



Technische Universität Berlin

Software and Embedded Systems Engineering Group

Prof. Dr. Sabine Glesner

www.sese.tu-berlin.de Sekr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin



Softwaretechnik und Programmierparadigmen WiSe 2022/2023

Prof. Dr. Sabine Glesner
Milko Monecke
Simon Schwan

Übungsblatt 03



Aufgabe 1: Listenfunktionale

Verwendet die Funktionen `map`, `filter`, `foldr` oder `foldl`, um die folgenden Aufgaben zu lösen. Der Typ `CommandR` ist aus dem vorangegangenen Tutorium bereits bekannt.

- a) Verwendet ein Listenfunktional um eine Funktion `nvcls :: String -> String` zu implementieren, die aus einem String alle Vokale entfernt.
- b) Wendet Eure `sqr` und `pot` auf alle ganze Zahlen im Intervall $[0, 10]$ an.
- c) Sammelt in der Funktion `listValues :: [CommandR] -> [Int]` aus einer Liste vom Typ `CommandR` alle Blattwerte `ValR` in einer neuen Liste vom Typ `Int` auf.
- d) Erstellt eine Funktion `badluck :: CommandR -> [CommandR] -> Command`, in der von einem Ursprungsstapel eine beliebige Menge Einsätze vom Typ `CommandR` abzieht. Sind `foldr` und `foldl` dafür beide geeignet?
- e) Implementiert ein eigenes Listenfunktional `any :: (a -> Bool) -> [a] -> Bool`, welches `True` ergibt, wenn das gegebene “Prädikat” für mindestens ein Element der Liste gilt. Kann `any` mit anderen Listenfunktionalen implementiert werden? Terminiert das Listenfunktional, wenn es auf unendliche Listen angewandt wird?





Schlüssel:

- Ein ergänzendes Video wird zur Vor- oder Nachbereitung veröffentlicht.
- Wird im Tutorium besprochen.

- f) Implementiert die vordefinierte Funktion `elem :: a -> [a] -> Bool` mithilfe eines Listenfunktional neu. 
- g) Erstellt eine kompaktere Version der `hasVowel`. 




Aufgabe 2: List Comprehensions

Verwendet *List Comprehensions*, um die folgenden Listen zu erstellen:

- a) Die ersten 20 Vielfachen von 7. 
- b) Jede Zahl bis 100, die nicht durch fünf teilbar ist. 
- *) (Zusatz) Alle Primzahlen bis 100. 
- ***) (Zusatz) Kleine Hilfe einen coolen Namen für den Nachwuchs auszusuchen, der auch noch auf den Perso passt. Alle Dreier-Kombinationen ohne Wiederholungen von Vornamen aus der Liste `["Jon", "Theon", "Samwell", "Joffrey", "Bran"]` mit Bindestrich, die zusammen mit dem Nachnamen Schubert nicht über 26 Zeichen kommen. 

Aufgabe 3: Typklassen

Unser neuer Datentyp `CommandR` ist in der Handhabung noch unbequem. Er soll einige Typklassen instanziiieren, damit er vielseitig eingesetzt werden kann.



- a) Der Haskell Interpreter weiß nicht, wie er den neuen Datentyp ausgeben soll, wenn man ihn nicht explizit in einen String umwandelt. Diskutiert verschiedene Wege, dies mit der Typklasse `Show` zu lösen (Überprüft ggf. die `Show`-Typklasse mit `:i` bzw. die Signatur der notwendigen `show`-Funktion in `ghci` mit `:t`). 
- b) Damit unsere neuen Grundrechenarten tatsächlich so aussehen, als würde mit normalen Zahlen gerechnet werden, wollen wir die Typklasse `Num` instantiieren. **Wie kann man die Funktionen `(+)`, `(-)` und `(*)` für diesen Typ implementieren?**  

Außerdem sind Definitionen für `abs` und `signum` nötig. Ersteres ist trivial: da wir nur mit natürlichen Zahlen umgehen, gilt `abs c == c` für alle `c :: CommandR`. Signum hängt andererseits vom Wert ab. Zum Beispiel ist `signum (ValR 0) == ValR 0`, aber `signum (ValR 100) == ValR 1`. Deswegen führen wir einen neuen Konstruktor ein:

```
1 data Command = Put Command Command
2               | Take Command Command
3               | Win Command Command
4               | Val Int
5               | Sig Command
6
7 eval :: Command -> Int
8 eval (Sig v)      = signum (eval v)
9 -- ...
10 -- der Rest ist wie evalR aus Blatt 02
```

Es bleibt noch `fromInteger` zu implementieren. Weil eine Instanz `Num Int` existiert, ist die `fromInteger` für `Ints` dabei hilfreich¹. **Was muss dabei noch berücksichtigt werden, damit das Ergebnis immer einen gültigen Chipstapel repräsentiert?**

Instanziiert die Typklassen `Show` und `Num` für den neuen `Command`-Typ und probiert, damit man Terme bei der Konsole wie `2 * 3 + 5 :: Command` einzugeben.

- c) Wir wollen vordefinierte Funktionen verwenden können, die Vergleiche benötigen. Instanziiert die Typklassen `Eq` und `Ord` und überprüft die Implementierung an geeigneten Beispielen. 
- d) Überprüft die Signatur der vordefinierten Funktion `max` mithilfe von `:t`. Was fällt Euch auf? Testet die Funktion an einem geeigneten Beispiel mit `Command`. 

¹Zum Beispiel ergibt `(fromInteger (42 :: Integer) :: Int)` die Zahl `(42 :: Int)`