

Altklausur Bry 2012 Remastered für Programmierung und Modellierung 2016

Alexander Isenko

July 17, 2016

Besprechung am 22. Juli 2016

Aufgabe 1

Hier wird nach Funktionen gefragt die ihr euch selber einfallen lassen könnt, total egal wie kompliziert oder einfach sie sind, sie müssen lediglich den Bedingungen entsprechen.

- a) Definieren sie eine monomorphe Funktion (keine Typvariablen, nur feste Typen)
- b) Definieren sie eine polymorphe Funktion (mit Typvariablen)
 - i) parametrisch polymorph (keine Typvariablen vor ($=>$))
 - ii) ad-hoc polymorph (mid. eine Typvariable vor ($=>$))
- c) Definieren sie eine *gecurrierte* Funktion (eine Funktion die ein Tupel nimmt, aber mit `curry` stattdessen die Argumente hintereinander akzeptiert)

```
1  -- Hilfestellung:
2
3  > :t curry
4  curry :: ((a, b) -> c) -> a -> b -> c
```

Aufgabe 3

Definieren sie die Funktion `reverse` für Listen in Haskell.

```
1      > :t reverse
2      [a] -> [a]
3
4      > reverse "hallo"
5      "ollah"
6
7      > reverse [1,2,3]
8      [3,2,1]
9
10     > reverse []
11     []
```

Aufgabe 4

Sei folgender Code gegeben, was ist das Ergebnis von `res`?

```
1  y = 5
2  x = 2
3  goo y      = x * y
4  fuu (x,y) = x + goo y
5  res = (goo y, fuu (5,7))
```

Aufgabe 6

Definieren sie das Standardvektorkreuzprodukt mithilfe von `zip` und `Data.List.foldl/foldr`

```
1  -- Typsignatur:
2  --
3  skalarProdukt :: Num a => [a] -> [a] -> a
4  --
5  -- Beispiel:
6  --
7  skalarProdukt [1,2,3] [4,5,6] = 1*4 + 2*5 + 3*6 = 4 + 10 + 18 = 32
8
9  -- Hilfestellung:
10 --
11 zip :: [a] -> [b] -> [(a,b)]
12 zip []      []      = []
13 zip (x:xs) (y:ys) = (x,y) : zip xs ys
14
15
16
```

```

17
18 foldl :: (b -> a -> b) -> b -> [a] -> b
19 foldl f acc [] = acc
20 foldl f acc (x:xs) = foldl f (f acc x) xs
21
22 foldr :: (a -> b -> b) -> b -> [a] -> b
23 foldr f acc [] = acc
24 foldr f acc (x:xs) = f x (foldr f acc xs)

```

Aufgabe 7

Sei folgender Code gegeben.

```

1  ks = [3,5,7]
2
3  pred = \x -> x < 0
4
5  f p x (y:ys) = if p y
6                  then f (not . p) x ys
7                  else f p      x ys
8  f p x _      = p x
9
10 g x = let f x = x
11        in g (f x)

```

Geben sie wenn möglich den Wert bzw. den Typ der folgenden Ausdrücke an:

a) `f`

b) `f pred 0 []`

c) `f pred 0 ks`

d) `g`

e) `g ks`