Probabilistic Functional Programming

Donnacha Oisín Kidney

July 11, 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

```python
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

```python
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

```
p_1 = expect(
    lambda children: all(child.gender == 'girl'
                         for child in children),
    mr_jones)
p_2 = expect(
    lambda children: all(child.gender == 'boy'
                         for child in children),
    mr_smith)
```

$$\text{p\_1} \approxeq \frac{1}{2}$$
$$\text{p\_2} \approxeq \frac{1}{3}$$

Monadic Modeling

A DSL

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

A DSL

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

Three questions for this DSL:

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?

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?

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]:

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

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/S0956796805005721`. URL: `http://web.engr.oregonstate.edu/~erwig/papers/abstracts.html%5C#JFP06a` (visited on 09/29/2016).

We could (for example) encode a die as:

$$\mathit{die} :: \mathit{Dist\ Integer}$$
$$\mathit{die} = \mathit{Dist}\ [(1, \tfrac{1}{6}), (2, \tfrac{1}{6}), (3, \tfrac{1}{6}), (4, \tfrac{1}{6}), (5, \tfrac{1}{6}), (6, \tfrac{1}{6})]$$

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]
```

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)

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

```python
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`

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

```python
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 i

Assignment expressions can be translated into lambda expressions:

$$\begin{aligned} & \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \\ \equiv & \\ & (\lambda x.e_2)\ e_1 \end{aligned}$$

In the context of a probabilistic language, e_1 and e_1 are distributions. So what we need to define is application: this is encapsulated by the “monadic bind”:

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

Assignment ii

For a distribution, what's happening inside the λ is e_1 given x. Therefore, the resulting probability is the product of the outer and inner probabilities.

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

Assertion

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

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

Return

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

$$
\begin{aligned}
&\mathit{return} :: a \rightarrow \mathit{Dist}\ a \\
&\mathit{return}\ x = \mathit{Dist}\ [(x, 1)]
\end{aligned}
$$

Putting it all Together

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

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

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

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

Alternative Interpreters

Once the semantics are described, different interpreters are easy to swap in.

Monty Hall i

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

Monty Hall ii

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

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

We could alternatively draw a diagram of the process.

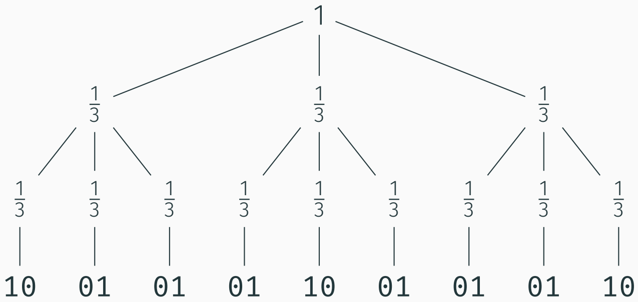

Figure 1: AST from Monty Hall problem. `1` is a win, `0` is a loss. The first column is what happens on a stick, the second is what happens on a loss.

Theoretical Foundations

Stochastic Lambda Calculus

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

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

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

[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).

Giry[4] gave a categorical interpretation of probability theory.

---

[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: `10.1007/BFb0092872`. URL: `http://link.springer.com/10.1007/BFb0092872` (visited on 03/03/2017).

Categories, Quickly

Categories, Quickly

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$$

Categories, Quickly

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$$

Objects $\mathrm{Ob}(\mathsf{C}) = \{X, Y, Z\}$

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "g\circ f"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$$

Objects $\mathrm{Ob}(\mathsf{C}) = \{X, Y, Z\}$

Arrows $\hom_{\mathsf{C}}(X, Y) = X \to Y$

$$
\begin{tikzcd}
X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\
& Z
\end{tikzcd}
$$

Objects $\mathrm{Ob}(\mathsf{C}) = \{X, Y, Z\}$

Arrows $\hom_{\mathsf{C}}(X, Y) = X \to Y$

Composition $\circ$

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$$

