1 Probabilistic Programming

Probabilistic programs are programs that work with randomness in some way. This could mean that the program has access to a random number generator, or that the program is deterministic but we wish to analyze its behavior on random inputs, or some combination of the two. In order to understand and reason about such programs, it is essential to have a solid definition of their semantics.

The foundations of probabilistic program semantics were laid in the early 1980's [1, 2], but the topic has enjoyed a recent resurgence of interest driven by new applications in machine learning and statistical analysis of large datasets. The emergence of languages such as Church, Anglican, Stan, and others, which allow statisticians to construct and sample distributions and perform Bayesian inference, has created a need for sound semantic foundations and tools for specification and reasoning.

Probabilistic reasoning is more difficult than non-probabilistic. Whereas ordinary programs deal with more *truth-functional* reasoning, such as

- Correctness: Is the postcondition always satisfied?
- Termination: Does this program halt on all inputs?
- Worst-case complexity: Does this program always halt within $O(n^2)$ time?

probabilistic reasoning is more *quantitative*:

- Correctness: What is the *probability* that the postcondition is satisfied?
- Termination: Does this program halt with probability at least $3/4$?
- Average-case complexity: What is the *expected halting time*, given this input distribution on inputs of length n?

In this lecture we derive a denotational semantics for a simple imperative language $\mathsf{IMP}^\$$, which is IMP augmented with a source of randomness.

2 $\mathsf{IMP}^\$$ Syntax

Recall from way back in Lecture 8 our imperative language IMP over program variables *Var* ranging over numeric values:

$$\begin{array}{rrl} \mathsf{AExp} & a & ::= x \in \mathit{Var} \mid c \mid a_1 + a_2 \mid a_1 * a_2 \mid \ldots \\ \mathsf{BExp} & b & ::= \mathsf{true} \mid \mathsf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \ldots \mid b_1 \wedge b_2 \mid \neg b \mid \ldots \\ \mathsf{Com} & c & ::= \mathsf{skip} \mid x := a \mid c_1 ; c_2 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c \end{array}$$

where AExp, BExp, and Com denote the set of arithmetic expressions, Boolean expressions, and commands, respectively. To this language we will add a new command:

$$c ::= \ldots \mid x := \mathsf{random}$$

that allows the program to make calls on a random number generator. Every time $x :=$ random is executed, a new independent random number is sampled from some fixed distribution. This could be a coin flip, or it could be a random real number chosen uniformly from the unit interval $[0, 1]$, or something else. But we will assume that it is fixed and its distribution is known, and that each time $x :=$ random is executed, a new independent sample is generated and assigned to x.

Given such a program, we can ask about the probability of certain events. For example, suppose the random number generator gives a fair coin flip; so evaluation of random produces heads (1) or tails (0), each with probability $1/2$. The following program assigns 1 to x with probability $2/3$ and 0 to x with probability $1/3$:

```
x := 1;
y := random;
while y = 1 {
   x := 1 - x;
   y := random;
}
```

Here is another example called the *von Neumann trick* after its inventor, John von Neumann (1903–1957). It is a device for simulating a fair coin with a biased coin. Suppose the random number generator generates a biased coin flip in which the result 1 occurs with probability p, $0 < p < 1$, and 0 occurs with probability $1 - p$. Here is a program that halts with probability 1 and assigns 1 or 0 to x, each with probability $1/2$:

```
x := random;
y := random;
while x = y {
   x := random;
   y := random;
}
```

Each step of the while loop halts with probability $2p(1-p)$ and assigns 1 or 0 to x with equal probability $p(1-p)$, otherwise it tries again. The program eventually halts with probability 1. Note that we do not even care what p is!

3 Semantics

3.1 Recap of IMP semantics

In the denotational semantics of IMP, we interpreted programs as partial functions $[\![p]\!] : S \rightharpoonup S$ (alternatively, total functions $[\![p]\!] : S \to S_\perp$), where S is the set of *states* $s : \mathit{Var} \to \mathbb{Z}$.

