University of Warwick
Department of Computer Science

CS141

Functional Programming

Cem Yilmaz

January 11, 2022

Contents

1 What is Functional Programming **2**

1.1 History . . . 2

1.2 Today . . . 2

1.3 Programming Paradigms . . . 3

1.4 What Haskell is good for . . . 3

1.4.1 Web Services . . . 3

1.4.2 Domain-specific language . . . 3

1.4.3 Games . . . 4

1.4.4 System Software . . . 4

2 Basics of Haskell **4**

2.1 Modules . . . 4

2.2 Imports . . . 4

2.3 Definitions . . . 5

2.4 Lambda Notation and Mapping notations . . . 5

2.5 Partial Functions . . . 5

2.6 Smooth Operator . . . 5

2.7 Difference between a function and an operator . . . 6

2.8 Pattern Matching . . . 6

1 What is Functional Programming

1.1 History

In 1928, David Hilbert posed the question "Given any true mathematical statement, is there an algorithm for verifying that it is true?". This is now known as the "Entscheidungsproblem", that is, the decision problem. E.g.

$$\begin{aligned} 1 &= 1 \\ 2 &= 2 \\ 2 &< 3 \\ 1 &< 3 \end{aligned}$$

In 1931, Kurt Gödel came up with the paradox with the statement "This statement is not provable". If we assume not provable, there is a double negation which implies it is provable. This contradicts the assumption. If we assume true, we have a true statement that is not provable. This is called the incompleteness theorem. This answered the question that mathematics cannot answer every statement.

In 1936, Alonzo Church came up with $\lambda-$calculus as a system for describing algorithms. Kurt Gödel then believed that he can do better. He believed that some algorithms would not be able to be described with $\lambda-$calculus. Kurt then came up with a system of recursive functions. Alonzo Church then claimed that any algorithm that can be described using recursive functions can also be described using $\lambda-$calculus.

Alan Turing then came along and then came up with his own system of describing algorithms utilising Turing machines. However, he also showed that anything described using Turing machine could be described with $\lambda-$calculus. However, notice that despite these being different systems, they were also equivalent in describing algorithms.

1.2 Today

Today, programming as you know it, for example:

$$\prod_{i=1}^{4} = 1 \times 2 \times 3 \times 4$$

If we wanted to turn this to the roughly equivalent program in a language such as Java or C

```java
int x = 1;
for (int = 1; i <= 4; i++) {
  x *= i;
}
```

Listing 1: Product in Java

Table 1: Table of results

Variable	Value
x	1
x	2
x	6
x	24

However, in a function language, we can express it as

```haskell
product[1..4]
```

Listing 2: Product in Haskell

That is, product is a function. You can also expand this to be

```
product[1,2,3,4]
1 * product[2,3,4]
1 * 2 * product[3,4]
1 * 2 * 3 * product[4]
1 * 2 * 3 * 4
24
```

Listing 3: Expanded Product

The definition of product function is rather simple:

```
product [n] = n -- Described in terms of 2 equations. First it checks if there a single item
    and returns it
product(n:ns) = n * product ns -- Takes the the first item in the list and keeps the rest
```

Listing 4: Product function

Let us now compare imperative programming with functional

Imperative	Functional
Mutation of state Tell the computer how you want to do something Statements executed in order specified Loops	Reduction of expression Tell the computer what you want to compute and let it work out how to do it Sub-expressions can often be evaluated in an arbitrary order Recursion

1.3 Programming Paradigms

In history, there used to be a clear distinction between programming paradigms. However, today, this has changed. That is,

- Java / C are now multi-paradigm: They're imperative, object-oriented, functional, etc.
- Python, JavaScript, C++ have similarly multi-paradigm.

As such, learning functional programming will be useful as paradigms have blended together.

1.4 What Haskell is good for

1.4.1 Web Services

Furthermore, a particularly nice application to functional programming is that they're good at web services.

