Aflevering 12

Jens Kristian R. Nielsen, 201303862 Thomas D. Vinther , 201303874

29. april 2019

Opgave 121

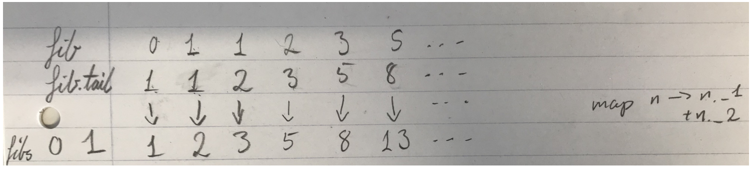
```
object Exercise 121 {

  sealed abstract class List[+T]
  case object Nil extends List[Nothing]
  case class Cons[T](x: T, xs: List[T]) extends List[T]

  abstract class Comparable[T] {
    /** Returns <0 if this<that, ==0 if this==that, and >0 if this>that */
    def compareTo(that: T): Int
  }

  class Student(val id: Int) extends Comparable[Student] {
    def compareTo(that: Student) = this.id - that.id
  }

  def mergeSort[T <: Comparable [T]](xs: List[T]): List[T] = {
    val n = length(xs) / 2
    if (n == 0) xs
    else {
      val (left, right) = split(xs, n)
      merge(mergeSort(left), mergeSort(right))
    }
  }
  def merge[T <: Comparable[T]](xs: List[T], ys: List[T]): List[T] = mergee(xs,ys,Nil)
  def mergee[T<:Comparable[T]](xs: List[T], ys: List[T], ass: List[T]):
  List[T] = (xs,ys) match{
    case (Nil,Nil) => reverse(ass)
    case (Nil,Cons(z,zs)) => mergee(xs,zs,Cons(z,ass))
    case (Cons(z,zs),Nil) => mergee(zs,ys,Cons(z,ass))
    case (Cons(z,zs),Cons(w,ws)) =>
      if(z.compareTo(w)<=0) mergee(zs,ys,Cons(z,ass))
      else mergee(xs,ws,Cons(w,ass))
  }
  def reverse[T](xs: List[T]): List[T] = xs match {
    case Nil => Nil
    case Cons(x, ys) => append(reverse(ys), x)
  }
  def append[T](xs: List[T], x: T): List[T] = xs match {
    case Nil => Cons(x, Nil)
    case Cons(y, ys) => Cons(y, append(ys, x))
  }
  def length[T](xs: List[T]): Int = xs match {
    case Nil => 0
    case Cons(_, ys) => 1 + length(ys)
  }
  // ...
}
```

Opgave 122

```
object Exercise122 {

  def main(args: Array[String]) = {

    // ...

        def filter(p: T => Boolean): Stream[T] = this match {
          case SNil => SNil
          case SCons(y, ys) =>
            //val f = () => if (p(y)) y else Nil
            //SCons(() => f,() => ys().filter(p))
            if(p(y())) SCons(y,() => ys().filter(p)) else ys().filter(p)
        }

      def zip[U](ys: Stream[U]): Stream[(T, U)] = (this,ys) match{
        case (SNil,SNil) => SNil
        case (SNil,SCons(_,_)) => SNil
        case (SCons(_,_),SNil) => SNil
        case (SCons(x,xs),SCons(z,zs)) => SCons(() => (x(),z()),() => xs().zip(zs()))
      }

    // ...
  }

}
```

`primes` virker ved at vi ved to .tail() kald smider 0 og 1 væk, dernæst kalder vi sieve som siger at det forreste tal (head) er et primtal, og derefter filtrerer alle tal som .head deler uden rest væk og fortsætter såfremdeles igennem de naturlige tal, en ineffektiv men korrekt måde at bestemme alle primtal på. Det største primtal vi kune finde var: 100391 inden vi fik en StackOverflowError

`fibs2` virker ved



fib 0 1 1 2 3 5 ...
fib.tail 1 1 2 3 5 8 ...
↓ ↓ ↓ ↓ ↓ ↓ ... map n -> n._1 + n._2
fibs 0 1 1 2 3 5 8 13 ...

Opgave 127

```
object Exercise 127 {

  // ...

  def unfoldRight[A, S](z: S, f: S => Option[(A, S)]): Stream[A] = f(z) match {
    case Some((h, s)) => SCons(() => h, () => unfoldRight(s, f))
    case None => SNil
  }

  val onez = unfoldRight(1,(x: Int)=>Option(1,1))

  val natz = unfoldRight(0,(x:Int)=> Option(x,x+1))

  val fibz = unfoldRight[Int,(Int,Int)]((0,1),x => Option((x._1,(x._2,x._1+x._2))))

}
```