Universität Bremen

Praktische Informatik 3 WS 18/19

Christoph Lüth
Thomas Barkowsky
Andreas Kästner
Gerrit Marquardt
Tobias Haslop
Matz Habermann
Berthold Hoffmann

6. Übungsblatt

Ausgabe: 20.11.18
Abgabe: 27.11.18 12:00

Ziel dieses Übungsblattes ist ein gründliches Verständnis von `map`, `filter`, `foldr/foldl`, sowie darüber hinaus η-Kontraktion, partielle Anwendung und Funktionen höherer Ordnung im Allgemeinen. Deshalb geht es in diesem Übungsblatt mehr als bei den vorherigen nicht nur darum, dass die Tests durchlaufen, sondern dass Sie die Konzepte dahinter verstehen.

Das bedeutet bei der Bewertung wird ein besonderes Augenmerk darauf gelegt, ob Sie die Funktionen höherer Ordnung auf eine möglichst gute Weise verwenden. In diesem Übungsblatt ist daher die Verwendung von *rekursiv definierten Funktionen verboten*. Darüber hinaus müssen Sie für dieses Übungsblatt auch *keine eigenen Tests* schreiben.

6.1 *map, foldl, foldr, filter* — *5 Punkte*

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

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

```
prod :: [Int]→ Int
```

Beispiel:

```
prod [1 .. 5] ⇝ 120
```

(2) Keines der Elemente der Liste ist gerade.

```
noneIsEven :: [Int] → Bool
```

Beispiel:

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

(3) Berechnet die Summe der Längen der Zeichenketten.

```
lengthSum :: [String] → Int
```

Beispiel:

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

(4) Eine Wertetabelle von 0 bis 150 der Funktion $x^2 + 2x + 7$.

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

Beispiel:

```
functionLookupTable !! 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]
```

6.2 *Andere Funktionen in* base *5 Punkte*

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) Gibt die Liste der Partialsummen zurück.

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

Beispiel:

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

(2) Füllt die Strings am Ende mit Freizeichen auf, so dass alle Strings die gleiche Länge haben; die Länge aller Strings soll die Länge des längsten Strings in der Eingabeliste sein. (Die Strings sollen alle endlich sein.)

```
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 zweite Funktion auf die inneren Listen in einer Liste von Listen an, und die erste Funktion auf jede Ergebnisliste.

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

Beispiel:

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