Universität Siegen
Lehrstuhl Theoretische Informatik
Carl Philipp Reh

Übungsblatt 1

Aufgabe 1. Installieren Sie GHC (Glasgow Haskell Compiler, `https://www.haskell.org/ghc/`).

Aufgabe 2. Geben Sie jeweils einen Typ zu folgenden Haskell-Ausdrücken an, falls möglich.

(a) `10`

Lösung. `Int`

(b) `1 : []`

Lösung. `[Int]`

(c) `[1,"x"]`

Lösung. Kein möglicher Typ.

(d) `f` mit `f x = x + 1`

Lösung. `Int -> Int`

(e) `f` mit `f g x = (g x) + 1`

Lösung. `f :: (a -> Int) -> a -> Int`

Aufgabe 3. Auf `https://hoogle.haskell.org/` kann man nach Typen und Funktionen suchen, die in Haskell selbst oder auch in externen Bibliotheken definiert sind. Geben Sie die Definitionen folgender Typen bzw. Funktionen an.

(a) `String`

Lösung. `type String = [Char]`

(b) `Bool`

Lösung. `data Bool`, Constructors: `False`, `True`

(c) `filter`

Lösung. `filter :: (a -> Bool) -> [a] -> [a]`

`filter f l` liefert die Liste, die genau die Elemente `x` aus `l` enthält, für die `f x = True` gilt.

(d) `div`

Lösung. `div :: Integral a => a -> a -> a`

Dividiert zwei Zahlen und rundet gegen negativ unendlich. Z.B. gilt `div 5 2 = 2`, aber `div (-5) 2 = -3`.

`Integral a =>` bedeutet hier, dass `a` alles sein kann, was `Integral` implementiert, also z.B. `Int`. Wir werden später sehen, wie man Klassen wie `Integral` definieren und für gewisse Typen implementieren kann.

(e) `mod`

Lösung. `mod :: Integral a => a -> a -> a`

Dividiert zwei Zahlen und liefert den Rest der Division. Dabei gilt `(div x y) * y + (mod x y) = x`.

Aufgabe 4. Für `b :: Bool` und `x,y :: t` können Sie die Syntax

`if b then x else y`

verwenden. Implementieren Sie folgende Funktionen:

(a) `isPowerOf2 :: Int -> Bool`,

wobei `isPowerOf2 x = True`, wenn $x \geq 1$ und es ein $y \in \mathbb{N}$ gibt mit $2^y = x$. In allen anderen Fällen soll `isPowerOf2 x = False` gelten.

Lösung.
```
isPowerOf2 :: Int -> Bool
isPowerOf2 x = if x <= 0 then False
               else if x == 1 then True
               else if mod x 2 /= 0 then False
               else isPowerOf2 (div x 2)
```

Sei $n \in \mathbb{Z}$. Wenn $n \leq 0$, so ist n keine Zweierpotenz und unsere Funktion liefert `False`. Sei nun $n \geq 1$. Wenn $n = 1$, dann gilt $2^0 = 1$, also ist 1 eine Zweierpotenz. In dem Fall liefert unsere Funktion `True`. Sei nun $n \geq 2$. Wenn n nicht durch 2 teilbar ist, dann ist n offensichtlich keine Zweierpotenz. In dem Fall liefert unsere Funktion `False`. Wenn n durch 2 teilbar ist, dann gibt es ein $m \in \mathbb{N}$ mit $m \geq 1$ so, dass $n = 2m$ und es gilt $\text{div}(n, 2) = m$. In dem Fall liefert unsere Funktion `isPowerOf2 m`. Da $m < n$, können wir annehmen, dass dieser Wert korrekt ist. Wenn m eine Zweierpotenz ist, dann existiert ein $y \in \mathbb{N}$ mit $2^y = m$. Dann ist auch $n = 2m = 2 \cdot 2^y = 2^{y+1}$ eine Zweierpotenz. Umgekehrt ist $n = 2m$ keine Zweierpotenz, wenn m keine Zweierpotenz ist.

(b) `filterPowersOf2 :: [Int] -> [Int]`,

wobei `filterPowersOf2 l` genau die Elemente `x` von `l` behält, für die `isPowerOf2 x = True` gilt. Die Reihenfolge der Elemente von `l` soll beibehalten werden.

Lösung. `filterPowersOf2 = filter isPowerOf2`

(c) `filter :: (a -> Bool) -> [a] -> [a]` aus Aufgabe 3 c. Nennen Sie Ihre Implementierung `filter'`, da `filter` in Haskell bereits existiert.

Lösung.
```
filter' :: (a -> Bool) -> [a] -> [a]
filter' f x = case x of
   [] -> []
   (x : xs) -> if f x
               then (x : filter' f xs)
               else filter' f xs
```