- $[\![x := a]\!](s) \triangleq s[[\![a]\!](s)/x]$
- $[\![c_1 \,;\, c_2]\!](s) \triangleq [\![c_2]\!]([\![c_1]\!](s))$
- $[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!](s) \triangleq \begin{cases} [\![c_1]\!](s), & \text{if } [\![b]\!](s) = \mathsf{true} \\ [\![c_2]\!](s), & \text{if } [\![b]\!](s) = \mathsf{false}. \end{cases}$

Recall that we could abbreviate the conditional by defining a "guard" relation $[\![b]\!]$ for tests, treating it like a command. With this definition, the conditional takes the form

$$[\![\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] = [\![b; p]\!] \cup [\![\neg b; q]\!]. \tag{1}$$

The while loop $[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]$ is the least solution W of the equation

$$W = \lambda s.\ \begin{cases} W([\![c]\!](s)), & \text{if } [\![b]\!](s) = \mathsf{true} \\ s, & \text{if } [\![b]\!](s) = \mathsf{false}, \end{cases}$$

that is, the least fixpoint of the monotone map

$$\tau(W) \triangleq \lambda s.\ \begin{cases} W([\![c]\!](s)), & \text{if } [\![b]\!](s) = \mathsf{true} \\ s, & \text{if } [\![b]\!](s) = \mathsf{false}. \end{cases}$$

Thus the while loop is the union of a chain of finite approximants:

$$[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!] = \bigcup_n \tau^n(\varnothing) \qquad\qquad \varnothing \subseteq \tau(\varnothing) \subseteq \tau^2(\varnothing) \subseteq \cdots .$$

3.2 Adding Probability

In our probabilistic semantics, besides adding the random number generator random, we will allow variables to range over the reals $\mathbb{R}$. This is because we want to illustrate how to handle continuous distributions. This will also allow us to use the uniform distribution on the unit interval $[0, 1]$ as a random number generator instead of just a simple discrete coin flip.

But now we will see that many of the mathematical concepts that we have worked with so far must be generalized. The following table lists concepts on the left that we have encountered in ordinary determinstic semantics that will need to be generalized. The list on the right gives the corresponding probabilistic analog.

states	$\Rightarrow$	(sub-)probability measures
predicates	$\Rightarrow$	measurable functions
satisfaction relation $\vDash$	$\Rightarrow$	Lebesgue integral $\int$
binary relations	$\Rightarrow$	Markov kernels
powerset monad	$\Rightarrow$	Giry monad
denotational semantics	$\Rightarrow$	measure transformers
predicate transformers	$\Rightarrow$	measurable function transformers

3.3 Measurable sets and measurable functions

To talk about the probability of events, it is necessary to designate a set $\mathcal{B}$ of *measurable subsets* (or *events*) in our state space S. These are the sets that can have a probability. The set $\mathcal{B}$ must be a *σ-algebra* (closed under complements, countable unions, and countable intersections). A *measurable space* is a pair $(S, \mathcal{B})$, where $\mathcal{B}$ is a σ-algebra.

Here are some typical examples of measurable spaces:

- a *discrete space* is any $(A, 2^A)$ where A a finite or countable set
- $(\mathbb{R}, \mathcal{B})$, where $\mathcal{B}$ are the *Borel sets* of $\mathbb{R}$, the smallest σ-algebra on $\mathbb{R}$ containing all intervals $[a, b]$
- $(\mathbb{R}, \mathcal{L})$, where $\mathcal{L}$ are the *Lebesgue measurable sets* (a little larger than $\mathcal{B}$)
- $([0, 1], \{A \cap [0, 1] \mid A \in \mathcal{B}\})$

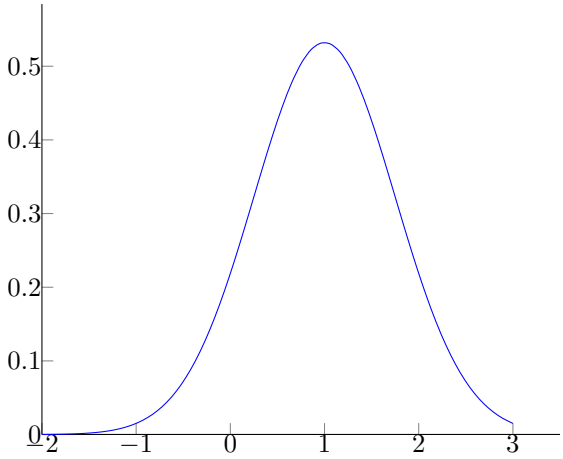
- $(\mathbb{R}^n, \mathcal{B}^{(n)})$, where $\mathcal{B}^{(n)}$ is the smallest σ-algebra containing all the *measurable rectangles* $A_1 \times \cdots \times A_n$ for $A_i \in \mathcal{B}$, $1 \leq i \leq n$. We will take this to be the state space of a program containing n program variables $x_1, \ldots, x_n$, where the i^{th} component of an n-tuple denotes the value of x_i.

