PROGRAMMING IN HASKELL

Overview

Chapter 1

- Introduction
- History of Haskell

Chapter 2

- Basic maths, lists, functions, scripts

  - Glasgow Haskell Compiler (ghci)
  - Maths: 34.5 + (3.2 * 4.5)
  - Lists: head [1,2,3]
  - Lists: [3,2,1] ++ [1,2,3]
  - Function application: f x y
  - Haskell .hs scripts
  - Load script :load in GHCi

Chapter 3

- Types, polymorphism, predefined type classes

- Predefined types: Bool, Int, Float, Char, String
- Type notation: [3,4] :: [Int]
- Command :type in GHCi
- Lists and tuples
- Function types; eg Int -> Int -> Int
- Curried functions: eg Int -> (Int -> Int)
- Polymorphism: [a] -> a
- Overloading: Num a => [a] -> a

Chapter 4

- Defining functions

  - Conditional expression: if …. then … else
  - Guarded expressions (using |)
  - Pattern matching (list patterns)
  - Lambda expressions: eg (\x -> x+x)
  - Operators: (+) 2 3; (+2) 3

Chapter 5

- List comprehension and strings

  - List comprehension:

    [sqrt(x^2+y^2) | x<-xs, y<-ys]

  - List comprehension with guards:

    [sqrt(x^2+y^2) | x<-xs, y<-ys, x<y]

  - Strings: String is [Char]:

    “abc” means [‘a’, ’b’, ’c’]

Chapter 6

- Recursive functions

- Recursive functions: eg

```
fac :: Int -> Int
fac 0 = 1
fac n = n * fac (n-1)
```

- Recursion on lists
- Mutual recusion (two functions)
- Computational efficiency (eg tail recursion)

Chapter 7

- Higher-order functions

  - Functions as values: eg v = (\x -> x+x)
  - Functions as arguments: eg map f list
  - List aggregation: foldr f v list
  - Composition: f . g x = f (g x)
  - Returning function values: f . g
    eg add 3 . mult 2 :: Int -> Int

Chapter 8

- Defining types

- Declaring types: type Board = [Int]
- Data declarations:

  data Answer = Yes | No | Unknown
- Constructors: eg Rectangle Float Float
- Parametric data types: eg data Tree a
- Recursive data types:

  data Nat = Zero | Succ Nat

Chapter 10

- Interactive programming; IO

  - The type IO a
  - Sequencing using do block and return
  - putStrLn, getLine: writing and reading from the terminal
  - Recursion in sequenced code.

Chapter 15

- Lazy Evaluation

- Evaluation by application of definitions
- Innermost and outermost reduction
- Sharing thunks
- Lazy evaluation = Outermost reduction + Sharing
- Lazy evaluation is efficient strategy
- Infinite lists: ones = 1:ones