Universität Bremen

Praktische Informatik 3 WS 16/17

Christoph Lüth
Tobias Brandt
Tristan Bruns
Johannes Ganser
Berthold Hoffmann
Alexander Kurth

4. Übungsblatt

Ausgabe: 15.11.16
Abgabe: 25.11.16, 12:00 Uhr

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. Nehmen Sie sich also Zeit, helfen Sie sich in der Gruppe, gruppenübergreifend oder im Tutorium.

Die Erkenntnisse aus diesem Blatt werden in allen folgenden Übungsblättern vorausgesetzt; Verständnislücken in diesem Blatt werden sich in den folgenden Blättern fortsetzen.

4.1 *Refactoring: Aus Alt mach Neu.* *10 Punkte*

Im ersten und zweiten Übungsblatt wurden verschiedene Funktionen durch strukturelle Rekursion implementiert. Das können Sie nun natürlich viel eleganter und leichter verständlich mit Fold und seinen Freunden. Reimplementieren Sie bitte die folgenden Funktionen unter Berücksichtigung der neu gelernten Konstrukte:

```
stripVowels   :: String → String          — Blatt 1
encrypt       :: String → String          — Blatt 1
numWords      :: Line → Int               — Blatt 2
len           :: Line → Int               — Blatt 2
app           :: Line → String → Line     — Blatt 2
lineToString  :: Line → String            — Blatt 2
```

In der Datei `Refactor.hs` liegen die alten, manuellen Implementierungen aller Funktionen als Referenz. Fügen Sie die Neuimplementierungen dort an. Da es natürlich keine zwei Funktionen mit dem selben Namen geben kann, erweitern Sie den Namen jeder Funktion um einen Apostroph (d.h. `foo'` für `foo`).

Da zwischenzeitlich auch Polymorphie und Listen eingeführt wurden, wurden die alten Implementierungen bereits so angepasst, dass sie auf Listen arbeiten— da geht das Mappen, Filtern und Falten gleich viel einfacher.

4.2 *Map und Filter sind auch nur Folds* *4 Punkte*

Implementieren Sie zwei Funktionen `map'` und `filter'`, die sich exakt wie `map` bzw. `filter` verhalten, jeweils mithilfe einer Fold-Funktion (in der Datei `Fold.hs`)

Dokumentieren Sie schrittweise die Auswertung des folgenden Ausdrucks (mit Ihrer Definition von `map'`):

```
map' (2+) [1, 5, 8]
```

4.3 *Buchhaltung* *6 Punkte*

Für eine Bankensoftware sollen einige Funktionen entworfen werden, die Listen von Buchungen auf Girokonten verarbeiten. Eine Buchung wird repräsentiert durch den Datenypen

```
data Buchung = Einzahlung Integer | Auszahlung Integer
```

Der gebuchte Betrag ist jeweils vom Typ `Integer` und repräsentiert Eurocents.

Implementieren Sie die folgenden Funktionen in der Datei `Buchung.hs`, indem Sie Funktionen höherer Ordnung wo immer möglich benutzen:

```
flatten     :: [Buchung] → [Integer]
kontostand  :: Integer → [Buchung] → Integer
abhebungen  :: [Buchung] → Integer
einzahlungen :: [Buchung] → Integer
```

`flatten` wandelt eine Liste von Buchungen in eine andere Repräsentation um: Eine Liste von Integern, die Eurocentbeträge repräsentieren, wobei negative Beträge für Abhebungen und positive für Einzahlungen stehen.

`kontostand` erhält den ursprünglichen Kontostand und gibt den Kontostand nach Verbuchung aller Buchungen zurück.

`abhebungen` und `einzahlungen` geben die Summe aller Abhebungen bzw. Einzahlungen zurück.

Hinweise: Die folgenden Funktion können hilfreich sein:

```
negate :: Integer→ Integer
sum    :: [Integer]→ Integer
```

? *Verständnisfragen*

1. Was kennzeichnet strukturell rekursive Funktionen, wie wir sie in der Vorlesung kennengelernt haben, und wie sind sie durch die Funktion `foldr` darstellbar?
2. Welche anderen geläufigen Funktionen höherer Ordnung kennen wir?
3. Was ist η-Kontraktion, und warum ist es zulässig?
4. Wann verwendet man `foldr`, wann `foldl`, und unter welchen Bedingungen ist das Ergebnis das gleiche?