Probabilistic Functional Programming

Donnacha Oisín Kidney

July 9, 2018

Modeling Probability

How do we model stochastic and probabilistic processes in programming languages?

The Boy-Girl Paradox

1. Mr. Jones has two children. The older child is a girl. What is the probability that both children are girls?
2. Mr. Smith has two children. At least one of them is a boy. What is the probability that both children are boys?

The Boy-Girl Paradox

1. Mr. Jones has two children. The older child is a girl. What is the probability that both children are girls?
2. Mr. Smith has two children. At least one of them is a boy. What is the probability that both children are boys?

Is the answer to $2\ \frac{1}{3}$ or $\frac{1}{2}$?

The Boy-Girl Paradox

1. Mr. Jones has two children. The older child is a girl. What is the probability that both children are girls?
2. Mr. Smith has two children. At least one of them is a boy. What is the probability that both children are boys?

Is the answer to 2 $\frac{1}{3}$ or $\frac{1}{2}$?

Part of the difficulty in the question is that it's ambiguous: can we use programming languages to lend some precision?

An Ad-Hoc Solution i

Using normal features built in to the language.

```
from random import randrange, choice

class Child:
    def __init__(self):
        self.gender = choice(["boy", "girl"])
        self.age = randrange(18)
```

An Ad-Hoc Solution ii

```
from operator import attrgetter

def mr_jones():
    child_1 = Child()
    child_2 = Child()
    eldest = max(child_1, child_2,
                 key=attrgetter('age'))
    assert eldest.gender == 'girl'
    return [child_1, child_2]
```

An Ad-Hoc Solution iii

```
def mr_smith():
    child_1 = Child()
    child_2 = Child()
    assert child_1.gender == 'boy' or \
           child_2.gender == 'boy'
    return [child_1, child_2]
```

Unclear semantics

What contracts are guaranteed by probabilistic functions?
What does it mean *exactly* for a function to be probabilistic?
Why isn't the following[1] "random"?

```
int getRandomNumber()
{
  return 4; // chosen by fair dice roll.
            // guaranteed to be random.
}
```

[1] Randall Munroe. *Xkcd: Random Number*. en. Title text: RFC 1149.5 specifies 4 as the standard IEEE-vetted random number. Feb. 2007. URL: https://xkcd.com/221/ (visited on 07/06/2018).

What about this?

```
children_1 = [Child(), Child()]
children_2 = [Child()] * 2
```

How can we describe the difference between `children_1` and `children_2`?

Underpowered

There are many more things we may want to do with probability distributions.

What about expectations?

```
def expect(predicate, process, iterations=100):
    success, tot = 0, 0
    for _ in range(iterations):
        try:
            success += predicate(process())
            tot += 1
        except AssertionError:
            pass
    return success / tot
```

The Ad-Hoc Solution

```
expect(lambda children: all(child.gender == 'girl'
                            for child in children),
       mr_jones)
expect(lambda children: all(child.gender == 'boy'
                            for child in children),
       mr_smith)
```

Monadic Modeling

A DSL

What we're approaching is a DSL, albeit an unspecified one.

Three questions for this DSL:

- Why should we implement it? What is it useful for?
- How should we implement it? How can it be made efficient?
- Can we glean any insights on the nature of probabilistic computations from the language? Are there any interesting symmetries?

The Erwig And Kollmansberger Approach

First approach[2]:

$$\textbf{newtype}\ \mathit{Dist}\ a = \mathit{Dist}\ \{\mathit{runDist} :: [(a, \mathit{Rational})]\}$$

A distribution is a list of possible events, each tagged with a probability.

---

[2] Martin Erwig and Steve Kollmansberger. "Functional Pearls: Probabilistic Functional Programming in Haskell". In: *Journal of Functional Programming* 16.1 (2006), pp. 21–34. ISSN: 1469-7653, 0956-7968. DOI: 10.1017/S095679680500572l. URL: http://web.engr.oregonstate.edu/~erwig/papers/abstracts.html%5C#JFP06a (visited on 09/29/2016).

