Till Mossakowski
Stephan Günther

5. Übungsblatt

Ausgabe: 2019-11-16
Abgabe: 2019-11-20

5.1 * *Run-length encoding*

(1) Implementieren Sie eine Funktion

```
rle :: Eq a => [a] -> [(Int, a)]
```

die ihr Argument via Run-length encoding codiert. Die Funktion `rle` erstellt also aus einer Liste eine Liste von Paaren, so dass gilt:

```
l == concatMap (\(n, x) -> replicate n x) (rle l).
```

Anwendungsbeispiele:

```
rle [1, 2, 2, 3, 3, 3] = [(1,1),(2,2),(3,3)]
rle [2, 2, 2, 1, 1, 3] = [(3,2),(2,1),(1,3)]
```

(2) Implementieren Sie eine Funktion, die von einem gegebenen Startwert die passende "Look-and-say sequence" als unendliche Liste erzeugt. In der "Look-and-say sequence" wird der Nachfolger eines gegebenen Werts x bestimmt, in dem die Bestandteile von x gezählt werden. Die Resultate dieser Zählung bilden, jeweils gepaart mit den Originalwerten, hintereinandergeschrieben das nächste Element in der Sequenz. Zur Illustration sind im folgenden jeweils die drei nächsten Elemente der Sequenz zu einem gegebenen Startwert aufgeführt:

```
123 ⇝ 111213, 31121113, 1321123113
 22 ⇝ 22, 22, 22
  4 ⇝ 14, 1114, 3114
 99 ⇝ 29, 1219, 11121119
321 ⇝ 131211, 1113111221, 3113312211
```

Ziel der folgenden Aufgaben ist ein gründliches Verständnis von `map`, `filter`, `foldr`/`foldl`, sowie η-Kontraktion, partieller Anwendung und Funktionen höherer Ordnung im Allgemeinen. Daher ist die Verwendung von *rekursiv definierten Funktionen* zur Lösung der ab hier folgenden Aufgaben *verboten*.

5.2 *map, foldl, foldr, filter*

Schreiben Sie folgende Funktionen mithilfe der Funktionen `map`, `filter`, `foldr` und `foldl`.

(1) Berechnet das Produkt einer Liste von ganzen Zahlen:

```
product' :: [Int]→ Int
```

Beispiel:

```
product' [1 .. 5] ⇝ 120
```

(2) Testet ob keins der Listenelemente gerade ist.

```
noneIsEven :: [Int] → Bool
```

Beispiel:

```
noneIsEven [5,7] ⇝ True
noneIsEven [4]   ⇝ False
```

(3) Berechnet die Summe der Längen der Zeichenketten.

```
sumLengths :: [String] → Int
```

Beispiel:

```
sumLengths ["Hallo", "DU!"] ⇝ 8
```

(4) Berechnet eine Wertetabelle der Funktion $x^2 + 3x + 5$ für die Argumente 0 bis 150.

```
xSquaredPlusThreeXPlusFive :: [(Integer, Integer)]
```

Beispiel:

```
xSquaredPlusThreeXPlusFive !! 1 ⇝ (1, 10)
```

(5) Gibt alle Zahlenwerte zurück, bei denen der zugeordnete String gleich dem zweiten Parameter ist.

```
getByKey :: [(String, Int)] → String → [Int]
```

Beispiel:

```
getByKey [("a",2), ("b",3), ("a",6), ("c",4)] "a" ⇝ [2,6]
```

5.3 *Andere Funktionen in* `base`

Schauen Sie sich für diese Aufgabe zuerst die Funktionen aus Data.List an zum Suchen und Erstellen von Listen und implementieren Sie anschließend auf möglichst geschickte Weise die nachfolgenden Funktionen.

(1) Berechnet die Liste der Partialsummen.

```
partialSums :: Num a ⇒ [a] → [a]
```

Beispiel:

```
partialSums [1,2,3,4] ⇝ [0,1,3,6,10]
```

(2) Füllt die Strings am Ende mit Leerzeichen auf, so dass alle Strings die gleiche Länge haben. Die Länge aller Strings soll die Länge des längsten Strings in der Eingabeliste sein. Sie können annehmen, dass die alle endlich sind.

```
toSameLength :: [String] → [String]
```

Beispiel:

```
toSameLength ["Christoph", "Lueth"] ⇝ ["Christoph", "Lueth␣␣␣␣"]
```

(3) Sortiert die inneren Listen nach ihrer Länge in aufsteigender Reihenfolge.

```
sortByLength :: [[a]] → [[a]]
```

Beispiel:

```
sortByLength [[3,4,5], [11], [1]] ⇝ [[11], [1], [3,4,5]]
```

(4) Wie `sortByLength`, allerdings wird zusätzlich der Index zurückgegeben, den die jeweilige Liste vor dem Sortieren hatte.

```
sortByLengthRememberingIndex :: [[a]] → [(Int ,[a])]
```

Beispiel:

```
sortByLengthRememberingIndex ["abc", "p", "xx"] ⇝ [(1, "p"), (2, "xx"), (0, "abc")]
```

(5) Wendet die erste Funktion auf die inneren Listen in einer Liste von Listen an, und die zweite Funktion auf jede Ergebnisliste.

```
innerOuterMap :: (a → b) → ([b] → c) → [[a]] → [c]
```

Beispiel:

```
innerOuterMap (∗3) sum [[1..4], [5,6]] ⇝ [30,33]
```