COMP0020: Functional Programming

University College London

Academic Year 2019

Course Objective

- Explores the functi ctional programming lan
- Uses the functiona l

NB : this module **does not aim to teach Haskell** s Haskell in passing).

Student Objectives

1. Understand the basics of the lambda calculus and combinators and how they are used in the implementation of functional languages.
2. Understand the m
3. Understand type c m as implemented in Miranda.
4. Write, understand and analyse non-trivial functional programs i
5. Understand the computation and memory management issues a implementation of lazy functional languages.
6. Solve problems relating to all of the above, under examination conditions.

Students are expected to i .

In the second half of the course students are expected to use independent study to **ead extensively** about implementation issues, which are then disc

Approximate Schedule of Lectures

Tuesdays :	10am	Bentham House LG26 Lecture Room
Wednesdays :	10am	Bentha
Fridays :	1pm	Cruciform Building B4

Start of Term :	Monday 13th January 2020
5 weeks of lectures :	Lambda Calculus, Miranda Programming, Advanced C
(Reading Week)	
4-5 weeks of lectures :	Advanced Concepts, Implementation Techniqu

Moodle page : **CO** COMPGC16) — essential reading !

There is no marked coursework, but students must complete formative e lectures. The book and the Moodle page have simple self-study exercises a past paper questions, without answers. Feedback is given on attempted s questions.

Introduction t

Many people think of Alan Turing as 'the father of computing", but there are many other important figures in computer science. In particular, Alonzo Church (Turing's PhD supervisor).

Church and Turing prom ... g :

- Turing was interes ... l set of rules.
- Church was interested in how to express problems in a precise way, to b ... transformation using a small set of rules.

These approaches have much in common, but they lead to two radically different styles of programming.

Consider the notion of programming languages in general :[1]

- How many progra
- Does the choice of lan
- Are older program
  - should we only u
  - are the rest just “junk” ?

1. See “The next 700 programming laguages”, P.J.Landin, 1965 at https ://homepages.inf.ed.ac.uk/wadler/papers/papers-we-love/landin-next-700.pdf

- Can you just use (say) Java for every programming task?
  - if a language (a at is Turing-equi
  - Since many pr

- What else do progra protection from common errors (e.g. type systems), different wa
- Different languages, different computational models,

2. Turing-equivalence and Turing-completeness are outside the scope of this module.

- It can be helpful to group programming languages into different categories (though it can be difficult to do this precisely, since some languages incorporate more than one computational model).

- There are many ways to do this ; one way is to identify
  - the “imperati
    - ★ e.g. Java

  - the “declarative” languages (typically based on the concept of sol
    - ★ e.g. Haskell, Miranda, Prolog

- The class of “declarative” languages contains both “functional” languages (e.g. based on Church’s λ-calculus) and “logic” languages (e.g. based on first-order predicate calculus)

A simple example of the difference in programming style

- Assume “results” is the name of a variable that contains a sequence of 100 integers, and write code to select all those with a value less than 10
- Imperative style :


```
int      small[100];
int      j,k;
for      (j = 0, k = 0; j < 100; j++)
           if (results[j] < 10){ small[k] = results[j];
return   (small);
```


- Functional style :

$$filter\ (< 10)\ results$$

  - Concise, low syntactic “clutter”, reduced need to specify storage of intermediate results

An example of literate programming style using Miranda

A practical example of functional programming style

- Functional Prog

  - But are they like Japanese Haiku poetry (elegant, but not very practi

  - Or are they like Karate (elegant, and useful in a fight) ?

- Prototyping a large object-oriented design using a functional programming language
  - A commercia
  - The world's la
  - A "mission-critical" financial system
  - Over 100 programmers
  - C++ required by client

Component-based system design : a network of components (“nodes”) communicating via streams of data (”arcs”). One or more inputs (“arcs”), one output (“arc”).

- Project requirements :
  - Discrete-ev

  - Prototyping of central optimisation and approximation algori
  - The main constraint was C++

    - ⋆ Too slow for rapid prototyping work (execution speed was very fast, but development time and debugging effort too great for prototyping many different designs)

- IT Consultancy's dilemma :
  - C++ required by client, but "not viable" for prototyping/simulation
    - Would t
  - Rapid protot
    - Smalltalk known to client — raised issue of suitability of client's choice of C++ (consultancy did not wish to "insult" client)

- Alternative approach — use a functional language (Miranda)
  - Higher Order, Statically Typed, Lazy, with Garbage Collection, no pointers, no assignment
  - Unknown to client (!)

- Selling points :

    - Speed and Clarity with which algorithms can be
        - express
        - validat
    - Can simulate key object-oriented designs in detail
        - With minimal detail for other components!

- Access to expertise :
    - A "champion"

- Note : Speed of execution was almost totally irrelevant !

Modelling the component network

FIGURE : A network of components (nodes) sending streams of d

- Recursive (looping) functions a, b, c and d
- c1, c2 etc. are streams of (time, value) events — represented by potentially-infinite lists of 2-tuples

- Assume recursive (
- Define the streams o

$$
\begin{aligned}
c1 &= read\ ''file1''\\
c5 &= read\ ''file2''\\
c9 &= b(c2, c6, c8)\\
(c2, c3) &= a(c1, c4)\\
(c4, c6) &= c(c5, c7)\\
(c7, c8) &= d(c3)
\end{aligned}
$$

- Simulating behaviour
  - Simple, beha
  - Expression-
- Synthetic (statistical) data generation
- Used Miranda algorithms as specification for subsequent implementation in C++

Results

- Rapid development
  - About 5 times fa
- Concise expressio
  - 6 pages of Miranda = about 25 pages of C++
- Simulation and specification of complex processes
  - Design optimised early in lifecycle
  - Confidence increased through validation on real data

Results

- Almost NO errors in

- Vast reduction in er

- Viewed as a commercial advantage

- Promoted worldwide within the IT Consultancy — “champion” promoted to Manager