A random integer, then, is:

```
type RandInt = Dist Int
```

This lets us encode (in the types) the difference between:

```
children_1 :: [Dist Child]
children_2 :: Dist [Child]
```

As we will use this as a DSL, we need to define the language features we used above:

```
def mr_smith():
    child_1 = Child()
    child_2 = Child()
    assert child_1.gender == 'boy' or \
           child_2.gender == 'boy'
    return [child_1, child_2]
```

1. `=` (assignment)
2. `assert`
3. `return`

Assignment

This is encapsulated by the “monadic bind”:

$$(\gg\!=) :: \mathit{Dist}\ a \to (a \to \mathit{Dist}\ b) \to \mathit{Dist}\ b$$

When we assign to a variable in a probabilistic computation, everything that comes later is conditional on the result of that assignment. We are therefore looking for the probability of the continuation given the left-hand-side; this is encapsulated by multiplication:

$$\begin{aligned} \mathit{xs} \gg\!= f = \mathit{Dist}\ [\ & (y, \mathit{xp} \times \mathit{yp}) \\ & \mid (x, \mathit{xp}) \leftarrow \mathit{runDist}\ \mathit{xs} \\ & ,\ (y, \mathit{yp}) \leftarrow \mathit{runDist}\ (f\ x)] \end{aligned}$$

Assertion

Assertion is a kind of conditioning: given a statement about an event, it either occurs or it doesn't.

$$
\begin{aligned}
&\mathit{guard} :: \mathit{Bool} \to \mathit{Dist}\ () \\
&\mathit{guard}\ \mathit{True}\ \ = \mathit{Dist}\ [((), 1)] \\
&\mathit{guard}\ \mathit{False} = \mathit{Dist}\ [\,]
\end{aligned}
$$

Return

Return is the “unit” value for a distribution; the certain event, the unconditional distribution.

$$\mathit{return} :: a \to \mathit{Dist}\ a$$
$$\mathit{return}\ x = \mathit{Dist}\ [(x, 1)]$$

Putting it all Together

$$
\begin{aligned}
&\mathit{mrSmith} :: \mathit{Dist}\ [\mathit{Child}] \\
&\mathit{mrSmith} = \mathbf{do} \\
&\quad \mathit{child1} \leftarrow \mathit{child} \\
&\quad \mathit{child2} \leftarrow \mathit{child} \\
&\quad \mathit{guard}\ (\mathit{gender}\ \mathit{child1} \equiv \mathit{Boy} \vee \mathit{gender}\ \mathit{child2} \equiv \mathit{Boy}) \\
&\quad \mathit{return}\ [\mathit{child1}, \mathit{child2}]
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{expect} :: (a \to \mathit{Rational}) \to \mathit{Dist}\ a \to \mathit{Rational} \\
&\mathit{expect}\ p\ xs = \frac{\mathit{sum}\ [p\ x \times xp \mid (x, xp) \leftarrow \mathit{runDist}\ xs]}{\mathit{sum}\ [xp \mid (\_, xp) \leftarrow \mathit{runDist}\ xs]}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{probOf} :: (a \to \mathit{Bool}) \to \mathit{Dist}\ a \to \mathit{Rational} \\
&\mathit{probOf}\ p = \mathit{expect}\ (\lambda x \to \mathbf{if}\ p\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0)
\end{aligned}
$$

$$\mathit{probOf}\ (\mathit{all}\ ((\equiv)\ \mathit{Girl} \circ \mathit{gender}))\ \mathit{mrJones} \equiv \frac{1}{2}$$
$$\mathit{probOf}\ (\mathit{all}\ ((\equiv)\ \mathit{Boy} \circ \mathit{gender}))\ \mathit{mrSmith} \equiv \frac{1}{3}$$

