Neel Pai

Professor Mattox Beckman

UIUC CS 421 -- Programming Languages

07/28/2024

CS 421 Final Paper

One of the key classes of problems that computers are designed to solve is that of probabilistic modeling. Often requiring significant computational power, simulating complex distributions is often difficult for humans to do innately; yet, a computer is tailor made to quickly and efficiently answer these questions. However, basic functional programming language paradigms are insufficient to express probabilistic concepts, as they rely on deterministic outcomes with no structured way to simulate randomness. Erwig and Kollmansberger's probabilistic functional programming (PFP) library offers a solution to this problem by building a set of probabilistic types and functions directly into Haskell, allowing for more convenient representations of randomness. This paper explores the major structures of Haskell's PFP library, demonstrating how it facilitates probabilistic programming functionality, and how this set of types and functions can be used to answer relevant classes of probability questions.

The PFP library's approach to implement probabilistic modeling functionality into Haskell is to encapsulate probabilities and distributions within custom data types; this enables the code to maintain referential transparency (the idea that a function f applied to a value x always yields one and the same value y = f(x)), a core tenet of functional programming. Distributions are typed as collections of outcomes paired against their respective likelihood, with the sum of probabilities in a distribution equalling one. We also include Spread as a type which allows the formation of popular distributions (e.g. unif, norm, etc.) This manifests as the following code:

```
newtype Probability = P Float
newtype Dist a = D { unD :: [(a, Probability)] }
type Spread a = [a] -> Dist a
```

The rest of the PFP library contains an extensive list of functions, types, and problems, but these are the core types to understand.Using this extensive class of probability functions from the PFP library, we are able to simulate a variety of probability questions in Haskell (included in this repository). In my project proposal, I promised to do the following things: 1) Write this paper as a reflection of the work. 2) Update the probability.hs file to add additional functionality. 3) Create two new example hs files (not included in the PFP library) that will utilize the PFP types and functions. 4) Create a simulation file which will contain sample runs of all probabilistic examples within the library (including my two new examples). The full list of my changes is included below, proving that I have met each of these 4 criteria:

- Updated Libraries and packages so all imports functioned correctly. (point 2)
- Built YAML file and directories to correctly set up a Main.hs file to run and test functions. (point 4)
- Within the probability.hs file:
  - Added some up-front Monad functions (e.g. applicative, alternative) to help fix flow of Monad logic (point 2)
  - Added new distributions (gamma, log-gamma, beta) to the set of default distributions (point 2)
  - Built a few print functions to better output distributions. (point 2)
- Within other files:
  - Generalized Monty Hall Problem to accept an input n to determine number of doors (rather than defaulting to 3) (point 2)
  - Built dicesums functions within dice to create a distribution according to dice sums, rather than individual numbers on dice (point 2)
  - Built a new file for playing blackjack (point 3)
  - Built a new file, Coupons.hs, to simulate the coupon collector's problem. (point 3)
- Within the Main.hs file:

    - Built a menu of tests for each of the given simulations (point 4)
    - Built a fully functioning blackjack game (point 4)
    - Built a full test of the Coupon Collector Problem (point 4)

I am confident that my changes were accretive and done well for the most part -- I think there was even more opportunity for me to leverage the wealth of code and functionality available within the PFP library. It felt as though I had barely scratched the surface of types and operations that the full library had to offer. The PFP library offers significant benefits in terms of efficiency, succinctness, and composability, making it a valuable tool for both teaching and applying probabilistic modeling in functional programming contexts. I hope to get more familiar with this library in the future.

Tests and code are available for viewing in the rest of this repository.

Works Cited

ERWIG, M., & KOLLMANSBERGER, S. (2005). Functional pearls: Probabilistic functional programming in Haskell. *Journal of Functional Programming*, *16*(1), 21–34. https://doi.org/10.1017/s0956796805005721