Chapter 3: The Logic of Types, Part I

Higher-order functions

Sergei Winitzki

Academy by the Bay

December 2, 2017

Types and syntax of functions that return functions

"Curried functions" in Scala

- A function that returns a function:

```
def logWith(topic: String): (String ⇒ Unit) = {
   x ⇒ println(s"$topic: $x")
}
```

- Calling this function:

```
val statusLogger: (String ⇒ Unit) = logWith("Result status")
primaryLogger("success")
```

- One-line syntax: `logWith("Result status")("success")`
- Alternative syntax – "Curried" function:

```
val logWith: String ⇒ String ⇒ Unit =
  topic ⇒ x ⇒ println(s"$topic: $x")
```

- Syntax conventions: `x ⇒ y ⇒ z` means `x ⇒ (y ⇒ z)`
  - This is so because `f(g)(h)` means `(f(g))(h)`

Functions with fully parametric types

"No argument type left non-parametric"

Compare these two functions (note tuple type syntax):

```
def hypothenuse = (x: Double, y: Double) ⇒ math.sqrt(x*x + y*y)
def swap: ((Double, Double)) ⇒ (Double, Double) =
  { case (x, y) ⇒ (y, x) }
```

- We can fully parameterize the argument types for `swap`:

```
def swap[X, Y]: ((X, Y)) ⇒ (Y, X) = { case (x, y) ⇒ (y, x) }
```

- (The first function is too specific to generalize the argument types.)
- Note: Scala does not support a `val` with a parametric type
  - Instead we can use `def` or parametric classes/traits
- More examples:

```
def identity[T]: (T ⇒ T) = x ⇒ x
def const[C, X]: (C ⇒ X ⇒ C) = c ⇒ x ⇒ c
def compose[X, Y, Z](f: X ⇒ Y, g: Y ⇒ Z): X ⇒ Z = x ⇒ g(f(x))
```

- Functions with fully parametric types *are* useful despite appearances!

Worked examples

- For the functions `const` and `identity` defined above, what is the value `const(identity)` and what is its type? Write out the type parameters.
- Define a function `twice` that takes a function f as its argument and returns a *function* that applies f twice. For example, `twice(x ⇒ x+3)` should return a function equivalent to `x ⇒ x+6`. Find the type of `twice`.
- What does `twice(twice)` do? Test your answer on this expression: `twice(twice[Int])(x ⇒ x+3)(10)`. What are the type parameters here?
- Implement a function that applies a given function f repeatedly to an initial value x_0, until a given condition function `cond` returns true:

```
def converge[X](f: X ⇒ X, x0: X, cond: X ⇒ Boolean): X = ???
```

- Take a function with two arguments, fix the value of the first argument, and return the function of the remaining one argument. Define this operation as a function with fully parametric types:

```
def firstArg[X, Y, Z](f: (X, Y) ⇒ Z, x0: X): Y ⇒ Z = ???
```

- Infer missing types: `def p[...]:...  = f ⇒ f(2)`. Does `f(f)` work?
- Infer missing types: `def p[...]:...  = f ⇒ g ⇒ g(f)`

Exercises I

- For the function `identity` defined above, what is `identity(identity)` and what is its type? Same question for `identity(const)`.
- For the function `const` above, what is `const(const)`, what is its type?
- For the function `twice` above, what does `twice(twice(twice)))` do? Write out the type parameters. Test your answer on an example.
- Define a function `thrice` that applies its argument function 3 times, similarly to `twice`. What does `thrice(thrice(thrice)))` do?
- Define a function `ence` that applies a given function *n* times.
- Take a function with two arguments, and define a function of these two arguments swapped. Package this functionality as a function `swapFunc` with fully parametric types. To test:

```
def f(x: Int, y: Int) = x - y // check that f(10, 2) gives 8
val g = swapFunc(f)  // now check that g(10, 2) gives – 8
```

- Infer missing types: `def r[...]:...  = f ⇒ f(g ⇒ g(f))`
- Infer missing types: `def s[...]:...  = f ⇒ g ⇒ g(x ⇒ f(g(x)))`