Notes Functional Programming Presentation

Bas Bossink

April 2010

0.1 Contents

- Taxonomy = classification

0.2 Introduction

0.2.1 Disclaimer

- No *real* project experience with FP
- Just read some books and did a few excercises

0.2.2 Inventory

- Who has used a functional language?
- Who has used a functional language after completing their education?
- Who has written a LINQ query?

0.2.3 Anecdote

- Euclid Greek mathematician 300 BC known for:

    - Elements, 13 books containing from the ground up geometry and number theory
    - Euclidean algorithm, efficient algorithm to calculate gcd

    When Ptolemy the First asked Euclid if there was no shorter road to geometry than the Elements, he replied, "there is no royal road to geometry."

0.2.4 About

- Open-source GFDL

- On github

- Cross platform

- Tools used:

  - pandoc (Haskell markup 2 markup transformer)
    * S5 (requires firefox)
    * just HTML, CSS, javascript
  - dot (graph layout engine)
  - Rake (ruby build automation)
  - git (source control)
  - vim (ide)
  - gnuplot (plotting graphs)
  - perl
  - Miktex/latex (generate notes in pdf)
  - imagemagick (svg to png conversion)
    * native svg in firefox ok on linux but broken on windows

0.3 Definition

0.3.1 first class citizen

- comparison to OO:

  - objects can be created
  - passed around
  - returned from methods
  - mutated
  - transformed into new objects

- support for operations on functions:

  - composition
  - (partial) application

0.3.2 functions are mathematical

- same input, same output

0.4   Example

- for each x there's 0 or 1 y value, never 2 several x's can have the same y

0.5   Non-example

- same input x, can have multiple y values

0.6   History

0.6.1   Lambda calculus

- Research in foundations of mathematics
- formalise function definition, application, recursion
- typed is computationally weaker but logically consistent

0.6.2   Timeline

- Stripped down history of programming languages
- Lisp one of the first programming languages
  - predates C by 14 years
  - based on untyped lambda calculus
- Scheme attempt to clean-up Lisp
- Erlang also influenced by Prolog
- CLOS Common Lisp Object System: OO added to Lisp
- OCaml is OO added to Caml
- Scala runs on both JVM and .Net
- F# syntax compatible with OCaml
- Haskell 2010 is about to be released
  - Haskell is designed by committee
  - Haskell was designed to focus research of typed, lazy functional languages
- Clojure a Lisp on the JVM and .Net

0.7 Taxonomy

- Languages can be classified in different ways

- Next is rough sketch, no scale

- To see who your friends are

0.7.1 Static/Dynamic, Strong/Weak

- Functional languages are mostly strongly typed

- Two basic camps still exist

    - no equivalent of Duck Typing, no *objects* to receive the message

0.7.2 Lazy/Eager, Single/Multi-paradigm

- Erlang, really functional, Armstrong says: "Concurrency Oriented"

- Common Lisp, CLOS, Common Lisp Object System

- F#:

    - Functional
    - Imperative
    - Object-Oriented
    - Language-Oriented

- Scala:

    - Functional
    - Object-Oriented

- Lazy here means by default, F# and Scala support lazy with special syntax

0.8 Features

0.8.1 Read Eval Print Loop (REPL)

- Also found in:

    - Python
    - Ruby
    - Scala

- F# no interpreter but on the fly compilation

0.8.2 Type inference

- Lately mostly doing Haskell, preparation a bit disappointing F# inference does not use info about signature sqrt Haskell does.

- Haskell :type ghci is different from type of bound expression

0.8.3 Pattern matching

- Examples

- F# supports **Active Patterns** functions to be run as part of pattern matching Makes it possible to match over arbitrary types and parameterize pattern matching.

0.8.4 Algabraic data types

- What's in a name

- Abstraction of tuple/enum

  an algebraic data type (sometimes also called a variant type[1]) is a datatype each of whose values is data from other datatypes wrapped in one of the constructors of the datatype. Any wrapped datum is an argument to the constructor

- Somewhat similar to `union`

0.8.5 Function Composition

- Examples

0.8.6 Anonymous function (lamdba expressions)

- Examples

Closure

  In computer science, a closure is a first-class function with free variables that are bound in the lexical environment. Such a function is said to be "closed over" its free variables.

Java JSE 7 proposed syntax quite ugly

0.8.7 Sequence Expressions (List Comprehensions)

- Think PowerShell ranges on steroids

- Also available in Python and Erlang

- F# generalizes list comprehensions to Sequence Expressions with syntactic sugar for seq, list and array

0.8.8 Partial function application

- Also known as Currying

- show arrows in types

0.8.9 Higher order functions

- functions that take functions as arguments or return functions

1 Functional Concepts

1.1 Tail Recursion

- example

- needed optimization for arbitrary length recursion

1.2 Continuation passing style

- example

- naive is **not** tail recursive

    - large and/or unbalanced tree's -> stackoverflow

- continuation passing, pass the function that should be invoked with answer

- both sizeCont left (fun leftSize -> .. are tail recursive

1.3 Memoization

- think caching in a closure

- example

- caching in a local function

2 Monads

- hiding complexity
- think control structures
- chaining computations
- cs example

3 When to use it?

3.1 Stateless

- web/REST

4 When not to use it?

- a lotta state
- interoperability, e.g. Haskell, Erlang only interoperate with C
- team knowledge