

Class 25: Haskell (3)

Held: Monday, 4 April 2011

Summary: We conclude our consideration of Haskell.

Related Pages:

- EBoard.
- Lab: Haskell (3).
- Reading: Tate 8.4-8.5.

Notes:

- I hope you had a good spring break. I once again succumbed to the planning falacy.
- I will distribute the reading for Wednesday later today. No Piazzza questions necessary. (If I had time, I would have distributed Jean Sammet's 1972 history of programming languages, available at <http://portal.acm.org/citation.cfm?doid=361454.361485>).
- President Kington is talking to the faculty today about "Choosing our Future". I'll read to you from some key points.
- Today will be a lecture-style class.
- I will reserve time at the start of class for questions on mid-semester examination.
- I will reserve time at the start of class for discussion of the presentations.

Overview:

- Haskell's type system.
- Monads.

Types in Haskell

- As you've seen, Haskell has a rich and robust type system.
- Haskell also takes advantage of an interface-style approach: You can indicate that a type implements an interface using the `instance` syntax.
- We'll look at a simple tree example.

Monads

- So, how do you do I/O and other side-effecting operations in a pure language?
- In essence, you need to pass the state around at all times. (That is, you make the implicit explicit.) For example, a random function, rather than returning just a random number, must also return the updated random number generator.
- Monads provide a uniform way of dealing with these issues.

- You are required to
 - indicate how to apply functions
 - make your functions return values in the same domain
 - indicate how to extract a value from the environment at the end of the monad.
- In some sense, Monads are just custom and syntax for dealing with these “imperative” issues.
- However, Monads also have a deep mathematical structure
 - I’ve tried to read the papers on that structure, and failed.