Haskell – Overview

David Grisham

31 October 2017

Properties

Outline

- Polymorphically statically typed
- Lazy
- Purely functional

Type System

- Polymorphic
- Static typing
- Strong typing

Type System

Polymorphism

1. ***Parametric*** polymorphism
   - *Unconstrained type variables*
   - e.g. `id :: a -> a`
2. ***Ad-hoc*** polymorphism
   - *Constrained type variables*
   - e.g. `sort :: Ord a => [a] -> [a]`

Type System

Static typing

- Type-checking happens at *compile time*
- Efficient
  - No runtime type-checks
  - Known memory requirements

Type System

Strong typing

- *No* implicit type conversions. . .
- . . . but *does* have type inference and polymorphism

Lazy

- Evaluation only when needed

```
> x = 1 `div` 0
> print x
*** Exception: divide by zero
```

Lazy

Advantages

- Save computation time
- More modular and expressive

Disadvantages

- Memory usage less predictable
- Might slow down execution

Purely Functional

- *Pure*: prohibits side effects
- Functions operating on *immutable data*
- Referential transparency
  - Lazy evaluation

Ecosystem

Tools

- GHC
- Stack

GHC

- Standard compiler for Haskell
- Simon Peyton Jones, Simon Marlow

Stack

- Development environment for Haskell
- Package management, testing, . . .

History

From the man himself...

Click

My Haskell Projects

Elements of Computing Systems

- **Compiler** for Jack (Java-like language)
  - Jack $\rightarrow$ VM
  - VM $\rightarrow$ assembly language
- Only group using Haskell
  - Python, Java, Ruby, ...

Compiler: Benefits of Haskell

- `Parsec` ('Parser combinators')
  - Build complex parsers from simple ones
  - Practical intro to more esoteric Haskell
- Relatively minimal Haskell knowledge
  - Result still fairly robust

Imperative Language Interpreter

- Direct execution (instead of compiling)
  - Expressiveness

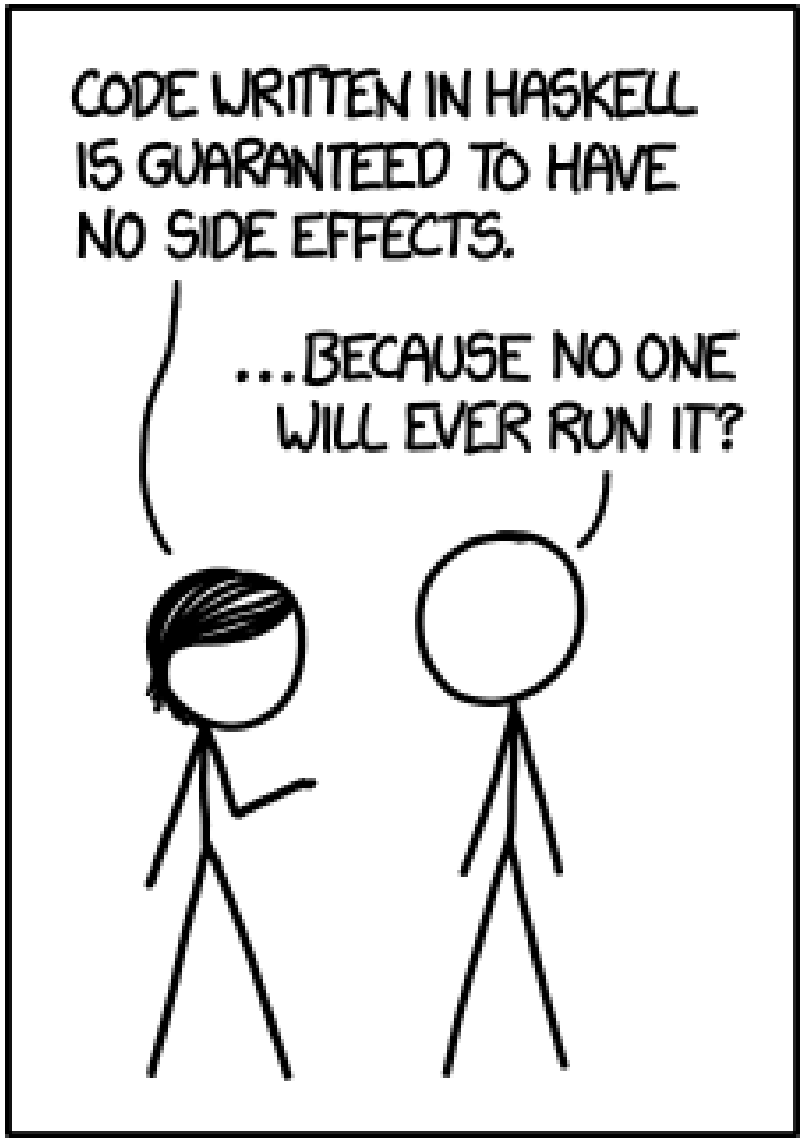
```
exec (IfStmt condition stmt1 stmt2) env
| condition = exec stmt1 env
| otherwise = exec stmt2 env
```

Other Projects

- **Decoy routing**
  - Game-theoretic simulation
- **Movie suggestion script**
  - Filter movies based on genre/etc.
  - Spit out random movie from result
- **Todo-list manager**
  - Add task, schedule task, ...
  - `taskwarrior`

Why care about Haskell?

XKCD


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

Motivation

- Expressive
- Fewer runtime bugs
- Easier to debug
- Easier to maintain
- Code reuse

Worst practices should be difficult

- Sensible defaults in Haskell
  - `Maybe`/`Nothing` over `NULL`/`None`
  - Immutability
  - Minimal `IO`

Programming Paradigm

- Informs how you think about coding
- *Very* useful to broaden
- Better code in other languages

Coming Up With Haskell Projects

Challenge isn't "what can I do in Haskell"

Challenge is "how can I do X in Haskell"

Further Reading

Links (articles)

- Beating the Averages
  - Competitive advantage in programming language choice
- Worst practice should be hard
  - Long-term language productivity

Links (paper and talk)

- Von Neumann vs. Functional Languages
  - First 10 pages, more if you want
- Functional Programming Design Patterns
  - Straightforward explanations of functional advantages

Links (learning resources)

- Real World Haskell
  - Brian O'Sullvian, et al.
- Stanford 240h: Functional Systems in Haskell
  - David Mazieres and Brian O'Sullvian
  - All resources available (except lecture vids)
- /r/haskell
  - Consistently worthwhile content here