Page 1

2017-12-08 16:57

Functional Programming

Reflections

Questions and Answers

Page 2

Course content

- Functions as first-class values.
- Algebraic and abstract data types.
- Polymorphism and classes.
- Testing functional programs.
- Lazy evaluations and infinite objects.
- Monads.

Page 3

Learning outcomes

- Write small to medium-sized functional programs for a variety of applications.
- Exploit a variety of programming techniques typical in functional programming, such as:
  - Use of recursion,
  - Modelling with recursive datatypes,
  - Abstraction and reuse with the help of higher order functions and monads.
- Appreciate the strengths and possible weaknesses of the functional programming paradigm.

Page 4

Functions as first class values

- This is a key aspect of functional programming.
- Higher order functions
  - Powerful way to build modular and reusable code.
    - Especially when combined with polymorphism.
- Using functions to represent data

Page 5

Example: an abstract data type for sets

```
data Set a

empty          :: Set a
singleton      :: a -> Set a
insert         :: a -> Set a -> Set a

union, intersection, difference
               :: Set a -> Set a -> Set a

complement     :: Set a -> Set a

member         :: a -> Set a -> Bool

toList         :: Set a -> [a]
```

- Probably need some constraints, `Eq a` or `Ord a`, depending on implementation.

Page 6

How to represent sets

Three possible representations

- Lists

```
data Set a = Set [a]
```

- Binary search trees

```
data Set a = Empty | Node a (Set a) (Set a)
```

- Functions

```
data Set a = Set (a->Bool)
```

- Which is easier?

Page 7

Representing sets as lists

```
data Set a = Set [a] -- Invariant: no duplicates

empty = Set []
singleton x = Set [x]

insert x (Set xs) | x `elem` xs = Set xs
                  | otherwise   = Set (x:xs)

member x (Set xs) = x `elem` xs
```

```
union        (Set xs) (Set ys) = Set (xs++[y|y<-ys,y `notElem` xs])
intersection (Set xs) (Set ys) = Set [x|x<-xs,x `elem` ys]
difference   (Set xs) (Set ys) = Set [x|x<-xs,x `notElem` ys]
```

- Fairly easy. Only finite sets. Only types in the **Eq** class. No `complement`.
- Variant: keep the lists ordered for efficiency…

Page 8

Representing sets as binary search trees

```
data Set a = Empty | Node a (Set a) (Set a)
-- Invariant: smaller elements to the left,
--            bigger to the right
empty = Empty
singleton x = Node x Empty Empty

insert x Empty = singleton x
insert x (Node y l r) | x==y = Node y l r
                      | x<y  = Node y (insert x l) r
                      | x>y  = Node y l (insert x y)

member x Empty        = False
member x (Node y l r) = x==y ||
                        member x (if x<y then l else r)
```

- More complicated. Only finite sets. Only types in the `Ord` class. No `complement`.
- Even more complicated if we want to keep the trees balanced. (Data.Set)

Page 9

Representing sets as functions

```
data Set a = Set (a->Bool)

empty = Set (const False)
singleton x = Set (==x)
insert x s = union (singleton x) s

member x (Set f) = f x
```

```
union        (Set f) (Set g) = Set (\x->f x || g x)
intersection (Set f) (Set g) = Set (\x->f x && g x)
complement   (Set f)         = Set (not . f)

difference s1 s2 = intersection s1 (complement s2)
```

- It's the easiest of the three! All operations are one-liners!
- Allows infinite sets and `complement`, but no `toList`, unlike the others.

Page 10

Software prototyping experiment (1)

Haskell vs. Ada vs. C++ vs. Awk vs. ...
An Experiment in Software Prototyping Productivity

Paul Hudak and Mark P. Jones

Research Report YALEU/DCS/RR-1049
October 1994

This work was supported by the Advanced Research Project Agency and the Office of Naval Research under ARPA Order 8888, Contract N00014-92-C-0153.

