Deklarative Programmierung, 1. Teilklausur (Gruppe A), 12.04.2024

[Egiraffe-Angabe trw4711]

1. **Tail-Rekursion:** Gegeben sei folgende rekursive Funktion.

   ```
   def pihalf(n: Int): Double = n match {
     case 0 => 1
     case _ => 4*n*n / (4*n*n - 1.0) * pihalf(n - 1)
   }
   ```

   Adaptieren Sie die Implementierung so, dass Ihre adaptierte Variante tail-rekursiv ist und somit effizient ausgeführt werden kann.

   Lösung:

   ```
   def pihalfTail(n: Int, number: Double): Double = n match {
     case 0 => number
     case _ => pihalfTail(n - 1, 4*n*n / (4*n*n - 1.0) * number)
   }
   print(pihalfTail(20, 1))
   ```

2. **Funktionen höherer Ordnung:** Definieren Sie die Funktion validOp(a: Int, b: Int, op: (Int, Int) => Int, check: Int => Boolean): Int, welche das Ergebnis der Operation op von a und b nur dann zurückgeben soll, wenn sowohl für a als auch b check wahr ist. Ansonsten sollte a zurückgegeben werden.

   Ändern Sie außerdem die Funktionssignatur bzw. Reihenfolge der Argumente so, dass zuerst op und anschließend check jeweils curried übergeben werden. Zuletzt sollen a und b gemeinsam übergeben werden.

   Lösung:

```
def validOp(op: (Int, Int) => Int)(check: Int => Boolean)(a: Int, b: Int): Int = {
  if (check(a) && check(b)) op(a, b)
  else a
}
```

   z.B.
   print(validOp((x, y) => x+y)(x => x % 2 == 0)(1,2))
   => Output: 1

   print(validOp((x, y) => x+y)(x => x % 2 == 0)(4,2))
   => Output: 6

3. **Listen-Funktionen höherer Ordnung:** Definieren Sie die folgende tail-rekursive Funktion (also ohne Verwendung von fold oder anderen Listen-Funktionen) rangeIf(start, end, pred), die eine Liste mit allen Ganzzahlen zwischen den Integer-Parametern start und end erstellt, für welche die einstellige (unäre) Boolean-Funktion pred wahr ist. Die untere Grenze start soll in der Ergebnis-Liste inkludiert sein, falls für diese pred wahr ist, während die obere Grenze end nicht enthalten sein soll.

   Schreiben Sie in der Definition auch alle benötigten Datentypen explizit an. Sie dürfen davon ausgehen, dass die Funktion nur mit start <= end aufgerufen wird.

   Lösung:

```
def rangeIf(start: Int, end: Int, pred: Int => Boolean): List[Int] = {
  def rangeIfTail(start: Int, end: Int, pred: Int => Boolean, result: List[Int]):
  List[Int] = {
    if (start > end) result
    else if (start < end && pred(start))
          rangeIfTail(start + 1, end, pred, result ++ List(start))
    else rangeIfTail(start + 1, end, pred, result)
  }
  rangeIfTail(start, end, pred, List())
}
```

   z.B.
   print(rangeIf(-2, 4, x => x % 2 == 0))
   => Output: List(-2, 0, 2)

4. **Fold-Map-Filter:** Gegeben ist die Integer-Liste range mit den Zahlen von start bis inklusive end. Rechnen Sie die Summe aller Zahlen von start bis exklusive end aus, für die pred wahr sind.

   Starten Sie hierfür mit der unten gegebenen inklusiven Liste range und wenden Sie auf diese ausschließlich die Funktionen foldLeft, map und/oder filter an. Dabei dürfen foldLeft, map und/oder filter auch mehrmals hintereinander verwendet werden bzw. müssen nicht alle davon verwendet werden.

   ```
   // given: start, end, pred
   val range: List[Int] = (start to end).toList // inclusive
   ```

   Lösung:

   ```
   def foldMapFilter(start: Int, end: Int, pred: Int => Boolean): Int = {
     val range: List[Int] = (start to end).toList // inclusive
     range.filter(_ != end).foldLeft(0)((acc, head) =>
       if (pred(head)) acc + head else acc)
   }
   ```

   z.B.
   ```
   print(foldMapFilter(1, 6, x => x % 2 == 0))
   ```
   => Output: 6