Kevin Liao

CMSCB380 Functional Programming

Professor Richard Eisenberg, Bryn Mawr College

I Am Now a Convert: A Reflection on a Semester of Haskell

What I'm doing this summer says I'm a convert. I'm spending this summer working on Integrity Engineering at Facebook, which mainly means fighting spam with Haskell. When I'm answering questions for my intro students in CS104 about Python, part of me wants to throw something at the wall. (I do this ruminating quietly, and maintain my cheerful exterior as not to put them off programming) Most of their problems are related to not being able to figure out when there is or isn't IO occurring, and most of them are having difficulties differentiating between mutation of state inside of objects, and functional style code that returns a value. Part of me wishes that in intro, we taught a stricter style of programming that would avoid all of these problems.

In Python, we talk about doing things in the "One True Way". In other words, the syntax/semantics of Python should work themselves out in such a way that there is a clear path to take to solve each problem. Pragmatically, I feel that this isn't the case. In Haskell, the isolation of all IO operations and the clear separation of imperative from functional means that it really feels like we are forced to reason about things in separate paradigms. That to me is a powerful separation that even other functional programming languages I've learned on the side (Elixir/Erlang) do not have. I'm thinking about writing code in Python in these terms now. I try to isolate my IO operations and make as much of my work pure functional and isolated from IO/other side effects as possible.

Haskell's type system works well when parsing very well-defined data structures. Working with TagSoup, for example, was a joy. Once I got around the documentation, the line of

reasoning became very clear. We think about HTML and the DOM as a clear tree hierarchy. That means parsing well-formed HTML with Haskell was easy through pattern matching.

Haskell's downside is obviously going to be the learning curve. With that comes the documentation. While working on *bb*, the utility I made for my final project, I was constantly hunting through the documentation for TagSoup and the time library. Too much of the time, Haskellers throw types into documentation and let us sort it out ourselves. Often, that can make things confusing for users as we decipher meaning behind a confusing array of type-level glyphs.

While doing things the 'One True Way' means that best practices are followed, it can sometimes introduce unnecessary engineering overhead. For example, when getting the current time from the machine clock to compare with blue bus times, I needed to get the time in UTC, convert it to about 3 different formats, before getting the final information that I needed in the time zone I needed. These are things I probably could've gotten around had I used a programming language with a less strict type system with less of a focus on excessive best practices. Then again, working with time in general is a pain, and introducing as much type safety as possible to avoid bugs makes sense.

I'd like to spend time working on Haskell documentation and tutorials. Not only would it increase my understanding of the programming language, but I feel like there is a genuine lack of plain English tutorials out there. I think there is a tendency for Haskellers (and FP programmers in general) to sell Haskell on the idea that "it's just Math". Unfortunately for a lot of people, that's just not a very good sell. Learn You a Haskell for Great Good might be outdated, but it was incredibly helpful to me throughout the semester because it really explained things in a way that I could understand. Maybe that's what Haskell needs: More plain English tutorials. I mentioned earlier that I'd like to see a more pure-functional style of programming taught at the

intro-level. I believe it's possible. I think it might even be easier, constraining students to a very strict kind of IO and a very strict paradigm. At the same time, the lack of resources makes this a very risky proposition.

All in all, the way that I see programming has changed. I tried building a GTK app in Rust over the weekend as a fun challenge, but writing imperative code again felt like a chore. Every line I was writing felt impure and filled with side effects. Things were mutating and it felt wrong to me. Haskell has turned my world upside down. Thank goodness I found a job that pays writing this stuff, because I don't know what I'd do otherwise.