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 2023/2024

Prof. Dr. Sabine Glesner
Simon Schwan
Julian Klein

Übungsblatt 03

Aufgabe 1: Parametrisierte Datentypen

Listen und Tupel sind prominente Vertreter der parametrisierten Datentypen. Schreibt einige Funktionen mit pattern matching für Listen.

a) Schreibt eine Funktion `rep`, die eine Liste `n`-mal wiederholt.

b) Schreibt eine Funktion `mirror`, die eine Liste am Ende spiegelt (z.B., `[1,2]` wird `[1,2,2,1]`).

c) Schreibt eine Funktion `drop2`, die die ersten *zwei* Elemente einer Liste entfernt und den Rest zurückgibt.

d) In einer Liste vom Typ `CommandR` sind die aktuellen Chips einer Menge von Teilnehmern hinterlegt. Es sollen in der Funktion `kick :: [CommandR] -> [CommandR]` alle Elemente aussortiert werden, die zu 0 evaluieren, also von Teilnehmys die all ihre Chips verspielt haben.

e) Wieder stellt eine Liste von `CommandR` die "Kontostände" von Besuchern dar. Etwas ist in der letzten Runde schiefgegangen, also sollen alle durch die Funktion `payback :: [CommandR] -> [CommandR]` ihr Geld zurückkriegen, die im letzten Schritt Chips eingesetzt haben. Wenn die letzte Operation ein `TakeR` mit einem `ValR`

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

als zweitem Operand ist soll dies rückgängig gemacht werden. Wenn die Bedingung nicht erfüllt ist, soll nichts verändert werden.

f) Manchmal möchte man eine bestimmte Anzahl Chips von einem Pot an mehreren Stapeln verteilen. Das soll in der folgenden Funktion implementiert werden.

```
share :: CommandR -> [CommandR] -> CommandR -> (CommandR, [CommandR])
```

Es wird von dem Pot (erster Parameter) an jedes Element einer `List CommandR` (zweiter Parameter) einen bestimmten Teil (dritter Parameter) abgegeben werden, d.h. mit `TakeR` vom Pot abgenommen und mit `PutR` zum Element hinzugefügt werden. Ergebnis der Funktion soll der übrige Pot und eine neue Liste mit aktualisierten `CommandR` sein.

Aufgabe 2: Partial application und Komposition

a) Leitet von dem '^'-Operator die Funktionen `sqr` (n^2) und `pot` (2^n) durch partial application ab.

b) Leitet aus der `elem`-Funktion[1] die `isVowel`-Funktion ab.

c) Haskell bietet die Funktionen `ord :: Char -> Int` und `chr :: Int -> Char` für eine Konvertierung zwischen einen `Char` und ihre numerische räpresentation als "Unicode Code Point" an, was für Buchstaben gleich der ASCII-Kodierung[2] ist. Verwendet Partial Application und Funktionskomposition um aus `ord` und `chr` eine Funktion `lower :: Char -> Char` zu konstruieren, die Großbuchstaben in Kleinbuchstaben überführt. Eingaben, die keine Großbuchstaben sind, müssen nicht gesondert behandelt werden.

Aufgabe 3: Listenfunktionale I

Verwendet die Funktionen `map`, `filter`, `foldr` oder `foldl`, um die folgenden Aufgaben zu lösen.

a) Verwendet ein Listenfunktional um eine Funktion `nvcls :: String -> String` zu implementieren, die aus einem String alle Vokale entfernt.

b) Wendet `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.

[1] entspricht dem mathematischen '$\in$'. `elem :: a -> [a] -> Bool`
[2] z.B.: `ord 'a'` = 97; `ord 'A'` = 65