Scala solution
How to make your Scala controll effects a-la Haskell

George Leontiev

deltamethod GmbH

February 28, 2013

(λx.folonexlambda-calcul.us)@
folone.info

Introduction

Note: I intentionally made it more "interesting" to show more neat scalaz stuff
I won't cover everything though. If something seems strange, please ask.

Core

Main functions

```
wordCount :: String → Map (String, Int)
acceptedChars :: Char → Boolean
```

Helper functions

```
time :: (a → IO b) → IO b
close :: Closeable a ⇒ a →IO ()
```

```
def acceptedChars(c: Char) = {
  val sum: (((Boolean, Boolean), Boolean)) ⇒
    Boolean = _ match {
      case ((a, b), c)  a || b || c
    }
  val fun = ((_: Char).isLetterOrDigit) &&&
            ((_: Char).isWhitespace)    &&&
            ((_: Char) == '-')
  (fun >>> sum)(c)
}
```

```
def wordCount(text: String): Map[String, Int] =
  text.filter(acceptedChars)
      // split words
      .toLowerCase.split("\\W").toList
      // Optionally parallelize
      .par
      // group
      .groupBy(identity)
      // calculate group sizes
      .map { case(key, value) =>
        key.trim -> -value.length
      }
      // Get results from parallel computation
      .seq
```

Typeclass instances

```
val N = 10
implicit val mapInstances = new Show[List[(String,
  override def shows(l: List[(String, Int)]) =
    l.filterNot(_._1.isEmpty)
      .sortBy(_._2)
      .take(N)
      .foldLeft("") { case(acc, (key, value)) ⇒
        acc + "\n" + key + ": " + (-value)
    }
}
```

Executing

```
// function :: String → IO String
def main(args: Array[String]) = {
  val path   = args(0)
  val action = for {
    result ← time(function(path))
    _ ←          putStrLn(result)
  } yield ()
  // Yuck!
  action.unsafePerformIO()
}
```

All set

Let's see how far we can push this solution.

First attempt

```
def wholeFile(path: String): IO[String] =
  IO { Source.fromFile(path) }.bracket(close) {
    source ⇒
      IO {
        val text   = source.mkString
        val result = wordCount(text)
        result.toList.shows
      }
  }
```

First attempt

Works fine, but eats all the heap on a large enouth file.

Second attempt

```
def byLine(path: String): IO[String] =
  IO { Source.fromFile(path) }.bracket(close) {
    source ⇒
      IO {
        val stream = source.getLines.toStream
        val result = stream.map(wordCount)
          .foldLeft(Map.empty[String, Int]) {
            case(acc, v) ⇒ acc |+| v
        }
        result.toList.shows
      }
  }
```

Second attempt

Just what is this $|+|$?

Typeclasses

```
instance Show [(String, Int)] where ...
instance Show Monoid b ⇒ Map a b where ...
```

Monoids

$(S, \otimes, 1)$

$\forall a, b \in S : a \otimes b \in S$

$\forall a, b, c \in S : (a \otimes b) \otimes c = a \otimes (b \otimes c)$

$\forall a \in S : 1 \otimes a = a \otimes 1 = a$

Second attempt

Pretty good, but can we do better?

Iteratees

Scala machines (https://github.com/runarorama/scala-machines) Gave similar performance on a by-line basis. Thought, three times faster if we provide a Process to ssplit it by words and then monoidally merge single-element Maps.

Iteratees – same as Stream

```
def wordFreq(path: String) =
  getFileLines(new File(path),
    id outmap wordCount) execute
```

Iteratees – 3x faster

```
def splitWords(text: String): List[String] =
  text.filter(acceptedChars)
      .toLowerCase.split("\\W").toList

val words: Process[String, String] = (for {
  s ← await[String]
  _ ← traversePlan_(splitWords(s))(emit)
} yield ()) repeatedly

def wordCount(path: String) =
  getFileLines(new File(path),
    (id split words) outmap (
      _.fold(l ⇒ (1, Map.empty[String, Int]),
             w ⇒ (0, Map(w → -1))))) execute
```

Wordcounting software

Scoobi	http://nicta.github.com/scoobi/
Spark	http://spark-project.org/
Scalding	https://github.com/twitter/scalding/wiki/Type-safe-api-reference

Wordcounting

I did not have time to try to use those. But turns out, this code should work for these "as is".

That's it

Questions?