Advanced Streams

Enumerating the rationals

In this lecture, we discussed two sophisticated uses of streams that are presented in the following article:
http://dl.acm.org/citation.cfm?id=1132893

If you're off-campus, you may not be able to access the article. (It will prompt you to buy it.) You can visit this address when you're connected to *eduroam* or any other on-campus network to get the article for free. We only discussed Sections 1 and 2.

The code from class is included below. It simply translates the code in the article to Scala

```
object Lecture {

  val nats: Stream[BigInt] = 0 #:: nats.map(n => n + 1)

  case class Rat(m: BigInt, n: BigInt) {
    override def toString(): String = s"$m/$n"
  }

  def alternate[A](stream1: Stream[A], stream2: Stream[A]): Stream[A] = (stream1, stream2) match {
    case (x #:: xs, y #:: ys) => x #:: y #:: alternate(xs, ys)
    case (Stream.Empty, _) => stream2
    case (_, Stream.Empty) => stream1
  }

  def diags[A](xss: Stream[Stream[A]], yss: Stream[Stream[A]]): Stream[A] = {
    if (yss.isEmpty) {
      xss.map(_.head) #::: diags(xss.map(_.tail), yss)
    }
    else {
      xss.map(_.head) #::: diags(yss.head #:: xss.map(_.tail), yss.tail)
    }
  }

  val rats1 = diags(Stream(), nats.map(m => nats.map(n => Rat(m, n))))

  def gcd(m: Int, n: Int): Int = {
    if (m < n) gcd(m, n - m)
    else if (m > n) gcd(m - n, n)
    else m
  }

  def igcd(m: Int, n: Int): (Int, List[Boolean]) = {
    if (m < n) {
      val (r, trace) = igcd(m, n - m)
      (r, false :: trace)
    }
    else if (m > n) {
      val (r, trace) = igcd(m - n, n)
      (r, true :: trace)
    }
    else {
      (m, Nil)
    }
  }

  def ungcd(r: Int, trace: List[Boolean]): (Int, Int) = trace match {
    case Nil => (r, r)
    case false :: rest => {
      val (m, n) = ungcd(r, rest)
      (m, n + m)
    }
```

```
    case true :: rest => {
      val (m, n) = ungcd(r, rest)
      (m + n, n)
    }
  }

  val boolseqs: Stream[List[Boolean]] =
    List() #:: boolseqs.map(bs => Stream(true :: bs, false :: bs)).flatten

  val rats3 = boolseqs.map(bits => {
    val (m, n) = ungcd(1, bits)
    Rat(m, n)
  })
}
```