Institut für Informatik **Ludwig-Maximilians-Universität München**
Prof. François Bry, Thomas Prokosch
Kilian Rückschloß

Programmierung und Modellierung, SoSe 21
Übungsblatt 7

Abgabe: bis Mo 07.06.2021 08:00 Uhr

Aufgabe 7-1 Funktionen höherer Ordnung

Implementieren Sie folgende Funktionen ohne Rekursion oder List-Comprehensions mittels Funktionen höherer Ordnung aus der Standardbibliothek wie `map`, `filter`, `foldr`, usw.

Tipp: Nutzen Sie zum Testen während der Implementierung die Funktionen `scanr` bzw. `scanl`, um die Zwischenschritte der Fold-Funktionen besser nachvollziehen zu können.

a) Zuerst ein alter Bekannter: Implementieren Sie die Funktion `length'`, die die Länge einer Liste berechnet.

b) Implementieren Sie die Funktion `capitalizedInitials :: String -> String`, die für einen beliebigen String die Anfangsbuchstaben der enthaltenen Wörter als Großbuchstaben zurückgibt.

c) Definieren Sie die Funktion `map'`, die sich gleich der vordefinierten Funktion `map` verhält, mittels `foldr`.

d) Defninieren Sie die Funktion `filter'`, die sich gleich der vordefinierten Funktion `filter` verhält, mittels `foldr`.

e) Definieren Sie die Funktion `takeWhile' :: (a -> Bool) -> [a] -> [a]`, die die ersten Elemente einer Liste in eine neue Liste zurückgibt, bis zum ersten Element, an dem die übergebene Prädikatsfunktion falsch ist: `takewhile' (\x -> x < 5) [1,4,3,7,3,2]` gibt `[1,4,3]` zurück.

Aufgabe 7-2 Monoide

In der Vorlesung haben Sie kennengelernt, dass man Zahlen auch als Monoide (bzgl. jeweils Addition oder Multiplikation) verstehen kann. Der `Ordering` Typ lässt sich ebenfalls als ein Monoid interpretieren, um auf elegante Art mehrstufige Vergleiche durchführen zu können.

Nehmen wir als Beispiel eine lexikographische Sortierung in umgekehrter Richtung: Personen werden rückwärts alphabetisch nach ihrem Namen sortiert, wobei Z vor Y vor A kommt, d.h. "Zebra" steht vor "Able". Nur in dem Fall, dass ein Name identisch mit dem Beginn eines anderen Namens ist, z.B. "Roth" und "Rothman", wird der kürzere Name zuerst geführt (hier also Roth).

a) Implementieren Sie die Typklasse `Monoid` für den Typ `Ordering`.
Stellen Sie sich bei der Bearbeitung folgende Fragen:

- Welches der drei Werte von `Ordering`, `LT, EQ, GT` ist das neutrale Element und warum?
- Wie muss sich die `mappend` Funktion verhalten, um das Ergebnis der "vorangehenden" Vergleiche in den anderen Fällen zu erhalten?
(D.h. das Ergebnis späterer Vergleiche, wie der Vergleich nach der Länge des Namens bei "Miller" und "Müller", als irrelevant zu ignorieren, da bereits der zweite Buchstabe unterschiedlich ist)
- Welche Eigenschaft(en) muss die `mappend` Funktion auf jeden Fall erfüllen, damit die Monoid-Eigenschaft erfüllt ist?

Hinweis: Die Typklasse `Monoid` ist für `Ordering` bereits standardmäßig implementiert, d.h. Sie müssen diese Teilaufgabe "freihand" ohne Evaluation durch ghc implementieren (Sie erhalten sonst eine Fehlermeldung "error: Duplicate instance declarations"). In der nächsten Teilaufgaben verwenden wir dann die Standardimplementierung.

b) Ab hier arbeiten wir mit dem vordefinierten Typ `Ordering`. Der oben im Text skizzierte Vergleich lässt sich ohne Verwendung der Monoid-Eigenschaft folgendermaßen implementieren:

```
revCompare :: String -> String -> Ordering
revCompare []     []     = EQ
revCompare []     (_:_)  = LT
revCompare (_:_)  []     = GT
revCompare (x:xs) (y:ys) = case flip compare x y of
                                EQ    -> revCompare xs ys
                                other -> other
```

Schreiben Sie die Funktion so um, dass Sie sich bei `revCompare (x:xs) (y:ys)` die Monoid-Eigenschaft von `Ordering` aus Aufgabe a) zunutze machen.

Aufgabe 7-3 Arithmetische Operationen ohne die Standardbibliothek

Gegeben sind die folgenden Definitionen `succ'` und `pred'`, die jeweils den direkten Nachfolger, bzw. direkten Vorgänger eines ganzzahligen Werts bestimmen.

```
succ' :: Integer -> Integer
succ' x = x + 1

pred' :: Integer -> Integer
pred' x = x - 1
```

Implementieren Sie ausschließlich unter Benutzung der gegebenen Funktionen die folgenden Funktionen.

a) Implementieren Sie eine Funktion

   `plus :: Integer -> Integer -> Integer`

   die zwei `Integer`-Werte addiert, sowie eine Funktion

   `minus :: Integer -> Integer -> Integer`

   die den zweiten Wert vom ersten abzieht. Benutzen Sie dabei keine Funktionen der Standardbibliothek.

b) Implementieren Sie jetzt aufbauend auf Ihrem Ergebnis von Teilaufgabe a) eine Funktion `mult :: Integer -> Integer -> Integer`, die zwei `Integer`-Werte miteinander multipliziert.

c) Implementieren Sie jetzt aufbauend auf den Ergebnissen der vorherigen Teilaufgaben einen Funktion `mod' :: Integer -> Integer -> Integer`, welche den Rest bei der Ganzzahldivison der ersten durch die zweiten Zahl zurückgibt.

Aufgabe 7-4 Wiederholung: Abstiegsfunktion

Gegeben sei die folgende Implementierung einer Funktion, welche die Länge einer übergebenen Liste bestimmt.

```
length' []     = 0
length' (x:xs) = 1 + length' xs
```

Zeigen Sie mit Hilfe einer Abstiegsfunktion, dass die Funktion `length'` terminiert.