Symbolic & Declarative Computing / Artificial Intelligence (COS5012-B)

October 19, 2016 Week 4 Lab2

LAB MANUAL: FUNCTIONS

In this tutorial you will learn about

- How to define functions
- Function Application
- Pattern matching and functions

Functions

Haskell is a functional language so the function concept is essential to the language. A function takes one or more arguments and computes a result. Given the same arguments, the result will always be the same. This is similar to a mathematical function and it means that in Haskell there are no side-effects. There are two fundamental operations on functions:

- Function definition (creating a function)
- Function application (using a function to compute a result).

Function Definition

Predefined: In Haskell, many functions are pre-defined in a standard library called the prelude.

User defined: But the essence of functional programming is defining your own functions to solve your problems!

A function is defined by an equation.

```
f x = x+1
```

This is equivalent to **`f(x) = x+1`** in mathematical notation.

Function Application

An application is an expression like `f 31`, where `31` is the argument.

The application is evaluated by replacing it with the body of the function, where the formal parameters are replaced by the arguments.

```
f  = \x - > x+1
  f 3
--  > {bind x=3}
  (x+1) where x=3
--  > {substitute 3 for x}
  3+1
--  >
```

```
4
```

Functions with multiple arguments

A function with three arguments:

```
add3nums x y z = x + y + z
```

To use this function

```
10 + 4* add3nums 1 2 3
= {- put extra parentheses in to show structure -}
  10 + ( 4* (add3nums 1 2 3) )
  -- >
  10 + (4*(1+2+3) )
  -- >
  10 + (4*6)
  -- >
  10 + 24
  -- >
  34
```

Function returning several results

- Actually, a function can return only one result.
- However, lists allow you to package up several values into one object, which can be returned by a function.
- Here is a function (`minmax`) that returns both the smaller and the larger of two numbers:

```
minmax x y = [min x y, max x y]
minmax 3 8  -- > [3,8]
minmax 8 3  -- > [3,8]
```

Pattern Matching

A function can have multiple patterns

```
guess :: Int -> [Char]
guess 42 = "correct!"
guess x  = "wrong guess!"
```

- Each pattern has the same type declaration
- Patterns are matched in order, top-down
- Only the first matched pattern is evaluated

- The patterns must exhaust the entire domain

```
guess :: Int -> [Char]
guess x  = "wrong guess!"
guess 42 = "correct!"
```

- Can you find the problem?

```
fib :: Int -> Int
fib n = fib(n-1) + fib(n-2)
fib 1 = 1
fib 0 = 1
```

- Whats wrong with the above code?
- Can you resolve the problem?

Pattern matching can also fail. If we define a function like this:

```
charName :: Char -> String
charName 'a' = "Albert"
charName 'b' = "Broseph"
charName 'c' = "Cecil"
```

Error Handling

Error messages can be produced through the `error` function.

```
Hugs> :t error
error :: String -> a
Hugs> error “There is some problem in your code”
```

Exercise

- Give your own implementation of the functions `fst` and `snd`.
- Write a function that computes the factorial of a number?
- Make a function that takes two vectors in a 2D space (that are in the form of pairs) and adds them together?
- Write your own versions of the following list functions
  - Head
  - Length
  - Sum

References

- http://learnyouahaskell.com/types-and-typeclasses