Page 11

Software prototyping experiment (2)

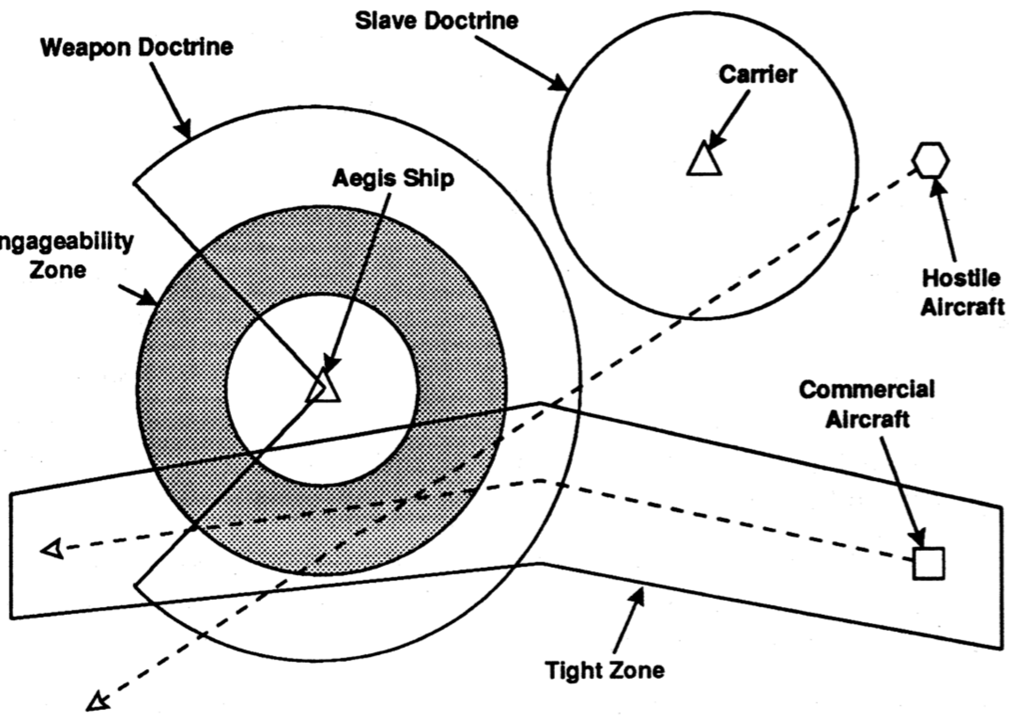

Figure 2: Geo-Server Input Data

Page 12

Software prototyping experiment (3)

Language	Lines of code	Lines of documentation	Development time (hours)
(1) Haskell	85	465	10
(2) Ada	767	714	23
(3) Ada9X	800	200	28
(4) C++	1105	130	–
(5) Awk/Nawk	250	150	–
(6) Rapide	157	0	54
(7) Griffin	251	0	34
(8) Proteus	293	79	26
(9) Relational Lisp	274	12	3
(10) Haskell	156	112	8

Figure 3: Summary of Prototype Software Development Metrics

Page 13

Software prototyping experiment (4)

Key design choice

```
type Region = Point -> Bool

type Point = (Double,Double)
```

- This makes all operations on regions easy to define
  - Basic shapes: circles, rectangles, etc
  - Geometric transformations, e.g. moving, scaling & rotating regions
  - Unions, intersections, complements
  - Membership tests

Page 14

What can Haskell be used for?

Examples

- GHC is implemented in Haskell.
- Hackage: lots of free Haskell libraries and applications.
- Investment banking: financial modelling, quantitative analysis.
- Facebook: HaXL, spam filtering.
- Keera Studios: game development, Android.
- More: Haskell in Industry
- Haskell Communities and Activities Report, November 2017 issue

Page 15

What can Haskell be used for? → Examples

From lwn.net: Stephen Diehl: Reflecting on Haskell in 2017:

- 14,000 new Haskell projects on Github!
- "It’s really never been an easier and more exciting time to be programming professionally in the world’s most advanced (yet usable) statically typed language."

Page 16

Some Haskell software I have worked on

- These slides: formatting and syntax high-lighting.
- WebFudgets.
- Programatica: Haskell compiler front-end (2001-2006)
- House: a prototype operating system in Haskell (2004-2006)
- Hardware emulation (6502 8-bit processor, used in C-64)
- An e-commerce system in Haskell (2006-2009)
- A web browser in Haskell (mid 1990s)
- Alfa: GUI for the proof assisant Agda (mid 1990s)
- Fudgets: GUI library in Haskell (early 1990s)

Page 17

Questions and Answers

Page 18

Recommended video

- [Keynote: Why Functional Programming Matters - John Hughes, Mary Sheeran] (Code Mesh, London 2015)

Page 19

The End

- Good luck with your projects!
- See you next week!