EECS 776

Functional Programming and Domain Specific Languages

Professor Gill

The University of Kansas

Jan 31st 2020

KU

Homework 1

Due Monday 3rd February

- Find a ghci interpreter
- Try out the following example
  - Write a function to compute the surface area of a sphere
  - Run the function on a small number of inputs
  - **Bring your interaction with ghci (what you typed, what ghci said back)**
- Remember to use the 776 coversheet

Small Haskell Program

```
-- This is a small Haskell program
module Main where

import System.Environment

main :: IO ()
main = do args <- getArgs
          printArgs args

-- printArgs print to stdout the
input list,
-- one line at a time.
printArgs :: [String] -> IO ()
printArgs (arg:args) = do putStrLn
arg

printArgs args
printArgs []          = return ()
```

```
$ ghc --make Main.hs
[1 of 1] Compiling Main (
Main.hs, Main.o )
Linking Main ...
$ ./Main Hello World
Hello
World
```

Types

Types are the distinguishing feature of Haskell-like languages

- What are types?

```
42 :: Int
```

- What is type-checking and type-inference?
  - Type-checking is checking if the types are **self-consistent**
  - Type-inference is checking **without being told** what the types are

Most modern languages have some form of type-checking, some have type-inference

Robin Milner

Robin was an outstanding and well-rounded computer scientist

- Machine-assisted proof construction (LCF)
- **Design of typed programming languages** (ML)

"Well-typed programs don't go wrong."

- Models of concurrent computation (CCS, π-calculus)

He was awarded the Turing Award in 1991

KU

Type systems in modern languages

Java - static typing

```
public int example(int
x,double y) {
   String z = "Hello";
   ...
}
```

Statically typed languages are dependable but rigid

JavaScript - dynamic typing

```
function example(x,y) {
   var z = "Hello";
   ...
}
```

Dynamically typed languages are flexible but unreliable

The type system in Haskell

In Haskell, you can give the types of the values ...

```
sphereArea :: Double -> Double
sphereArea r = 4 * pi * r^2
```

... or let Haskell infer it ...

```
sphereArea r = 4 * pi * r^2
```

The type says "take a **Double**, return a **Double**"
So **r** is a **Double**, and **4 * pi * r^2** is a **Double**

```
Prelude> :l Example.hs
*Main> sphereArea 5
314.1592653589793
```

The type system in Haskell (GHCi)

You can also give the type in GHCi ...

```
Prelude> let sphereArea :: Double -> Double ; sphereArea r
= 4 * pi * r^2
Prelude> :t areaOfSphere
areaOfSphere :: Double -> Double
```

... or let GHCi infer it ...

```
Prelude> let sphereArea r = 4 * pi * r^2
Prelude> :t
????
```

Type inference

Parametric polymorphism is a sweet spot on the typing landscape.

- Static typing,
- with Polymorphic values (give you dynamic-like typing when you need it)

The type inference in Haskell is really powerful.

It is considered good form (and documentation) to write some types, and let Haskell figure the rest out.

Haskell is not guessing the types, it is inferring them.

An inferred type is a high form of truth, and inference is a crowning achievement of centuries of mathematics.

Caveat: In order to be work within this powerful system, many primitives
in Haskell have non-obvious types. There is always a reason why.