Haskell Live

[02] Licht, mehr Licht

Bong Min Kim
e0327177@student.tuwien.ac.at

Christoph Spörk
christoph.spoerk@inode.at

Florian Hassanen
florian.hassanen@googlemail.com

Bernhard Urban
lewurm@gmail.com

15. Oktober 2010

Hinweise

Diese Datei kann als sogenanntes "Literate Haskell Skript" von `hugs` geladen werden, als auch per `lhs2TeX`[1] und LaTeX in ein Dokument umgewandelt werden.

Tips & Tricks

let & where

```
import Data.Char
l1 :: Integer → Integer
l1 x = let
           f :: Integer → Integer
           f y = y + 3 + x   -- 'x' auch hier verwendbar
           g :: Integer
           g = 1337
           in
           f (g + x)
w1 :: Integer → Integer
w1 x = f (g + x)
```

[1] `http://people.cs.uu.nl/andres/lhs2tex`

```
where
f :: Integer → Integer
f y = y + 3 + x   -- 'x' auch hier verwendbar
g :: Integer
g = 1337
```

Listen Funktionen

- `:` – "cons"
- `!!` – Zugriff per Index
- `++` – Verkettung
- `length` – Länge der Liste
- `head / last` – Erstes bzw. letztes Element
- `tail / init` – Alles außer dem ersten bzw. letzten Element
- `take / drop` – Die ersten Elemente nehmen bzw. löschen
- `reverse` – Liste reversieren

Guards & if/else

```
g1 :: Integer → String
g1 x = if x < 0 then
         "negativ"
       else
         if x < 10 then
           "kleiner zehn"
         else
           "groesser gleich zehn"
```

```
g2 :: Integer → String
g2 777 = "777"
g2 x   -- auch hier wieder Achtung! Reihenfolge beachten
  | x < 0 = "negativ"
  | x < 10 = "kleiner zehn"
  | otherwise = "groesser gleich zehn"
```

Tupeln

$$
\begin{aligned}
&t1 :: (Integer, Integer) \to Integer \\
&t1\ x = (fst\ x) + (snd\ x) \\
&t2 :: (Integer, Integer) \to Integer \\
&t2\ (x, y) = x + y
\end{aligned}
$$

quot, rem, div & mod

... sind unterschiedlich fuer negative Zahlen definiert (Details hier nachzulesen: `http://www.haskell.org/onlinereport/basic.html#sect6.4.2`).

Der Unterschied ist für die Übungsbeispiele nicht relevant, da hier nur postive Zahlen verwendet werden. Man kann daher `quot` & `rem` als auch `div` & `mod` verwenden.

Integer und Int...

Für das zweite Aufgabenblatt ist je nach Löesungsweg eventuell die Funktion `chr` aus der Library `Data.Char` mit dem Typen `Int -> Char` nötig. Die Aufgabenblatt verlangt aber einen `Integer`, deswegen ist eine Typumwandlung nötig.

$$
\begin{aligned}
&mychr :: Integer \to Char \\
&mychr\ x = chr\ (fromInteger\ x)
\end{aligned}
$$

Licht, mehr Licht!

$$
\begin{aligned}
&\textbf{type}\ Lampe = Bool \\
&durchschalten :: Lampe \to [Bool] \to Lampe \\
&durchschalten\ akt\ [\,] = akt \\
&durchschalten\ akt\ (x : xs) \\
&\quad \mid x = durchschalten\ (\neg\ akt)\ xs \\
&\quad \mid otherwise = durchschalten\ akt\ xs \\
&switch :: Integer \to Lampe \\
&switch\ n = head\ (reverse\ (switch'\ 1)) \\
&\quad \textbf{where} \\
&\quad switch' :: Integer \to [Lampe] \\
&\quad switch'\ pos \\
&\qquad \mid (pos - 1) \equiv n = [\,] \\
&\qquad \mid otherwise = lampe\_pos : (switch'\ (pos + 1)) \\
&\qquad \textbf{where} \\
&\qquad lampe\_pos :: Lampe
\end{aligned}
$$

```
lampe_pos = durchschalten False durchgaenge
durchgaenge :: [Bool]
durchgaenge = [pos `mod` i == 0 | i <- [1..n]]
```

Kürzere Variante:

```
licht :: Integer -> String
licht n
  | t = "aus"
  | otherwise = "an"
  where t = ((length [x | x <- [1..n], n `mod` x == 0]) `mod` 2) == 0
```