9. Aufgabenblatt zur Vorlesung

Informatik A

WiSe 2015/16

Wolfgang Mulzer

Abgabe am 5. Januar 2016 bis 12 Uhr in die Tutorenfächer

Alle Aufgaben auf diesem Zettel sind Zusatzaufgaben. Frohe Weihnachten!

Aufgabe 1 Falten *10 Zusatzpunkte*

Implementieren Sie die folgenden Funktionen mit Hilfe von Lambda-Ausdrücken und der Funktion `falte/foldr` (VL 15).

Hinweis: Man kann `if...then...else...` in Lambda-Ausdrücken verwenden.

(a) Gegeben eine Liste, berechne ihre Länge.

```
laenge:: [a] -> Int
```

(b) `wendeAn/map` aus der Vorlesung.

```
wendeAn:: (a -> b) -> [a] -> [b]
```

(c) `siebe/filter` aus der Vorlesung.

```
siebe:: (a -> Bool) -> [a] -> [a]
```

(d) Sei `f` eine Funktion mit Signatur `f:: a -> Bool` und `xs` eine Liste vom Typ `a`, liefere genau dann `True`, falls `f` für *alle* Elemente in `xs` den Wert `True` ergibt.

```
fuerAlleWahr:: (a -> Bool) -> [a] -> Bool
```

Aufgabe 2 Algebraische Datentypen: Mengen *10 Zusatzpunkte*

In der Vorlesung hatten wir den Datentyp `Liste` definiert. Definieren Sie jetzt einen Datentyp, der eine *Menge* darstellt.

```
data Menge a = Menge [a]
```

(a) Deklarieren Sie den Typ `Menge` als Instanz der Typklasse `Show` und implementieren Sie die Funktion

```
show :: Show a => Menge a -> String
```

Zum Beispiel soll die Ausgabe einer Menge `Menge [1,5,6]` folgendes Format haben: {`1,5,6`}.

(b) Implementieren Sie die Funktion

```
vonListe :: Eq a => [a] -> Menge a
```

die aus einer Liste vom Typ `a` eine Menge erstellt. Dabei sollen Duplikate aus der Liste entfernt werden. Beispielsweise soll `vonListe [1,2,2,3,1]` das Ergebnis `Menge [1,2,3]` zurückgeben. Im Folgenden können Sie davon ausgehen, dass alle Werte des Typs `Menge` durch die Funktion `vonListe` erzeugt wurden.

(c) Deklarieren Sie den Typ `Menge` als Instanz der Typklasse `Eq` und implementieren Sie die Funktion

```
(==) :: Eq a => Menge a -> Menge a -> Bool
```

Dabei soll `Menge xs == Menge ys` genau dann wahr sein, wenn `xs` und `ys` dieselben Elemente enthalten (die Reihenfolge spielt dabei keine Rolle!). Beispielsweise soll `Menge [1,5,2] == Menge [2,1,5]` wahr sein, während `Menge [1,5,2] == Menge [1,5]` falsch ist.

(d) Deklarieren Sie den Typ `Menge` als Instanz der Typklasse `Ord` und implementieren Sie die Funktion

```
(<=) :: Eq a => Menge a -> Menge a -> Bool
```

Dabei soll `Menge xs <= Menge ys` genau dann wahr sein, wenn `xs` eine Teilmenge von `ys` ist (die Reihenfolge spielt wieder keine Rolle).

Aufgabe 3 Produkte der anderen Elemente *10 Zusatzpunkte*

Sei $[a_1, \ldots, a_n]$`::[Int]`. Wir wollen eine Liste $[b_1, \ldots, b_n]$`::[Int]` finden mit

$$\begin{aligned} b_1 &= a_2 \cdot \cdots \cdot a_n \\ &\vdots \\ b_i &= a_1 \cdot \cdots \cdot a_{i-1} \cdot a_{i+1} \cdot \cdots \cdot a_n \\ &\vdots \\ b_n &= a_1 \cdot \cdots \cdot a_{n-1}. \end{aligned}$$

In der Vorlesung hatte wir angefangen, dazu einen Algorithmus zu implementieren, der `scanl` und `scanr` benutzt. Vervollständigen Sie die fehlenden Schritte. Den Code aus der Vorlesung können Sie auf der Veranstaltungswebsite finden.

(a) Implementieren Sie die Funktion

```
produkteRechts:: [Tripel] -> [Tripel]
```

analog zur Funktion `produkteLinks` aus der Vorlesung. Verwenden Sie jedoch `scanr` anstatt `scanl`. Diese ist wie folgt definiert.

```
scanr:: (a->b->b) -> b -> [a] -> [b]
scanr _ x0 []       = [x0]
scanr f x0 (x:xs)   = (f x y) : (y:ys)
                    where (y:ys) = scanr f x0 xs
```

Um eine Intuition für `scanr` zu bekommen, bietet es sich an, die Ausgabe von `scanr (*) 4 [1,2,3]` zu betrachten.

Hinweis: Bei `produkteLinks` hatten wir bei `scanl` für `x0` das erste Element a_1 der Liste gewählt, und für `(x:xs)` den Rest der Liste, also $[a_2, \ldots, a_n]$. Bei `scanr` verhält es sich spiegelverkehrt.

(b) Implementieren Sie die Funktion

```
multLR:: [Tripel] -> [Int],
```

die jeden Tripel `(a,b,c)` in einer Liste auf das Produkt `a*c` abbildet.

(c) Implementieren Sie die Funktion

```
produkte :: [Int] -> [Int],
```

mit Hilfe von `tripel`, `produkteLinks`, `produkteRechts` und `multLR`.

Geben Sie immer alle Signaturen an und machen Sie geeignete Testläufe. Versehen Sie Ihr Skript mit geeigneten Kommentaren.