Programming Language Survey Paper on Haskell

Kurt O'Hearn
Nicholas Olesak

April 17, 2013

Contents

1 Introduction

- Origin and history
- Explanation of the functional programming paradigm
- Application domains of the language

[TODO] [1]

```
module Main where
import Data.List

main :: IO ()
main = putStrLn "Hello World!" >> test

test = do
    print 123
```

2 Data and Control Abstractions

2.1 Data Abstractions

2.1.1 Data Types

- Basis data types: Int, Float, Char, Boolean, lists

  ```
  let i = 2
  let x = 3.14
  let c = 'c'
  let str = ['H','a','s','k','e','l','l','!']
  ; shorthand for the above
  let str2 = "Haskell!"
  ```

- Type variables: variables that can be any type (used in polymorphic functions)

  ```
  lastTwo :: [a] -> [a]
  lastTwo a = [a !! ((length a) - 2), a !! ((length a) - 1)]
  ```

2.1.2 Type Checking

- Haskell: static typing, type inferencing by context

```
let myStr = "2"
; will result in an exception as the desired type to convert to is
; ambiguous since Haskell has no context to infer the type
let myNum = read myStr
; not ambiguous, as the compiler infers the Int type is desired
let myNum2 = (read myStr) + 2
```

- Type classes: Eq, Ord, Show/Read, Bounded, Enum, Num/Integral/Floating

2.2 Control Abstractions

2.2.1 Expressions

- Function definitions
- Binding: let, where

2.2.2 Operators and Precedence

- Operators: postfix notation, naming typically consists of symbols
- Functions: infix notation, named using alphanumeric characters in camel case
- Any function can be used as an operator by calling it in postfix notation

```
f :: (Int a) => a -> a -> a
f a b = a + b
; invoke the function in postfix notation
print (1 ‘f‘ 2)
```

- Any operator can be called as a function in infix notation

```
print ((+) 2 5)
```

- Operators can be overloaded

```
(*) :: (Num a) => a -> a -> a
(*) a b = a * a * b
print (2 * 3)
```

2.2.3 Selection Constructs

- Pattern matching
- if/then/else
- Guards
- Case expression

2.2.4 Iterative Constructs

- List comprehensions
- Map/fold
- Recursion

2.2.5 Functions

- Definition/use
- Support parameter passing techniques

2.2.6 Scoping

- Static

2.2.7 Modules

2.2.8 Exception Handling

3 Advanced Topics

3.1 Inheritance

3.2 Concurrency Support

3.3 Introspection

References

[1] M. Lipovaa. Learn you a haskell for great good! http://learnyouahaskell.com/, 2011.