Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 0 - Sequence 1:

Functional Programming : a bit of history and motivation

Computing and *programming*

Computing

the study of algorithmic processes that describe and transform information.
The fundamental question is *"What can be (efficiently) automated?"*

1989 ACM report on Computing as a Discipline

Basic components of computing

- a program that describes the intended transformation of information
- a machine that executes the program

Many machines, and many ways of writing a program.
Some were invented well before the first modern computer.

Let's recall a bit of their history

Hilbert, ...

Hilbert's decision problem (1928)

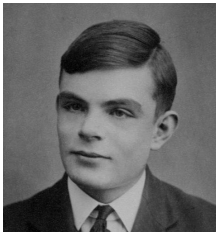
Can we devise a process to determine
in a finite number of operations
whether a first-order logic statement is valid?

The answer to this question is "no", but to find it, one needs to make precise what is

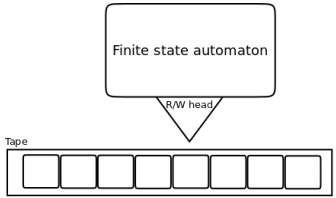
- an operation and
- a process of computation

Turing ...

Alan Turing answers Hilbert's question in 1936



inventing the now world-known Turing machine



Theoretical foundation of modern computers and imperative programming

- tape $\approx$ addressable read-write memory with stored program
- automaton $\approx$ microprocessor

Turing Machines and Imperative programming

In an imperative program we *read, write, perform operations and take decisions* based on the contents of *memory cells* that hold the contents of variables like `c`,`n`,`res` in the following Java simple example.

```
public class Factorial
{
    public static int compute(int n)
    {
        int res = 1;
        for (int c = 1; c <= n; c++)
            res = res * c;
        return res;
    }
}
```

Church ...

Alonzo Church (*Alan Turing's advisor*)

Also answers Hilbert's question in 1936 with a completely different approach, inventing the λ-calculus

- $\lambda x.M$ = nameless function with formal parameter x and body M (abstraction)
- MN = call function M with actual parameter N (application)

Theoretical foundation of functional programming

$$(\lambda x.M)N \rightarrow_\beta M[x := N]$$

The β reduction rule is the one and only computational device in the system!

The λ-calculus and Functional programming

In a *functional program* we *define* (possibly recursive) functions, and *compose* and *apply* them *to compute* the expected *results*.

Like in the following example

```
let rec fact =
      function n -> if n=0 then 1 else n*(fact (n-1))
```

In a truly functional programming language, functions are *first class citizen*. They can be:

- named
- evaluated
- passed as arguments
- returned as results
- used everywhere an expression can fit

The value of this will become evident at the end of the course!

The λ-calculus and Functional programming

(Ab)using Church's original notation one would write the second line

$$\lambda\ n.\text{if n=0 then 1 else n*(fact (n-1))}$$

That's why you hear all this excitement about *lambdas*
being introduced in Java and C++ in these recent years...
yes, they are just introducing *real* functions in the language!

The Church-Turing thesis

Equivalence of Turing machines and λ-calculus (Turing, 1937)

A function is computable by a Turing machine, if and only if it is computable using λ-calculus

Church-Turing thesis

A function that is computable by any computing device is also computable by a Turing machine

In simpler terms

All general purpose programming languages are computationally equivalent

But programming languages are not born equal...

They have *different expressiveness*

The quest for more expressive constructs is never ending, leading to

- different data representations
- different execution models
- different mechanisms of abstraction

And there are many other desirable features

- safety of execution
- efficiency
- maintainability
- ...

Depending on the problem at hand, some programming languages may be way better than others.

An early assessment from FORTRAN's very creator

Functional programs deal with structured data, ... do not name their arguments, and do not require the complex machinery of procedure declarations ...

Can programming be liberated from the von Neumann style?
John Backus, Turing lecture, 1978

Why functional programming is on the rise

Quoting the report on Introductory Computer Science Education at CMU
`http://www.cs.cmu.edu/~bryant/pubdir/cmu-cs-10-140.pdf`,
there are some clear emerging trends

Need for greater software reliability

(Pure) functional programs are easier to prove correct than imperative ones

Harnessing the power of parallel computation

A carefully chosen set of higher order functions allows to write programs that are easily parallelisable.

A very well known example: MapReduce

Functions all around us

The power and expressivity of functional programming is being recognised widely:

- Java 1.8 introduces *lambda* expressions
- C++ version 11 introduces *lambda* expressions

No matter what your preferred programming language is,... understanding functional programming principles is now a basic skill.

We will learn them using the *OCaml* language!

Credits

Photos

David Hilbert's photo :
https://commons.wikimedia.org/wiki/File:Hilbert.jpg, public domain.
Alan Turing's photo :
https://fr.wikipedia.org/wiki/Alan_Turing#/media/File:Alan_Turing_Aged_16.jpg, public domain.
Alonzo Church's photo :
https://en.wikipedia.org/wiki/File:Alonzo_Church.jpg, under *fair use* terms.