Recitation 4

Higher-Order Functions

Scala Built-Ins:

map [A] (f : (A) => B) : List[B]
foldLeft [A] (z : B) (f : (B, A) => B) : B
filter [A] (p : (A) => Boolean) : List[A]

1. Write a non-recursive 1-2 line function `sevens (l : List[Int]) : List[Int]` that returns a list containing all of the integers in `l` with the value 7.

Solution:
```scala
def sevens(l : List[Int]) : List[Int] =
  l.filter((x : Int) => x == 7)
```

2. Write a non-recursive 1-2 line function `squares (l : List[Int]) : List[Int]` that squares each of the integers in `l`.

Solution:
```scala
def squares(l : List[Int]) : List[Int] =
  l.map((x : Int) => x * x)
```

3. Write a non-recursive 1-2 line function `odd (l : List[Int]) : List[Int]` that returns a list containing the odd integers in `l`.

Solution:
```scala
def odd(l : List[Int]) : List[Int] =
  l.filter((x : Int) => (x % 2) == 0)
```

4. Write a non-recursive 1-2 line function `sum (l : List[Int]) : Int` that evaluates to the sum of the integers in `l`.

1
5. Write a non-recursive 1-2 line function `hasThree (l : List[Int]) : Boolean` that evaluates to true if and only if `l` contains the integer 3 at least once.

Solution:

```scala
def hasThree(l : List[Int]) : Boolean =
  l.foldLeft (false) ((acc : Boolean, x : Int) => x == 3)
```

6. Write a non-recursive 1-2 line function `isBig (l : List[Int]) : List[Boolean]` that returns a list replacing each integer in `l` with true if the integer is larger than 10 and false otherwise.

Solution:

```scala
def isBig(l : List[Int]) : List[Boolean] =
  l.map ((x : Int) => x > 10)
```

**Structural Induction**

Consider the following function:

```scala
def append[T](xs: List[T], ys: List[T]): List[T] = xs match {
  case Nil => ys
  case x::xs => x::append(xs, ys)
}
```

We claim the `append` is associative. State this claim formally and prove it using structural induction.

Solution: *Theorem:*
For all lists $\ell_0$, $\ell_1$, $\ell_2$, if

$\text{append} (\ell_0, \text{append} (\ell_1, \ell_2)) \rightarrow^* v$ and $\text{append} (\ell_0, \ell_1, \ell_2) \rightarrow^* v'$,
then \( v = v' \).

**Proof:** By induction on \( \ell_0 \).

**Base Case:** \( \ell_0 = \text{nil} \). Then

\[
\text{append}(\ell_0, \text{append}(\ell_1, \ell_2)) \\
\rightarrow^* \text{append}(\text{nil}, \ell_1::\ell_2) \\
\rightarrow^* \text{nil}::\ell_1::\ell_2 \\
\rightarrow^* \ell_1::\ell_2 \\
= v,
\]

and

\[
\text{append}(\text{append}(\ell_0, \ell_1), \ell_2) \\
\rightarrow^* \text{append}(\text{nil}::\ell_1, \ell_2) \\
\rightarrow^* \text{append}(\ell_1, \ell_2) \\
\rightarrow^* \ell_1::\ell_2 \\
= v', \text{ so } v = v' = \ell_1::\ell_2, \text{ as required.}
\]

**Inductive Hypothesis:** Let \( \ell_0 = e :: l_a \) (where \( e \) is any element). Assume that for all lists \( l_a, l_1, l_2 \), if \( \text{append}(\text{append}(l_a, l_1), l_2) \rightarrow^* v_a \) and \( \text{append}(l_a, \text{append}(l_1, l_2)) \rightarrow^* v'_a \), then \( v_a = v'_a \).

We will show that the theorem holds for \( \ell_0 \).

We have \( \text{append}(\text{append}(\ell_0, \ell_1), \ell_2) \)

\[
\rightarrow^* \text{append}(\text{append}(e::l_a, l_1), l_2) \text{ (substitution for } \ell_0) \\
\rightarrow^* \text{append}(e::\text{append}(l_a, l_1), l_2) \text{ (definition of append)} \\
\rightarrow^* e::\text{append}(l_a, \ell_1), l_2) \text{ (definition of append)} \\
\rightarrow^* e::v_a \text{ (by IH pt. 1)}
\]

and

\[
\text{append}(\text{append}(\ell_0, \ell_1, l_2)) \\
\rightarrow^* \text{append}(e::l_a, \text{append}(\ell_1, \ell_2)) \text{ (substitution for } \ell_0) \\
\rightarrow^* e::\text{append}(l_a, \text{append}(\ell_1, \ell_2)) \text{ (definition of append)} \\
\rightarrow^* e::v'_a \text{ (by IH pt. 2)}
\]

Since \( v_a = v'_a \), we have \( e::v_a = e::v'_a \) and so the theorem holds.