Unit 4d: Partial Function Application

1 Introduction

In imperative programming we might think of calling a function and supplying parameters to the function to give it the information it requires. Generally you need to supply all the parameters, although some languages support default values; and, calling a function always returns a result of the function's return type.

Lambda calculus provides functional programming with an alternative way of thinking about this – we apply a function to the parameters - don't necessarily have to apply the function to all the parameters.

2 Example 1

For example, here is a simple function that adds three numbers, each of which is a parameter of the function, and returns the result as an **Int**.

```
def sum(a:Int, b:Int, c:Int): Int  = { a+b+c}
```

You can apply this to three parameters and assign the result to a variable *f,* which then refers to an **Int**.

```
scala> val f = sum(1,2,3)
f: Int = 6
```

Alternatively, you can apply this function to two parameters only and assign the result to a variable *f* – this is partial application of the function:

```
scala> val f = sum(1, 2, _: Int)
f: Int => Int = <function1>
```

Note the _ indicates that the function is not to be applied to the parameter at that position in the list. The result of partial application is not an **Int** – it is a **function**! So, the variable *f* refers to a function, not a value.

You can call the function later – now fully applied so get a value with the return type of the original function:

```
scala> val g = f(3)
g: Int = 6
```

3 Example 2

This function might be useful in a program that generates HTML – it will wrap an element with a specified prefix and suffix :

```
def wrap(prefix:String, html:String, suffix:String) = {
  prefix + html + suffix
}
```

This is a very general function which might be used in a wide range of situations. However, it might be common to need a specific wrapping, e.g. wrap an element in ** .** You can create a new function to do this by partially applying the general function:

```
scala> val wrapWithDiv = wrap("<div>", _: String, "</div>")
wrapWithDiv: String => String = <function1>
```

As in the previous example, the variable *wrapWithDiv* refers to a function – this takes one parameter instead of the three parameters of the original *wrap* function. We can call this to wrap any HTML; we don't need to call the general function and specify ** ** and ** ** each time.

```
scala> wrapWithDiv("<p>Hello, world</p>")
res0: String = <div><p>Hello, world</p></div>
scala> wrapWithDiv("<img src=\"/images/foo.png\" />")
res1: String = <div><img src="/images/foo.png" /></div>
```