CS251-Languages Final Project Proposal

MaryBeth Kery

May 2015

1 Haskell

Haskell is a no-side-effects, lazy evaluation, purely functional language with a strong static type system (with sophisticated polymorphism) and type inference. As a final project, learning Haskell seems like a fittingly extremist point of view on many topics we've covered in CS251. I'll use ML as comparison. Haskell and ML appear to start from a common base of functional programming ideas, static type-system, pattern matching, ect. so it will be interesting to see at what point the languages start making different choices and why. Comparison to imperative programming also seems to be brought up a lot in discussion of monads, so where it makes sense I may also compare to Python.

1.1 Language Features

- **Lazy evaluation:** Haskell is lazy by default, which enables things like infinite lists and complex operations on first class functions. Since currying and partial-application seem extremely common, I'll re-examine those since it wasn't something I fully understood during CS251. I'll describe what behaviors lazy evaluation provides, and am very curious to look at what makes it a valuable language choice.
- **purity/side-effects:** Haskell is a "pure" functional language that doesn't allow side-effects and only very restricted mutation. Yet side-effects are clearly needed for a program to provide any bit of use for us humans. To interface with side-effects in a "quarantine the impure" kind of way, Haskell has a complex system of monads. I want to understand this logic and also why purity is valuable enough to justify the convoluted workarounds.
- **monads:** I've read enough to know what behavior to expect from monads, and can probably use them, but there seems to be some magical logic concepts borrowing from math behind monads worth exploring.
- **type classes**
- **Overall Design choices:** While Haskell is beautiful from a PL perspective, there's a steep learning curve even with some experience in functional

programming and the annoyance with side-effects makes Haskell possibly less usable to programmers out in the real world. [1] Are there performance gains? What kind of applications go well with Haskell?

1.2 Programs to build

After building some toy programs to test-out/demonstrate the basic ideas of Haskell, I want to make something that has a fair amount of back-and-forth with side-effects. For now my thoughts are a text-based adventure game, since you would need simple IO, some kind of data-structures and state. The challenge will be using functions in an interesting way.

References

[1] Haskell documentation. https://www.haskell.org/documentation.

[2] Learn you a haskell. http://learnyouahaskell.com/chapters.

[3] CAR Hoareetal. Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in haskell. *Engineering theories of software construction*, 180:47, 2001.

[4] Philip Wadler. Comprehending monads. *Mathematical structures in computer science*, 2(04):461–493, 1992.

[1] xkcd Haskell: https://xkcd.com/1312/