

Übungen zu Funktionaler Programmierung

Übungsblatt 7

Ausgabe: 1.12.2017, **Abgabe:** 8.12.2017 – 16:00 Uhr, **Block:** 4

Aufgabe 7.1 (6 Punkte)

- a) Schreiben Sie eine Instanz der Klasse Eq für den Typ Nat.
- b) Schreiben Sie eine Instanz der Klasse Ord für den Typ Nat. Es ist ausreichend den Operator (\leq) zu definieren.
- c) Schreiben Sie eine Instanz der Klasse Enum für den Typ Nat. Es ist ausreichend die Funktionen toEnum und fromEnum zu definieren.

Beispiel:

```
take 3 $ map fromEnum [Zero .. ] ~> [0,1,2]
```

- d) Schreiben Sie eine Instanz der Klasse Show für den Typ Nat. Die Ausgabe soll sich ähnlich wie vom Typ Int verhalten:

```
show Zero ~> "0"
show (Succ Zero) ~> "1"
show (Succ (Succ (Succ Zero))) ~> "3"
```

- e) Schreiben Sie eine Instanz der Klasse Num für den Typ Nat. Nutzen Sie folgende Vorgabe:

```
instance Num Nat where
  negate = undefined
  abs n = n
  signum Zero = Zero
  signum n = Succ Zero
  fromInteger = toEnum . fromInteger
```

Sie müssen lediglich die fehlenden Operatoren (+) und (*) definieren (Stichwort: Peano-Axiome).

- f) Ändern Sie den Typ der Liste solutions in [(Nat,Nat,Nat)] und passen Sie die Definition entsprechend an.

```
solutions :: [(Int,Int,Int)]
solutions = [ (x,y,z)
  | z <- [0..] , y <- [0..z^2], x <- [0..z^2]
  , 2*x^3 + 5*y + 2 == z^2 ]
```

Aufgabe 7.2 (3 Punkte) *Ausgabe*

Schreiben Sie eine Instanz der Klasse `Show` für den folgenden Datentyp für nichtleere binäre Bäume:

```
data STree a = BinS (STree a) a (STree a) | LeftS  (STree a) a
              | RightS  a (STree a)  | LeafS  a
```

Die Ausgabe soll sich an den binären Bäumen aus der Vorlesung orientieren:

```
BinS (LeftS (LeafS 9) 2) 4 (RightS 7 (LeafS 3)) ~> 4(2(9,),7(,3))
```

Aufgabe 7.3 (3 Punkte) *Bäume mit beliebigem Ausgrad*

Definieren Sie folgende Funktionen.

- a) `treeAnd :: Tree Bool -> Bool` – Verhält sich wie `and`. Ergibt `True`, falls alle Knoten den Wert `True` enthalten.
- b) `treeZip :: Tree a -> Tree b -> Tree (a,b)` – Eine Variante von `zip` für Bäume.