- Lots of cool frameworks for developing web applications
- Easy to embed domain-specific languages for routing, templates, etc.
- Servant: describe web service as a type, automatically generate client programs
- One of the coursework assignments uses a web service written in Haskell to provide a browser-based interface

1.4.2 Domain-specific language

Domain-specific languages are also a thing. For example, you can write music in a Haskell library to describe music.

```
import Mezoo

v1 = d qn :|: g qn :|: fs qn :|: g en :|: a en :|: bf qn :|: a qn :|: g hn
v2 = d qn :|: ef qn :|: d qn :|: bf_ en :|: a_ en :|: b_ qn :|: a_ qn :|: g_ hn

main = playLive (v1 :-: v2)
```

Listing 5: Music in Haskell

What's cooler is that if we try to compile, we would get an error that the composition is not harmonic, that is, if it does not sound good, it does not compile. In particular,

- Major sevenths are not permitted in harmony: Bb and B_
- Direction motion in a perfect octave is forbidden: Bb and B_, then A and A_
- Parallel octaves are forbidden: A and A_, then G and G_

1.4.3 Games

For example, the game magic cookies utilises functional reactive programming to describe its game logic. It is the same code across different platforms. It is also good for time-travel debugging.

1.4.4 System Software

- XMonad: Window manager
- OS: Mirage (OCaml), House (Haskell) and more

2 Basics of Haskell

2.1 Modules

Haskell code is separated into files called modules. A module file always begin with

```haskell
module Whatever where
...
```

Listing 6: Module

Note that the filename ends with $.hs$, and is the same as the module name.

2.2 Imports

Similarly, you can import nearby modules. For example, for the file $foo.hs$:

```haskell
module foo where
double x = x * 2
```

Listing 7: Import

And then in another file,

```haskell
module bar where
import Foo
quadruple x = double (double x)
```

Listing 8: Import of foo

In the same way, libraries are imported as they are just functions. For example,

```haskell
module Whatever where
import Data.Char
swapCase c =  if isUpper c
              then toLower c
              else toUpper c
```

Listing 9: Library import

2.3 Definitions

A Haskell file is a series of definitions. For example, for the code

```haskell
module Defn where
triangle n = sum [1..n]
```

Listing 10: Defn.hs

On the LHS of = is a name, with zero or more arguments. On the RHS is an expression which utilises the arguments. Note that Haskell does not utilise parentheses. Arguments are separated by whitespace.

2.4 Lambda Notation and Mapping notations

Recall that in lambda calculus we denote inputs using lambdas, that is

$$\begin{aligned} &\lambda xy.x \times y \\ =&\lambda x \lambda y.x \times y \end{aligned}$$

Similarly, in Haskell, the lambda notation is blended with the mapping arrow. That is, for example, we can define our function $multiply$ in following ways:

```haskell
import Multiply where
multiply x y = x * y
multiply = \x y -> x * y
multiply x = \y -> x * y
multiply = \x -> \y -> x * y
```

Listing 11: Multiply.hs

These all are the same thing.

2.5 Partial Functions

It is possible to provide a singular argument into a function that takes in multiple arguments. For example, for our code above, if declared $multiply 5$, then we would replace x with 5. Note that it is x in particular because it is the variable that is declared first.

```haskell
multiply = \x y -> y * x
multiply 5 --(\y -> y * 5)
```

Listing 12: Multiply 5

2.6 Smooth Operator

Other than declaring functions, it is also possible to declare operators in a similar fashion. Consider the following code:

```haskell
x ^^^ y = max x y
```

Listing 13: Operator

One can spot the similarities. However, note that operator names are made up of symbols rather than letters. Similarly to partial functions, there are also partially applied operators. For example,

```haskell
plusFive = (+ 5)
```

Listing 14: Partially applied operator

This would add a 5 to any other operator.

2.7 Difference between a function and an operator

Functions are a prefix, whilst operators go between arguments (infix). It is possible to treat a function like an operator. Consider the following code:

```
GHCi > 5 ‘max‘ 6
6

GHCI> (*) 5 6
30
```

Listing 15: Function to operator

For example, the max function is enclosed within “ which allows us to use it like an operator.The latter operation is another alternative way of writing it.

2.8 Pattern Matching

Consider the following definition of factorial of a variable x :

```
fac x = if x == 0
           then 1
           else x * f (x - 1)
```

Listing 16: factorial

This can be rewritten as

```
fac x = case x of
    0 -> 1
    n -> n * f (n - 1)
```

Listing 17: factorial redefined

You can also express it in top-level patterns. That is, we pattern match our arguments directly

```
fac 0 = 1
fac x = x * f (x - 1)
```

Listing 18: factorial pattern matched

Often, Haskell code is written like this Lastly, you can also introduce "guards" for the code. If we need to check some predicate of the input and not a specific pattern

```
f x
      | x == 0       = 1
      | otherwise    = x * f(x-1)
```

Listing 19: factorial guard