Introduction to Functional Programming

Anuj Dawar

Computer Laboratory
University of Cambridge
Lent Term 2000

Texts

Main Text:

Paulson, L.C. (1996). *ML for the Working Programmer*. Cambridge University Press (2nd ed.).

Other Useful Reading:

Backus, J. (1978). Can programming be liberated from the von Neumann style? *Communications of the ACM*, vol. 21, pp. 613-641.

Barendregt, H.P. (1984). *The Lambda Calculus: its Syntax and Semantics*. North-Holland.

Landin, P.J. (1966). The next 700 programming languages. *Communications of the ACM*, vol. 9, pp. 157-166.

Slides available (after the lecture) from:

www.cl.cam.ac.uk/~ad260/ifunprog.html

Imperative and Declarative

In an imperative programming language, the program provides a series of instructions (or commands) to the machine.

Examples of such languages include
C, Pascal, Modula2, Java

In a declarative programming language, the program (in principle) describes the computational task.

Functional: ML, Scheme, Haskell,...
Logic: Prolog, Godel,...

Programming Views

Imperative languages present a level of abstraction above the machine, hiding some details (memory addresses, registers, etc.)

Still, the view is machine-centred.

Declarative languages provide a still further level of abstraction.

A style of programming that is more programmer-centred.

Functional programming

In the functional programming style, the computational task to be programmed is taken to be a function (in the mathematical sense).

The job of the programmer is to describe this function.

Implicit in the description is a method for computing the function.

The function maps one domain (of inputs) to another (of outputs).

These may be: integers; real numbers; lists; strings; or even functions themselves

importance of types

Commands and Expressions

In a typical imperative language, commands are formed from assignments to variables:

$$x := E$$

by application of various control structures.

Sequencing

$$C_1; C_2$$

Conditionals

$$\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2$$

Looping

$$\texttt{while}\ B\ \texttt{do}\ C$$

Expressions

A functional program is just an expression to be evaluated.

An expression is built up from simpler expressions by means of function applications.

$$E_1 + E_2$$

or

$$\texttt{if } B \texttt{ then } E_1 \texttt{ else } E_2$$

There are no explicit notions of variable assignment, sequencing or control.

Example: the factorial

The factorial function can be written imperatively in C as follows:

```
int fact(int n)
{ int x = 1;
  while (n > 0)
   { x = x * n;
     n = n - 1;
   }
  return x;
}
```

whereas it would be expressed in ML as a recursive function:

```
fun fact n =
  if n = 0 then 1
  else n * fact(n - 1);
```

Recursion

Recursive definition of functions is crucial to functional programming.

There is no other mechanism for looping

Variables cannot be updated through assignment. They get their values from function calls.

Type Checking

ML provides type checking, which can help catch many programming errors.

Types in ML may be polymorphic.

```
fun length []      = 0
  | length (x::l) = 1 + length (l);
```

Advantages

"Attack complexity with simple abstractions"

- Clarity
- Expressiveness
- Shorter Programs
- Security through type system
- Ease of reasoning
- Better modularity

Disadvantages

- Input/Output
- Interactivity and continuously running programs
- Speed/Efficiency

There is no reasonable "pure" functional language

Brief History

- Lambda Calculus (Church 1936)
- LISP (McCarthy 1954)
- ISWIM (Landin 1966)
- ML (Milner et al., 1974), originally a Meta Language for the LCF Theorem Prover.
- Definition of Standard ML (Milner, Tofte and Harper 1990)
- Revised definitiona and standard library (1997)

Rest of the Course

11 more lectures covering

- Basic Types in Standard ML
- Lists and Recursion
- Sorting
- Datatypes
- Higher Order Functions
- Specification and Verification
- Types and Type Inference
- Substantial case study

Running ML

ML provides an interactive session.

Enter an expression. ML returns a value.

```
Moscow ML version 1.42 (July 1997)
Enter ‘quit();’ to quit.
- (2*4) + 18;
> val it = 26 : int
- 2.0 * 2.0 * 3.14159;
> val it = 12.56636 : real
```