Alternative Interpreters i

Once the semantics are described, different interpreters are easy to swap in. Take Monty Hall, for example:

$$
\begin{aligned}
&\textbf{data}\ \mathit{Decision} = \mathit{Decision}\ \{\mathit{stick} \quad :: \mathit{Bool} \\
&\qquad\qquad\qquad\qquad\qquad\quad , \mathit{switch} :: \mathit{Bool}\} \\
&\mathit{montyHall} :: \mathit{Dist}\ \mathit{Decision} \\
&\mathit{montyHall} = \textbf{do} \\
&\quad \mathit{car} \qquad\quad \leftarrow \mathit{uniform}\ [1\,..\,3] \\
&\quad \mathit{choice}_1 \quad \leftarrow \mathit{uniform}\ [1\,..\,3] \\
&\quad \textbf{let}\ \mathit{left} \quad\ = [\mathit{door} \mid \mathit{door} \leftarrow [1\,..\,3], \mathit{door} \not\equiv \mathit{choice}_1] \\
&\quad \textbf{let}\ \mathit{open} \quad\ = \mathit{head}\ [\mathit{door} \mid \mathit{door} \leftarrow \mathit{left}, \mathit{door} \not\equiv \mathit{car}] \\
&\quad \textbf{let}\ \mathit{choice}_2 = \mathit{head}\ [\mathit{door} \mid \mathit{door} \leftarrow \mathit{left}, \mathit{door} \not\equiv \mathit{open}]
\end{aligned}
$$

Alternative Interpreters ii

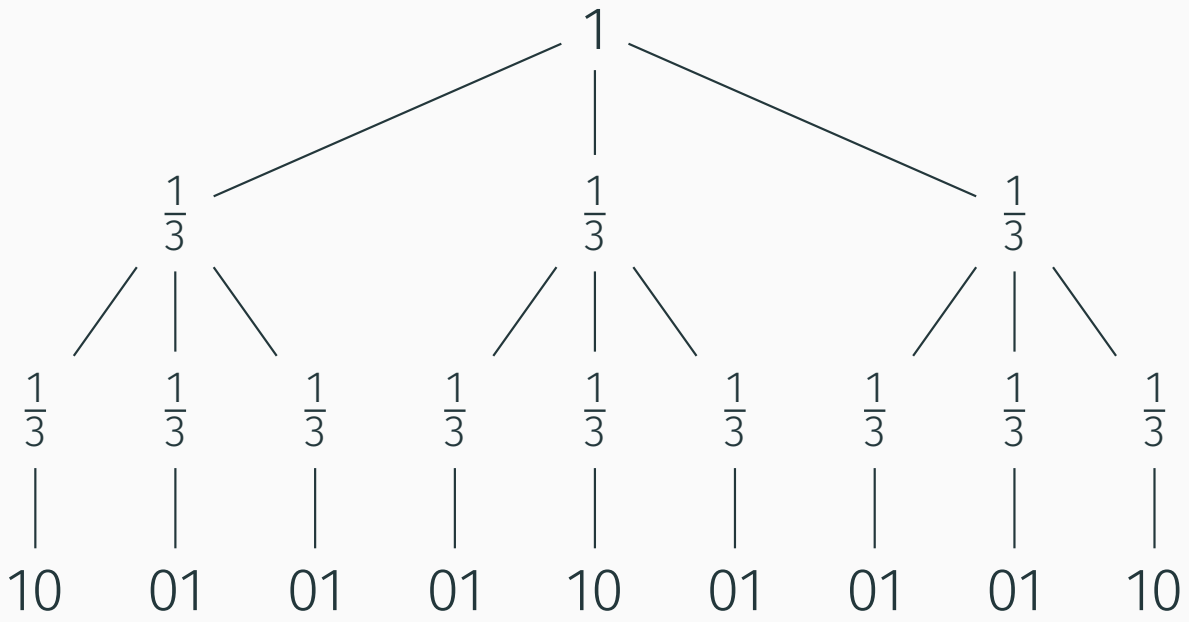
$$\begin{aligned} \mathit{return}\ (\mathit{Decision}\ \{\mathit{stick} &= \mathit{car} \equiv \mathit{choice}_1 \\ , \mathit{switch} &= \mathit{car} \equiv \mathit{choice}_2\}) \end{aligned}$$

