Loops

Principles of Functional Programming

Martin Odersky

Loops

Proposition: Variables are enough to model all imperative programs.

But what about control statements like loops?

We can model them using functions.

Example: Here is a Scala program that uses a `while` loop:

```
def power(x: Double, exp: Int): Double =
  var r = 1.0
  var i = exp
  while i > 0 do { r = r * x; i = i - 1 }
  r
```

In Scala, `while-do` is a built-in control construct

But how could we define `while` using a function (call it `whileDo`)?

Definition of `whileDo`

The function `whileDo` can be defined as follows:

```
def whileDo(condition: => Boolean)(command: => Unit): Unit =
  if condition then
    command
    whileDo(condition)(command)
  else ()
```

Note: The condition and the command must be passed by name so that they're reevaluated in each iteration.

Note: `whileDo` is tail recursive, so it can operate with a constant stack size.

Exercise

Write a function implementing a `repeat` loop that is used as follows:

```
repeatUntil {
  command
} ( condition )
```

It should execute `command` one or more times, until `condition` is true.

Exercise

Write a function implementing a `repeat` loop that is used as follows:

```
repeatUntil {
  command
} ( condition )
```

It should execute `command` one or more times, until `condition` is true.

The `repeatUntil` function starts like this:

```
def repeatUntil(command: => Unit)(condition: => Boolean) =
```

Exercise (open-ended)

Is it also possible to obtain the following syntax?

```
repeat {
  command
} until ( condition )
```

?

For-Loops

The classical `for` loop in Java can *not* be modeled simply by a higher-order function.

The reason is that in a Java program like

```
for (int i = 1; i < 3; i = i + 1) { System.out.print(i + " "); }
```

the arguments of `for` contain the *declaration* of the variable `i`, which is visible in other arguments and in the body.

However, in Scala there is a kind of `for` loop similar to Java's extended for loop:

```
for i <- 1 until 3 do System.out.print(s"$i ")
```

This displays `1 2`.

Translation of For-Loops

For-loops translate similarly to for-expressions, but using the `foreach` combinator instead of `map` and `flatMap`.

`foreach` is defined on collections with elements of type `T` as follows:

```
def foreach(f: T => Unit): Unit =
  // apply ‘f‘ to each element of the collection
```

Example

```
for i <- 1 until 3; j <- ”abc” do println(s”$i $j”)
```

translates to:

```
(1 until 3).foreach(i => ”abc”.foreach(j => println(s”$i $j”)))
```