A function $f : (S_1, \mathcal{B}_1) \to (S_2, \mathcal{B}_2)$ between measurable spaces is called a *measurable function* if the inverse image of any measurable set in $\mathcal{B}_2$ is in $\mathcal{B}_1$; that is, for any $A \in \mathcal{B}_2$, $f^{-1}(A) = \{s \in S_1 \mid f(s) \in A\} \in \mathcal{B}_1$. The measurable spaces and measurable functions form a category Meas.

You might ask: Why not always take all subsets 2^S as the measurable sets? The answer is that for the reals $\mathbb{R}$, this would violate the basic axioms of set theory. Using the Axiom of Choice, it is possible to construct certain non-Lebesgue-measurable sets (called *Vitali sets*); so we either have to give up the Axiom of Choice or give up some sets being measurable.

3.4 Measures

A *sub-probability measure* on $(S, \mathcal{B})$ is a map $\mu : \mathcal{B} \to [0, 1]$ such that for any countable collection $\mathcal{A}$ of pairwise disjoint measurable sets, $\mu(\bigcup \mathcal{A}) = \sum_{A \in \mathcal{A}} \mu(A)$. By convention, $\mu(\varnothing) = 0$. It is a *probability measure* if $\mu(S) = 1$.



Here are some examples:

- the *Dirac* (*point mass*) *measure* on a point s:

$$\delta_s(A) \triangleq \begin{cases} 1, & s \in A, \\ 0 & s \notin A. \end{cases}$$

- *uniform* (*Lebesgue*) *measure* λ on $[0, 1]$ generated by $\lambda([a, b)) = b - a$.
- If $f : (S_1, \mathcal{B}_1) \to (S_2, \mathcal{B}_2)$ is a measurable function and μ is a measure on S_1, the *pushforward measure* on S_2 is the measure $\mu \circ f^{-1} = \lambda A.\, \mu(f^{-1}(A))$.

3.5 Markov Kernels

Programs c will be interpreted as *Markov kernels* (aka *measurable kernels*, *stochastic kernels*, *stochastic relations*). These are functions $[\![c]\!] : S \times \mathcal{B}^{(n)} \to [0, 1]$ such that

- for fixed $s \in \mathbb{R}^n$, the map $[\![c]\!](s, -)$ is a *subprobability measure* $\mathcal{B}^{(n)} \to [0, 1]$;
- for fixed $A \in \mathcal{B}^{(n)}$, the map $[\![c]\!](-, A)$ is a *measurable function* $\mathbb{R}^n \to [0, 1]$.

Intuitively, for $s \in S$ and $A \in \mathcal{B}$, $[\![c]\!](s, A) \in [0, 1]$ is the probability that, when started in state s, the program halts in a state in A.

Markov kernels are the probabilistic analog of binary relations or partial functions. They form a monad called the *Giry monad*.

To compose two Markov kernels, we integrate "up the middle" using Lebesgue integration:

$$[\![c_1 \,;\, c_2]\!](s, A) \triangleq \int_t [\![c_1]\!](s, dt) \cdot [\![c_2]\!](t, A)$$

It is the probabilistic analog of relational composition. In the discrete case it is just matrix multiplication of stochastic matrices. This is Kleisli composition in the Giry monad.

4 Lifting

Let M be the space of subprobability measures on the state space $(S, \mathcal{B})$. Curried to take its arguments sequentially, a Markov kernel is of type $[\![c]\!] : S \to M$ and can be lifted by integration to a map

$$[\![c]\!]^\dagger : M \to M \qquad\qquad [\![c]\!]^\dagger(\mu)(A) \triangleq \int_t [\![c]\!](t, A) \cdot \mu(dt).$$

This is just Kleisli lifting in the Giry monad; then, like the other examples discussed there, the lifted maps just compose normally.

$$[\![c_1 \,;\, c_2]\!]^\dagger = [\![c_2]\!]^\dagger \circ [\![c_1]\!]^\dagger.$$

So we can think of a measure as a generalized state, and a program as a map taking input measures to output measures. If the input measure is a probability measure μ, then then output measure $[\![c]\!](\mu)$ will in general be a subprobability measure; its total mass $[\![c]\!](\mu)(S)$ is the probability that the program halts.