While we can interpret it in the normal way to solve the problem:

$$\begin{aligned} \mathit{probOf}\ \mathit{stick}\ \ \mathit{montyHall} &\equiv \tfrac{1}{3} \\ \mathit{probOf}\ \mathit{switch}\ \mathit{montyHall} &\equiv \tfrac{2}{3} \end{aligned}$$

Alternative Interpreters iii

We could alternatively draw a diagram of the process:



Theoretical Foundations

Stochastic Lambda Calculus

It is possible[3] to give measure-theoretic meanings to the operations described above.

$$\mathcal{M}\left[\!\left[\mathit{return}\ x\right]\!\right](A) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$\mathcal{M}\left[\!\left[d \gg\!= k\right]\!\right](A) = \int_X \mathcal{M}\left[\!\left[k(x)\right]\!\right](A) d\mathcal{M}\left[\!\left[d\right]\!\right](x) \tag{2}$$

[3] Norman Ramsey and Avi Pfeffer. "Stochastic Lambda Calculus and Monads of Probability Distributions". In: *29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Vol. 37. ACM, 2002, pp. 154–165. URL: `http://www.cs.tufts.edu/~nr/cs257/archive/norman-ramsey/pmonad.pdf` (visited on 09/29/2016).

The Giry Monad

Meas is the category of measurable spaces, where the morphisms are measurable maps. There is a functor **P** on **Meas**, where for some measurable space x, $\mathbf{P}(x)$ is the set of probability measures on x[4]. The monad for **P** can be defined with two natural transformations

$$\eta : A \to \mathbf{P}(A) \tag{3}$$

$$\mu : \mathbf{P}^2(A) \to \mathbf{P}(A) \tag{4}$$

Their definitions are from above:

$$\eta = \mathit{return} \tag{5}$$

$$\mu(P) = P \gg\!= \mathit{id} \tag{6}$$

[4] Michèle Giry. "A Categorical Approach to Probability Theory". In: *Categorical Aspects of Topology and Analysis*. Ed. by A. Dold, B. Eckmann, and B. Banaschewski. Vol. 915. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 68–85. ISBN: 978-3-540-11211-2 978-3-540-39041-1. DOI:

Implementation

The implementation of the Giry monad is quite direct:

$$\textbf{newtype}\ \mathit{Measure}\ a = \mathit{Measure}\ ((a \to \mathbb{R}) \to \mathbb{R})$$

Other Applications

Differential Privacy

It has been shown[5] that the semantics of the probability monad suitable encapsulate *differential privacy*.

[5] Jason Reed and Benjamin C. Pierce. "Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy". In: *ACM Sigplan Notices*. Vol. 45. ACM, 2010, pp. 157–168. URL: `http://dl.acm.org/citation.cfm?id=1863568` (visited on 03/01/2017).

PINQ

LINQ[6] is an API which provides a monadic syntax for performing queries (sql, etc.)

PINQ[7] extends this to provide *differentially private* queries.

---

[6] Don Box and Anders Hejlsberg. *LINQ: .NET Language Integrated Query*. en. Feb. 2007. URL: https://msdn.microsoft.com/en-us/library/bb308959.aspx (visited on 07/09/2018).

[7] Frank McSherry. "Privacy Integrated Queries". In: *Communications of the ACM* (Sept. 2010). URL: https://www.microsoft.com/en-us/research/publication/privacy-integrated-queries-2/.

Conclusion