Introduction to Haskell - Overview

Pranesh Srinivasan

June 17th 2010

About Me

- A returning intern at Google Hyderabad.
- Picked up Haskell in early 2008.

About Me

- A returning intern at Google Hyderabad.
- Picked up Haskell in early 2008.
- Stuff I have written in Haskell:

About Me

- A returning intern at Google Hyderabad.
- Picked up Haskell in early 2008.
- Stuff I have written in Haskell:
  - A translator from Simplified Pascal to C.
  - A simple accounting system for an average student's needs.
  - A simple markup language for simplifying creation of Beamer presentations (like this one).
  - An IRC bot.
  - Lots of small utilities.
  - Also my language of choice for Project Euler, spoj, etc..

Why this g2g?

- Siddharth Priya

Why this g2g?

- Siddharth Priya
- I *like* Haskell

How this talk is structured?

- Dive head first into examples.
- I'll explain things as we go along.
- Ask questions anytime.

What is Haskell?

Haskell is a statically typed pure functional language with lazy evaluation.

What is Haskell?

Haskell is a statically typed pure functional language with lazy evaluation.

Sounds too esoteric to be practical?

What is Haskell?

Haskell is a statically typed pure functional language with lazy evaluation.

Sounds too esoteric to be practical?

Quick Example: `mini-grep` - From the practice of programming.

- 32 lines of haskell vs 97 lines of C

What is Haskell?

Haskell is a statically typed pure functional language with lazy evaluation.

Sounds too esoteric to be practical?

Quick Example: `mini-grep` - From the practice of programming.

- 32 lines of haskell vs 97 lines of C
- How about Extending it?

Why Haskell?

- Pure - No Side Effects.
- Lazy - Nothing is evaluated until it has to be.
- Rigorous Type System.
- Compiled and Garbage collected.

Purity

Pure - No Side Effects.

- Easier to reason with - substitution model (SICP)
- Excellent for the compiler
- Parallel programming
- "state was always a bad idea"
- Programs as a function from input to output.

```
let x = 5
```

Purity

Pure - No Side Effects.

- Easier to reason with - substitution model (SICP)
- Excellent for the compiler
- Parallel programming
- "state was always a bad idea"
- Programs as a function from input to output.

```
let x = 5
```

```
let x = 7 (doesn't make sense in Haskell!)
```

Laziness

- Nothing is evaluated until it has to be.
- The best time optimisation is not to do it at all.
- No evaluating ugly constants.
- Encourages the stream style of programming.

```
primes = f [2..] where

generate_primes (x:xs) = x:generate_primes(

[a | a <- xs, a `rem` x /= 0]

)
```

Type System

- *Strong* typing (based on the Hindley Milner Type System).
- Has type inference.
- Supports Algebraic Data types.

Type System

- *Strong* typing (based on the Hindley Milner Type System).
- Has type inference.
- Supports Algebraic Data types.

"I write programs, make them compile, and they just run"

Type System

- *Strong* typing (based on the Hindley Milner Type System).
- Has type inference.
- Supports Algebraic Data types.

> *"I write programs, make them compile, and they just run"*

```
add :: Int -> Int -> Int

add x y = x + y

double x = add x x
```

Why Haskell?

Alan J Perlis:

> *A language that doesn't affect the way you think about programming, is not worth knowing.*

Installing Haskell

- Standard Compiler is GHC - Glorious Glasgow Haskell Compiler.
- Available on Windows, Linux, and Mac.
- A better option is the Haskell Platform.

Or tryhaskell.org

Basic Data Structures

- Int
- Integers
- Char
- String
- Tuples
- Lists

Hello World.

```
putStrLn "Hello, World!"
```

Factorial

What's the type?

Factorial

What's the type?

```
factorial :: Integer -> Integer
```

Factorial

What's the type?

```
factorial :: Integer -> Integer
```

```
let fact n = if n == 0 then 1 else n * fact (n-1)
```

Factorial v2 - I can match!

In GHCI

```
let fact2 0 = 1; fact2 n = n * fact2 (n-1)
```

Factorial v3 - I found the right function.

As the product of integers [1..n]

```
let fact3 n = product [1..n]
```

Factorial v4 - Reduce away.

```
let fact4 n = foldr (*) 1 [1..n]
```

Are these equivalent?

QuickCheck!

- Takes a function from any type `a` to `True/False`
- Runs it for several different values of `a`.
- Checks if True is obtained each time.

Are these equivalent?

QuickCheck!

- Takes a function from any type `a` to `True/False`
- Runs it for several different values of `a`.
- Checks if True is obtained each time.

```
quickCheck (\n -> fact n == fact2 n)
```

Are these equivalent?

QuickCheck!

- Takes a function from any type `a` to `True/False`
- Runs it for several different values of `a`.
- Checks if True is obtained each time.

```
quickCheck (\n -> fact n == fact2 n)
```

FAIL! Stack Overflow.

Is the type exactly what we want it to be?

We want **Positive Integers**.

```
quickCheck ((\n -> fact n == fact2 n)

:: Positive Integer -> Bool)
```

Success!!

Almost.

That's it for now.
More coming up:

- currying.
- mini-grep.
- an unjumble program.
- parallel programming.
- Finding Pi to 50,000 digits quickly.

`pranesh@google.com`

Till next time.

- http://learnyouahaskell.com/
- tryhaskell.org
- Haskell in 10 minutes - http://haskell.org/haskellwiki/Learn_Haskell_in_10_minutes
- The Haskell Prelude.