COMP 150PP Class Exercise: The Probability Monad

September 28, 2016

Questions are based on *Stochastic Lambda Calculus and Monads of Probability Distributions* by Norman Ramsey and Avi Pfeffer.

Design review

If `P` is a probability monad, then the Haskell code presented in this paper offers these functions:

```
return  :: a -> P a
(>>=)   :: P a -> (a -> P b) -> P b
                        -- pronounced "bind"
choose  :: Probability -> P a -> P a -> P a
support :: P a -> [a]
expectation:: (a -> Double) -> P a -> Double
sample  :: RandomGen g => P a -> g -> (a, g)
```

A design alternative: Bernoulli

Many probabilistic languages take as primitive the *Bernoulli distribution*, named after Jacob Bernoulli, who discovered the law of large numbers. The Bernoulli distribution is equivalent to a biased coin; `bernoulli` p is a distribution that is `True` with probability p and `False` with probability $1 - p$:

```
bernoulli :: Probability -> P Bool
```

Answer these questions:

1. Give algebraic laws for `support` and `expectation` when applied to a Bernoulli distribution.
2. If we remove `choose` from the set of primitive functions and replace it with `bernoulli`, have we lost any expressive power? Justify your answer in one of two ways:
   - If we have not lost any expressive power, then you must be able to show how to implement `choose` using `bernoulli`.
   - If we have lost some expressive power, then you must be able to construct, using the functions in the paper, a distribution that can no longer be constructed once `choose` is replaced by `bernoulli`.
3. If we add `bernoulli` to the functions given in the paper, have we gained any expressive power? Justify your answer in one of two ways:
   - If we have not gained any expressive power, then you must be able to show how to implement `bernoulli` using `choose`.
   - If we have gained some expressive power, then you must be able to exhibit a distribution constructed with `bernoulli` that cannot also be constructed using the functions in the paper.
4. If you wish, give algebraic laws for `sample` when applied to a Bernoulli distribution.

Comparing designs

Our own designs for finite probability distributions `F a` offer these functions:

```
unit     :: a -> F a
uniform  :: [a] -> F a
weighted :: [(Probability, a)] -> F a
pthen    :: F a -> (a -> F b) -> F (a, b)
pmap     :: (a -> b) -> (F a -> F b)
join2 :: F a -> F b -> (F a -> b -> F c) -> F c
```

It is not too difficult to prove that the set of functions from the paper is minimal. It is very easy to show that the set of functions from our design is *not* minimal. But now it's time to compare the designs more closely:

5. What functions are *identical* in both designs (except for possibly having different names)?
6. What functions in the paper's design are missing from our design but can be simulated using our functions? When simulations exist, *please show them*.
7. What functions in our design are missing from the paper's design but can be simulated using the paper's functions? Just sketch one or two simulations.
8. What functions in the paper's design, if any, *can't* be simulated using our functions?

9. What functions in our design, if any, *can't* be simulated using the paper's functions?

10. We haven't yet solved all the dice problems. And the paper's design can't. What crucial piece or pieces are missing from the paper's design, or from our design, that prevent either design from being able to do all the dice problems?

Deeper analysis of the paper

11. Explain Figure 2. (Think of Figure 2 as the specification for an interpreter; in particular, think of $\mathcal{P}$ as a recursive function that might be called `eval`. You might begin by asking about that function's *type*.)

12. In the first paragraph of Section 8 on page 163, what's being claimed? Are you convinced?

13. Section 5.1 talks about the performance of expectation queries over product spaces. You have intimate experience with product spaces from the dice world. Can you think of an expectation query that would take $O(|X| \times |Y|)$ time in the expectation monad but could be computed in $O(|X| + |Y|)$ time using some other technique?

Questions to take home

14. Unless you happen to know the related work already, sections about related work are usually boring. But there's a different reading of Section 7 than simply who did what. Looking through Section 7, what different language-design choices can you identify? What choices appeal to you? What choices do you find distasteful?

15. Section 3.2 mentions sampling functions. For a distribution with finite support, what would a sampling function look like?

16. What role should abstract integration play in a probabilistic programming language?

17. What, if any, is the *computational* consequence of equation (2) at the bottom of page 156?

18. At the end of Section 4 on page 158, the paper says that in the paper's calculus, it is not safe to duplicate a redex. Why not? Is there a simple explanation in terms of what kind of language design is being described?