SoSe 2014

Prof. Dr. Margarita Esponda

ProInformatik II: Funktionale Programmierung

15. Übungsblatt (16. Tag)

1. Aufgabe

Betrachten Sie folgende Definition der **add**-Funktion für den algebraischen Datentyp **Natural**:

```
data Nat = Zero | S Nat

add ::  Nat -> Nat -> Nat
add a Zero = a
add a (S b) = add (S a) b

add' ::  Nat -> Nat -> Nat
add' a Zero  = a
add' a (S b) = S (add' a b)
```

Beweisen Sie mittels Induktion, dass beide Funktionen äquivalent sind.

```
add  =  add'
```

2. Aufgabe

Betrachten Sie folgende Funktionsdefinitionen:

```
(++)  []     ys      = ys
(++)  (x:xs)  ys     = x : (xs ++ ys)

reverse   []          =  []
reverse  (x:xs)       =  reverse xs  ++  [x]

elem  x  []                  = False
elem  x  (y:ys) | x==y       =  True
                | otherwise  =  elem  ys

map  f  []            = []
map  f  (x:xs)        = f x : map  f  xs

(.) f g  x            = f(g(x))

filter  p  []             =  []
filter  p  (x:xs) | p x         =  x : filter  p x s
                  | otherwise   =  filter  p  xs
```

Beweisen Sie mittels struktureller Induktion über die Liste **xs**, dass für jede endliche Listen **xs** und **ys** folgende Gleichungen gelten:

a) elem a (xs ++ ys) = elem a xs || elem a ys

b) (filter p . map f) xs = ((map f) . filter (p.f)) xs

3. Aufgabe

Betrachten Sie folgende Haskell-Funktionen:

```
zip :: ([a], [b]) -> [(a, b)]
zip ([ ], _) = [ ]
zip (_, [ ]) = [ ]
zip ((x:xs), (y:ys)) = (x, y) : zip (xs, ys)

unzip :: [(a, b)] -> ([a], [b])
unzip [ ] = ([ ], [ ])
unzip ((x,y):zs) = (x:xs, y:ys)
                    where
                        (xs, ys) = unzip zs
```

Beweisen Sie, dass für alle endliche Listen xs :: [(a,b)], folgende Eigenschaft gilt:

```
zip (unzip xs) = xs
```

4. Aufgabe

Unter Verwendung folgender Funktionsdefinitionen

```
data Tree a = Leaf a | Node  (Tree a)  (Tree a)

sumLeaves  ::  Tree a   ->   Integer
sumLeaves   (Leaf x)         = 1
sumLeaves   (Node  lt  rt)   =  sumLeaves  lt  +  sumLeaves  rt

sumNodes :: Tree a -> Integer
sumNodes   (Leaf x)          =  0
sumNodes   (Node  lt rt)     =  1 +  sumNodes  lt   +   sumNodes  rt
```

Beweisen Sie, dass für alle endlichen Bäume t :: Tree a gilt:

```
sumLeaves  t  =  sumNodes  t  +  1
```

5. Aufgabe

Betrachten Sie folgende Variante des algebraischen Datentyps der 3. Aufgabe und folgende Funktionsdefinitionen

```
data Tree a = Nil | Node a (Tree a) (Tree a) | Leaf a

sumTree :: (Num a) => Tree a -> a
sumTree Nil              = 0
sumTree (Leaf x)         = x
sumTree (Node x l r)  = x + sumTree l  +  sumTree r

tree2list :: (Num a) => Tree a -> [a]
tree2list Nil              = []
tree2list (Leaf x)         = [x]
tree2list (Node x l r)   = tree2list l ++ [x] ++ tree2list r
```

```
sum :: (Num a) => [a] -> a
sum []       = 0
sum (x:xs)  = x + sum xs
```

Beweisen Sie, dass folgende Eigenschaft gilt:

```
sum.tree2list = sumTree
```