Aflevering 5

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

4. februar 2019

Opgave 51

Proposition: For alle $zs \in \mathsf{IntList}$ og $z \in \mathsf{Int}$:

$$\mathsf{reverse}(\mathsf{append}(zs, z)) = \mathsf{Cons}(z, \mathsf{reverse}(zs))$$

Beviset går ved struktural induktion i listen zs

Basis: $zs = Nil$

$$\begin{align*}
\mathsf{reverse}(\mathsf{append}(zs, z)) &= \mathsf{reverse}(\mathsf{append}(\mathsf{Nil}, z)) && \text{(I. basis)}\\
&= \mathsf{reverse}(\mathsf{Cons}(z, \mathsf{Nil})) && \text{(append case 0)}\\
&= \mathsf{append}(\mathsf{reverse}(\mathsf{Nil}), z)) && \text{(reverse case 1)}\\
&= \mathsf{append}(Nil, z) && \text{(reverse case 0)}\\
&= \mathsf{Cons}(z, \mathsf{Nil}) && \text{(append case 0)} \tag{1}\\
\mathsf{Cons}(z, \mathsf{reverse}(zs)) &= \mathsf{Cons}(z, \mathsf{reverse}(Nil)) && \text{(I. basis)}\\
&= \mathsf{Cons}(z, Nil) && \text{(reverse case 0)}\\
&= \mathsf{reverse}(\mathsf{append}(zs, z)) && \text{(pr (1))}
\end{align*}$$

Som ønsket. **Induktionsskridt:** lad intlisten $xs = \mathsf{Cons}(z, zs) \in \mathsf{IntList}$ og inten $x \in \mathsf{Int}$ være givet. Så induktionshypotesen bliver at propositionen holder for zs altså at $\forall y \in \mathsf{Int} : \mathsf{reverse}(\mathsf{append}(zs, y)) = \mathsf{Cons}(y, \mathsf{reverse}(zs))$.
Vi ønsker at vise at $\mathsf{reverse}(\mathsf{append}(xs, x)) = \mathsf{Cons}(x, \mathsf{reverse}(xs))$

$$\begin{align*}
\mathsf{reverse}(\mathsf{append}(xs, x)) &= \mathsf{reverse}(\mathsf{append}(\mathsf{Cons}(z, zs), x)) && \text{(xs "dekomposition")}\\
&= \mathsf{reverse}(\mathsf{Cons}(z, \mathsf{append}(zs, x)) && \text{(append case 1)}\\
&= \mathsf{append}(\mathsf{reverse}(\mathsf{append}(zs, x)), z) && \text{(reverse case 1)}\\
&= \mathsf{append}(\mathsf{Cons}(x, \mathsf{reverse}(zs)), z) && \text{(I.H. på zs)}\\
&= \mathsf{Cons}(x, \mathsf{append}(\mathsf{reverse}(zs), z)) && \text{(append)}\\
&= \mathsf{Cons}(x, \mathsf{reverse}(xs)) && \text{(definition af reverse på xs)}
\end{align*}$$

Opgave 56

```
  def merge(xs: IntList, ys: IntList): IntList = mergee(xs,ys,Nil)

  def mergee(xs: IntList, ys: IntList, ass: IntList):
  IntList = (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<w) mergee(zs,ys,Cons(z,ass))
          else mergee(xs,ws,Cons(w,ass))
}
```

```
def split(xs: IntList, n: Int): (IntList, IntList) =
  if( length(xs)<= n)
   (Nil,xs)
  else if (n<0)
    throw new RuntimeException("Illegal index")
  else
  splitt(xs,n,Nil)

def splitt(xs: IntList,n: Int, ass: IntList) : (IntList,IntList) = xs match{

  case Cons(z,zs) => if (n>0) splitt(zs,n-1,Cons(z,ass)) else (reverse(ass),xs)

 }


def ordered(xs: IntList): Boolean = xs match {
  case Nil => true
  case Cons(x,Nil) => true
  case Cons(x,Cons(y,ys)) => if (x<=y) ordered(ys) else false
 }

def randomIntList(): IntList = randomIntListt(Nil,new Random().nextInt(101))

def randomIntListt(ass : IntList,n:Int): IntList = {
  if(n>0)
  randomIntListt(Cons(new Random().nextInt(),ass),n-1)
  else
   ass
 }

def permuted(xs: IntList, ys: IntList): Boolean =
  if (length(xs) == length(ys)) listChecker(xs, ys) //a necessary condition
  else false
 def boringMerge(xs: IntList,ys: IntList): IntList = xs match{
  case Nil => ys //we are done
  case Cons(z,zs) => boringMerge(zs,Cons(z,ys)) //merges xs and ys with no regard for sequence
 }
 def elementChecker(x: Int, ys: IntList, ass: IntList):
 (Boolean,IntList,IntList) = ys match{
  case Nil => (false,ys,ass) //x was not found in ys
  case Cons(z,zs) =>
   if (x==z) (true,zs,ass) //x was found in ys, return elements after and before x
   else elementChecker(x,zs,Cons(z,ass)) //looks at next value in ys, with z added to accumulator
 }
 def listChecker(xs: IntList,ys: IntList): Boolean = xs match{
  case Cons(x,zs) =>
   val eC = elementChecker(x,ys,Nil) //checks if x is in y
   if(eC._1) listChecker(zs,boringMerge(eC._2,eC._3)) //continues without x in xs and x in ys
   else false
  case Nil => true //xs is empty, and since the length of xs and ys are the same and we remove 1 element from each, ys is empty too
 }

def testMergeSort(): Unit =  testMergeSortHelp(100)
def testMergeSortHelp(n: Int): Unit ={
  if (n > 0){
   val x = randomIntList()
   val y = mergeSort(x)
   assert(ordered(y))
```

```scala
      assert(permuted(x,y))
      testMergeSortHelp(n - 1)
    }
  }

def mergeSort(xs: IntList): IntList = {
    val n = length(xs) / 2
    if (n == 0) xs
    else {
      val (left, right) = split(xs, n)
      merge(mergeSort(left), mergeSort(right))
    }
  }



/**
* Helping functions
*/

def reverse(xs: IntList): IntList = xs match {
    case Nil => Nil
    case Cons(x, ys) => append(reverse(ys), x)
  }


def append(xs: IntList, x: Int): IntList = xs match {
    case Nil => Cons(x, Nil)
    case Cons(y, ys) => Cons(y, append(ys, x))
}

def length(xs: IntList): Int = xs match {
    case Nil => 0
    case Cons(_, ys) => 1 + length(ys)
  }
```