5 Measure Transformer Semantics

Now we are ready to give the semantics of all the programming constructs of $\mathsf{IMP}^{\$}$. Recall that *states* S are valuations $s : \{x_1, \ldots, x_n\} \to \mathbb{R}$, which we identify with $\mathbb{R}^n$. All of the arithmetic operators are measurable functions and the tests are measurable sets of states, that is, elements of $\mathcal{B}^{(n)}$.

- $[\![x := a]\!](s) \triangleq \delta_{s[[\![a]\!](s)/x]}$. This is just the pushforward measure with respect to the deterministic semantics of IMP. The measure $\delta_{s[[\![a]\!](s)/x]}$ is the point mass (Dirac) measure on $s[[\![a]\!](s)/x]$.
- $[\![c_1 \,;\, c_2]\!](s) \triangleq [\![c_2]\!]^\dagger([\![c_1]\!](s))$.
- $[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!](s) \triangleq \begin{cases} [\![c_1]\!](s), & \text{if } [\![b]\!](s) = \mathsf{true} \\ [\![c_2]\!](s), & \text{if } [\![b]\!](s) = \mathsf{false}. \end{cases}$

If we define a "guard" kernel $[\![b]\!]'(s, A)$ for tests analogous to (1) for IMP,

$$[\![b]\!]'(s, A) \triangleq \begin{cases} 1, & s \in A \wedge [\![b]\!](s) = \mathsf{true} \\ 0, & s \notin A \wedge [\![b]\!](s) = \mathsf{false} \end{cases}$$

we get the same abbreviation, except with $+$ (pointwise sum of measure transformers) instead of $\cup$:

$$[\![\textsf{if } b \textsf{ then } p \textsf{ else } q]\!] = [\![b; p]\!] + [\![\neg b; q]\!].$$

Similarly, the while loop is the least solution W of the equation

$$W = \lambda s. \begin{cases} W([\![c]\!](s)), & \text{if } [\![b]\!](s) = \textsf{true} \\ s, & \text{if } [\![b]\!](s) = \textsf{false}, \end{cases}$$

that is, the least fixpoint of the monotone map

$$\tau(W) \triangleq \lambda s. \begin{cases} W([\![c]\!](s)), & \text{if } [\![b]\!](s) = \textsf{true} \\ s, & \text{if } [\![b]\!](s) = \textsf{false}. \end{cases}$$

But this time, it is in the space of measure transformers, so it is a pointwise limit instead of a union:

$$[\![\textsf{while } b \textsf{ do } c]\!] = \sup_n \tau^n(0) \qquad\qquad 0 \le \tau(0) \le \tau^2(0) \le \cdots .$$

Finally, we have the random assignment. Assuming the random number generator gives the uniform (Lebesgue) measure λ on the Borel sets of $[0, 1]$, the definition is

$$[\![x_i := \textsf{random}]\!](s_1, \ldots, s_n, A_1 \times \cdots \times A_n) \triangleq (\prod_{j \neq i} \delta_{s_j}(A_j)) \cdot \lambda(A_i).$$

However, there was nothing special about λ; any random number generator can be substituted.

6 Predicate Transformer Semantics

Similar to forward-moving measure transformer semantics, there is backward-moving predicate transformer semantics, analogous to the weakest liberal preconditions of Dijkstra that we saw in Lecture 17. However, these now act on measurable functions instead of predicates (which are just 0,1-valued measurable functions).

What can you *observe* about a state? Previously it was whether a state satisfied a predicate: $s \vDash A$. Now, with probability distributions μ, we can ask about the *probability* of an event A. This is just $\mu(A) \in [0, 1]$. More generally, we can ask about the expected value of a measurable function $f : \mathbb{R}^n \to \mathbb{R}$. This is given by the Lebesgue integral:

$$\int_t \mu(dt) \cdot f(t) \in \mathbb{R}.$$

This can be viewed as the probability that the generalized state μ satisfies the generalized predicate f. It is the probabilistic analog of $s \vDash A$.

References

[1] Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981.

[2] N. Saheb-Djahromi. Cpo's of measures for nondeterminism. *Theor. Comput. Sci.*, 12:19–37, 1980.