Mini Course: Functional Programming

Session 1: Haskell syntax, Functions…

Jeremie Gillet

Outline

- Session 1:
  - Haskell syntax
  - Pure functions
  - Function composition
  - Currying
- Session 2:
  - Pattern Matching
  - Recursion
  - Folds
  - Maybe
- Session 3:
  - Immutability
  - Laziness
  - Parallelization
  - Exercises
- Session 4:
  - Algebraic Data Types
  - Mapping over ADTs
  - Reader, Writer, State...

Some Functional Programming Languages

In no particular order

- Common Lisp
- Scheme
- Clojure
- Wolfram Language
- Racket
- Erlang
  - Elixir
- OCaml
- Haskell
- F#
- Elm

Languages with some FP features

- C++11
- Kotlin
- Perl
- PHP
- Python
- Raku
- JavaScript
- R
- …

Haskell Syntax

Why Haskell?

Other than “because it blows my mind”

- Some features:
  - general purpose
  - statically typed
  - purely functional
  - type inference
  - lazy evaluation
- Developed in academia, many advanced features and applications (from research in type theory, category theory…), used in industry too
- One main compiler: GHC (Glasgow Haskell Compiler)
- Complete paradigm shift from other languages, flagship for FP
- Great tool for: parsing, compilers…

ghci

REPL for GHC

- 1 + 1 => Should return 2
- :set prompt “Mini Course> ” => Changes the prompt
- :load /path/to/file.hs, :l /path/to/file.hs => Load file
- :reload, :r => Reload last loaded file
- :cd /path => Change directory
- :! command => Runs bash command
- :type expr, :t expr=> shows the type of an expression
- tab => Autocomplete various things
- Not covering: times (:set +s), importing modules, help (:?)…

Haskell Syntax

Let’s go over some examples

- Dropbox > Public Files > HaskellSyntax.hs
- Open in a code editor (I will use Visual Studio Code with the Haskell 1.2.0 extension)
- Load with ghci and follow along, reload after a change
- Ask anything!
- Not covering: guards, records, list comprehension, type synonyms, IO, language pragmas, type classes…

Functions

Pure Functions

Definition

- Always return the same value for the same arguments
- No side effect
- Examples: max, floor, (+)
- Counter-examples: ls, rand()
- I/O is inherently impure, treated in a special way in FP
- Advantages
  - Easy to test
  - Easy to parallelize
- Purely functional programming language

Pure Functions

In Haskell

- \ x -> x + 1
- f = \ x -> x + 1
- f x = x + 1
- f x y = x + y + 1
- f =\ x y -> x + y + 1
- f x = \y -> x + y + 1
- f x = let y = 74 in x + y
- f x = x + y where y = 74
- Operators are functions too: (+) (*)
- Cool functions: id, const, flip

First Class Functions

A necessity for FP

- “Functions as first-class citizen”
- Functions can be passed as arguments
- Functions can return functions
- Functions can be assigned to variables
- Support anonymous functions
- Examples: map, filter, add1

Partial Application

In Haskell

- All of these are equivalent
  - f x = 1 + x
  - f x = (+) 1 x
  - f = (+) 1
  - f = (+1)
- Fun with functions (explore with :t in ghci)
  - (,,) 3 5
  - map (*) [1..5]
  - filter even
  - Redefine the function zip using zipWith

Function Composition

In Haskell

- add1 x = x + 1
  mult10 x = 10 * x
- Doing both operations:
  - add1 (mult10 x)
  - (add1 . mult10) x
  - mult10ThenAdd1 = add1 . mult10
  - mult10ThenAdd1 = (+1) . (*10)
  - mult100ThenAdd11 = mult10ThenAdd1 . mult10ThenAdd1
- Order matters, read right-to-left
- Functions need to be type compatible, check :t (.)

Currying

Spice up your programming style

- Converting a function with multiple arguments to a sequence of function with only one argument
  - x = f(a, b, c), a function that takes 3 parameters at once and returns a value x becomes
    - h = g(a)
      i = h(b)
      x = i(c)
      - Equivalently: x = g(a)(b)(c)
        g is a function that takes one parameter a and returns h, (a function that takes one parameter b and returns i, (a function that takes one parameter c and returns a value x))
- In Haskell, **every** function only takes a single argument
- Currying is built-in the notation: f a b c is the same as (((f a) b) c)
- Reading the type signature: a -> b -> c -> d is the same as a -> (b -> (c -> d))
- See also: curry, uncurry

Currying Exercises

Pen and paper first, then :t in ghci to verify

- Exercises
  - Simplify: curry id
  - Simplify: uncurry const
  - Express snd using curry or uncurry and other basic Prelude functions and without lambdas
  - Write the function \(x,y) -> (y,x) without lambda and with only Prelude functions