

ONE YEAR IN

A BEGINNER'S TAKE ON THE DIFFICULTIES OF LEARNING
HASKELL

Jake Keuhlen

<https://github.com/jkeuhlen/talks>

My Background

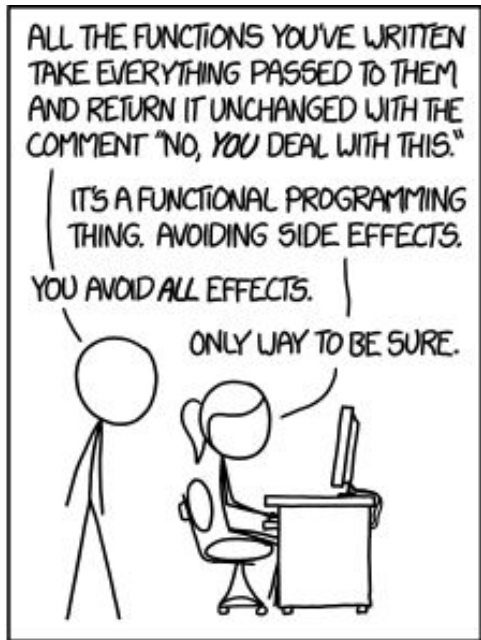
- BS in Engineering Physics
- Previous experiences heavy in Perl, C, and Ruby with mix of C++, Java, Python, etc.
- Started with Haskell in April 2016 at Holland & Hart
 - Haskell powers our next-generation legal AI that works to automate paralegal and attorney work at H&H
- Started teaching new hire in April 2017

HOLLAND & HART^{LLP}



How to Learn Haskell

- With Practice:
 - Jump into the deep end
 - Build your own project
- With Teaching:
 - Look into haskell forums
 - Help newcomers with questions
- With Reading:
 - Tutorials
 - Comics
 - Blog posts
 - Stackoverflow
 - Papers



My Journey

- Initial Tech Stack
- First Projects
- Road Blocks
- V2
- New Hire



Initial Tech Stack

- “Implicit Semantic Arrows” - Kleisli Arrows with special context wrappers
- Template Haskell
- QuasiQuoters
- Unique DSL's
- Lenses/Prisms
- Fix/Recursion Schemes
- Happstack, Conduit, Parsec, Aeson
- Monad Transformers



Starting Out

- Worked on expanding existing applications
 - Forced me to understand complex code
- Built two new projects from scratch
 - API Translation Layer for archaic DMS
 - Chat bot for natural language search
 - Often got stuck trying to find the right way to do things



Roadblocks - The “Haskell” Way vs TIMTOWTDI

- Which way is “correct”, idiomatic Haskell?

```
-- | doMath takes a number, adds three to it, then multiplies by  
three, then performs integer division by seventeen
```

```
doMath :: Int -> Int
```

```
doMath x = do
```

```
  let a = x + 3
```

```
  let b = a*3
```

```
  let c = b `div` 17
```

```
  c
```

```
doMathV2 :: Int -> Int
```

```
doMathV2 x = let a = x + 3
```

```
              b = a * 3
```

```
              in b `div` 17
```

```
doMathV3 :: Int -> Int
```

```
doMathV3 x = ((x + 3) * 3) `div` 17
```

```
.
```

```
doMathV4 :: Int -> Int
```

```
doMathV4 = (`div` 17) . (*3) . (+ 3)
```

Roadblocks - Recursion

- Switch from thinking in loops to thinking in folds or maps
 - No more off-by-one errors!
- Fix, Cata, \perp
 - Catamorphisms are confusing, and entirely foreign concept
 - Especially confusing for newcomers due to implicit recursion



Roadblocks - Language Extensions

- Compile errors don't always help you figure out when you're speaking the wrong dialect
 - Arrow Syntax
 - error: parse error on input `->' for: `proc x -> do`
 - Scoped Type Variables
 - Could not deduce (Bounded a1) arising from a use of 'minBound' from the context:
Bounded a bound by the type signature for: `myTest :: Bounded a => a`



V2

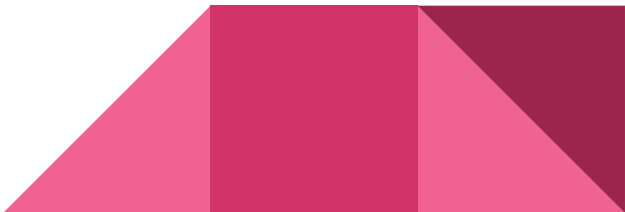
- Switched to Free Arrows

- Centralized algebras for effects
- Individual “compilers” that allow us to bake in our effects

- Free Arrows in GHCi

->>>(((Pure)+++((Tag(Tag((Pure)>>>(((Pure)+++(((Pure)**(((Effect)>>>(Pure))>>>(((Pure)+++(Pure))>>>(Pure))))>>>(((Pure)+++((Pure))>>>(Pure))))>>>(((Pure)+++((Pure))>>>(Pure))))>>>(Pure)))>>>(((Pure)+++((Pure))>>>(((Pure)+++(((Pure)>>>(((Pure)+++((Pure)>>>(((Pure)+++(((Pure)>>>(((Pure)+++(((Effect)>>>

- Type Level Programming



New Hire

- Uncovered a significant flaw in the system
 - Suddenly understood category theory while discussing an instance for Applicative Monoid
- Within two weeks, making significant contributions via work in a centralized area (parsec)
- Hardest part for him is learning Type Programming (syncing effects and errors, picking the right compilers)
- Now giving him his own project to build from scratch



Questions?

