technische universität dortmund

FLDIT

Lehrstuhl 1
Logik in der Informatik

Prof. Dr. Peter Padawitz

Jos Kusiek (*jos.kusiek@tu-dortmund.de*) Wintersemester 2016/2017

Übungen zu Funktionaler Programmierung
Übungsblatt 6

Ausgabe: 18.11.2016, **Abgabe:** 25.11.2016

Aufgabe 6.1 (4 Punkte)

1. Werten Sie folgenden Haskell-Ausdruck mit unendlicher Liste *lazy* aus. Werten Sie schrittweise immer nur den Teil aus, bei dem ein Ergebnis benötigt wird, um weiter auszuwerten zu können.
   `iterate (*2) 3 !! 2`

2. Schreiben Sie die Liste `solutions :: [(Int, Int, Int)]` aus Aufgabe 5.3 so um, dass sie *alle* Lösung der Gleichung $3x^2 + 2y + 1 = z$ enthält. Die Liste wird dadurch unendlich lang. Beispiel: `solutions !! 700` $\rightsquigarrow$ `(7,38,224)`

Lösungsvorschlag

1. ```
   iterate (*2) 3 !! 2
   ⇝ (3:iterate (*2) ((*2) 3)) !! 2
   ⇝ (iterate (*2) ((*2) 3)) !! 1
   ⇝ (((*2) 3):iterate (*2) ((*2) ((*2) 3))) !! 1
   ⇝ (iterate (*2) ((*2) ((*2) 3))) !! 0
   ⇝ (((*2) ((*2) 3)):iterate (*2) ((*2) ((*2) ((*2) 3)))) !! 0
   ⇝ ((*2) ((*2) 3))
   ⇝ (*2) 6
   ⇝ 12
   ```

2. Um eine Endlosschleife zu vermeiden, darf nur die erste Liste in der Komprehension unendlich sein. Da für $x, y > z$ keine Lösungen mehr existieren können, ist dies eine sinnvolle Einschränkung.

   ```
   solutions :: [(Int , Int , Int )]
   solutions = [ (x,y,z)
     | z <- [0..] , y <- [0..z] , x <- [0..z]
     , 3*x^2 + 2*y + 1 == z]
   ```

Aufgabe 6.2 (2 Punkte)

1. Erweitern Sie die vorgestellten Datentypen für Zahlen um einen Datentyp für rationale Zahlen. Basieren Sie den Datentyp nur auf den anderen Datentypen für Zahlen (`Nat, Int', PosNat`).

2. Definieren Sie eine Konstante $c = -\frac{1}{2}$ für Ihren Datentyp.

Lösungsvorschlag
   ```

```
data Rat = Rat Int' PosNat

c :: Rat
c = Rat (Minus One) (Succ' One)
```

Aufgabe 6.3 (4 Punkte) Definieren Sie folgende Haskell-Funktionen.

1. `indexNat :: [a] -> Nat -> a`, wie **`(!!)`** für den Datentyp **`Nat`** anstatt **`Int`**.
2. `colistTake :: Int -> Colist a -> Colist a`, wie **`take`** nur für **`Colist a`** anstatt **`[a]`**.
3. `getX :: Point -> Double` liest die x-Koordinate eines Punktes aus.
4. `setX :: Double -> Point -> Point` setzt die x-Koordinate eines Punktes auf den angegebenen Wert.

Die Datentypen `Point` und `Colist` sind wie folgt definiert:

```
data Point = Point Double Double
data Colist a = Colist (Maybe (a, Colist a))
-- Liste 5:3:[] als Colist:
example = Colist (Just (5, Colist (Just (3, Colist Nothing))))
```

Lösungsvorschlag

```
indexNat :: [a] -> Nat -> a
indexNat (a:_) Zero = a
indexNat (_:s) (Succ n) = indexNat s n

colistTake :: Int -> Colist a -> Colist a
colistTake 0 _              = Colist Nothing
colistTake n (Colist (Just (a,s)))
  | n > 0 = Colist (Just (a, colistTake (n-1) s))
colistTake _ (Colist Nothing) = Colist Nothing

getX :: Point -> Double
getX (Point x _) = x

setX :: Double -> Point -> Point
setX x (Point _ y) = Point x y
```

Aufgabe 6.4 (2 Punkte)

1. Zeigen Sie, dass der Typ `[()]` isomorph zu `Nat` ist. Schreiben Sie zwei Funktionen **`to :: [()] -> Nat`** und **`from :: Nat -> [()]`**, welche die Bedingungen `to . from = id` und `from . to = id` erfüllen.

Lösungsvorschlag

```
to :: [()] -> Nat
to (():xs) = Succ (to xs)
to []      = Zero

from :: Nat -> [()]
from (Succ n) = () : from n
from Zero     = []
```