\sum g(X_i)w_i}{\sum w_i} \xrightarrow[N\to\infty]{} \mathbb{E}(g(X)|S = s).$$

Conclusion

A **probabilistic program** simulates a (conditional) random variable and we want to compute its distribution, its mean, its variance.

Bayesian inference is the design of algorithms to compute conditional distributions but it is hard. That is why we need heuristic and approximate algorithms.

Verifying properties of probabilistic programs, soundness of transformations is also hard and needs formal methods which are the hear of the design, the semantics and the analysis of PPL.