Lecture 36: Functional Programming for the Web
10:00 AM, Apr 24, 2018

Contents

Objectives

By the end of this lecture, you will have:

- Seen how to think about web scripts functionally, rather than as session state as described in the GUIZilla handout.
- Wrapped up CS18 (lectures, anyway)

NOTE: This material will NOT be covered on the final.

Today’s lecture was a discussion of how to write web scripts to work around the coding challenges posed by web browsers (improperly shared state between uses of the same pages, back buttons, saved pages/bookmarks, and more). The lecture is best followed through the video-capture, but here are the highlights:

Understanding the challenges posed by browsers: see the slides posted with the lecture notes for an example of hotel bookings across multiple pages (the lecture capture discusses these slides, but they didn’t make it into the video).

An imperative/text-based solution to choosing hotels: Here, we looked at the following code, and why it doesn’t translate neatly to the context of a web program.

```scala
class TravelSite {
  def BookTrip(criteria : Hotel => Boolean) : Unit = {
    val options = Listings.filter(criteria)
    var stillSearching = true
    var viewHotel = new Hotel()
    while (stillSearching) {
      display(options)
      var choice = getUserInput()
      var viewHotel = findHotel(options, choice)
      showHotel(viewHotel)
      if (getUserInput().equals("makeBooking"))
        stillSearching = false
    }
    makeReservation(viewHotel)
  }
}
```
The shared `viewHotel` variable works fine as a regular program, but on the web each hotel can be viewed on a separate page, so the shared variable might be holding an old value.

A simple example: adding numbers  Here’s a simpler example – a web page to add two numbers. The GUZilla handout discusses this same example, creating session state to get information from one web form to another.

```scala
object WebAdder1 {
    def getNum(): Int = {
        println("Enter a number: ")
        scala.io.StdIn.readInt()
    }
    def Adder1() = {
        println(getNum() + getNum())
    }
}
```

Functions (scripts/forms) on the web terminate rather than return control to other functions, so this program dies after the user enters the first number. Web scripts therefore have to be written in a way that keeps computations moving, by always calling the next function/form for the next step of the computation. For example:

```scala
object WebAdder2pre {
    def getNum1(): Unit = {
        println("Enter a number: ")
        getNum2(scala.io.StdIn.readInt())
    }
    def getNum2(n1 : Int): Unit = {
        println("Enter a number: ")
        printSum(n1, scala.io.StdIn.readInt())
    }
    def printSum(n1 : Int, n2: Int): Unit = {
        println(n1 + n2)
    }
    def Adder2() = {
        getNum1()
    }
}
```

Here, as each function that involves I/O finishes, it passes its data along to the next function. This is similar to the concept of tail-recursion that was covered in CS17.

Continuations: easy conversion of code to web programs:  Of course, rewriting your code from the first adder version to the second is annoying (we showed a simple adder here, but an entire
travel website would be worse). For those of you looking for a fun programming construct to study, look up “continuations” (Racket has them).

With continuations, a simple construct effectively turns the first version of the adder into the second. Here’s an example:

```scheme
; an original adder with I/O
(println (+ (getnum) (getnum)))

; ------------

; a revised adder
(define (getnum next-form)
  (next-form (readln "Enter a number")))

(+ (let/cc nextform (getnum nextform))
  (let/cc nextform2 (getnum nextform2)))
```

In this version, you can have getnum exit after calling next-form and the program still runs properly. This is a super-fast and sketchy overview of continuations. If you want more, also explained in the asme context of web programming, check out the following chapter from Professor Krishnamurthi’s “Programming and Programming Languages” textbook:

https://papl.cs.brown.edu/2015/control-operations.html

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: https://cs.brown.edu/courses/cs018/feedback