Objects $\mathrm{Ob}(\mathsf{C}) = \{X, Y, Z\}$

Arrows $\mathrm{hom}_{\mathsf{C}}(X, Y) = X \to Y$

Composition $\circ$

Arrows form a monoid under composition

$$\begin{tikzcd} W \arrow[r, "f"] \arrow[dr, "g \circ f"'] & X \arrow[d, "g"] \arrow[dr, "h \circ g"] & \\ & Y \arrow[r, "h"] & Z \end{tikzcd}$$

$$(h \circ g) \circ f = h \circ (g \circ f) \tag{3}$$

$$\begin{tikzcd} A \arrow[loop right, "id_A"] \end{tikzcd}$$

$$\forall A. A \in \mathrm{Ob}(\mathsf{C})\ \exists\ id_A : \mathrm{hom}_{\mathsf{C}}(A, A) \tag{4}$$

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$$

Objects $\mathrm{Ob}(\mathsf{C}) = \{X, Y, Z\}$

Arrows $\mathrm{hom}_{\mathsf{C}}(X, Y) = X \to Y$

Composition $\circ$

Arrows form a monoid under composition

$$\begin{tikzcd} W \arrow[r, "f"] \arrow[dr, "g \circ f"'] & X \arrow[d, "g"] \arrow[dr, "h \circ g"] & \\ & Y \arrow[r, "h"] & Z \end{tikzcd}$$

$$(h \circ g) \circ f = h \circ (g \circ f) \tag{3}$$

$$\begin{tikzcd} A \arrow[loop right, "id_A"] \end{tikzcd}$$

$$\forall A. A \in \mathrm{Ob}(\mathsf{C})\ \exists\ id_A : \mathrm{hom}_{\mathsf{C}}(A, A) \tag{4}$$

Example
Set is the category of sets, where objects are sets, and arrows are functions.

Functors

The category of (small) categories, **Cat**, has morphisms called Functors.

Functors

The category of (small) categories, **Cat**, has morphisms called Functors.

These can be thought of as ways to “embed” one category into another.

Functors

The category of (small) categories, **Cat**, has morphisms called Functors.

These can be thought of as ways to "embed" one category into another.

$$
\begin{array}{ccc}
\mathsf{F}X & \xrightarrow{\mathsf{F}f} & \mathsf{F}Y \\
\uparrow & & \uparrow \\
X & \xrightarrow{f} & Y
\end{array}
$$

Functors which embed categories into themselves are called Endofunctors.

Monads

In the category of Endofunctors, **Endo**, a Monad is a triple of:

1. An Endofunctor m,
2. A natural transformation:
$$\eta : A \to m(A) \tag{5}$$
This is an operation which embeds an object.
3. Another natural transformation:
$$\mu : m^2(A) \to m(A) \tag{6}$$
This collapses two layers of the functor.

The Category of Measurable Spaces

Meas is the category of measurable spaces.

The Category of Measurable Spaces

Meas is the category of measurable spaces.

The arrows ($\mathsf{hom}_{\mathsf{Meas}}$) are measurable maps.

The Category of Measurable Spaces

Meas is the category of measurable spaces.

The arrows ($\mathbf{hom}_{\mathbf{Meas}}$) are measurable maps.

The objects are measurable spaces.

The Category of Measurable Spaces

Meas is the category of measurable spaces.

The arrows ($\mathbf{hom}_{\mathbf{Meas}}$) are measurable maps.

The objects are measurable spaces.

We can construct a functor ($\mathcal{P}$), which, for any given measurable space $\mathcal{M}$, is the space of all possible measures on it.

The Category of Measurable Spaces

Meas is the category of measurable spaces.

The arrows ($\mathbf{hom}_{\mathbf{Meas}}$) are measurable maps.

The objects are measurable spaces.

We can construct a functor ($\mathcal{P}$), which, for any given measurable space $\mathcal{M}$, is the space of all possible measures on it.

$\mathcal{P}(\mathcal{M})$ is itself a measurable space: measuring is integrating over some variable a in $\mathcal{M}$.

The Category of Measurable Spaces

Meas is the category of measurable spaces.

The arrows ($\mathbf{hom}_{\mathbf{Meas}}$) are measurable maps.

The objects are measurable spaces.

We can construct a functor ($\mathcal{P}$), which, for any given measurable space $\mathcal{M}$, is the space of all possible measures on it.

$\mathcal{P}(\mathcal{M})$ is itself a measurable space: measuring is integrating over some variable a in $\mathcal{M}$.

In code (we restrict to measurable functions):

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

We now get η and μ:

$$
\begin{aligned}
&\mathit{integrate} :: \mathit{Measure}\ a \to (a \to \mathbb{R}) \to \mathbb{R} \\
&\mathit{integrate}\ (\mathit{Measure}\ m)\ f = m\ f \\
&\mathit{return} :: a \to \mathit{Measure}\ a \\
&\mathit{return}\ x = \mathit{Measure}\ (\lambda \mathit{measure} \to \mathit{measure}\ x) \\
&(\ggg\!=) :: \mathit{Measure}\ a \to (a \to \mathit{Measure}\ b) \to \mathit{Measure}\ b \\
&\mathit{xs} \ggg\!= f = \mathit{Measure}\ (\lambda \mathit{measure} \to \mathit{integrate}\ \mathit{xs} \\
&\qquad\qquad\qquad\qquad\qquad (\lambda x \to \mathit{integrate}\ (f\ x) \\
&\qquad\qquad\qquad\qquad\qquad (\lambda y \to \mathit{measure}\ y)))
\end{aligned}
$$

Other Applications

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).

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