WS 2017/2018

Funktionale Programmierung

9. Übungsblatt

Prof. Dr. Margarita Esponda

(Abgabetermin: Mo., den 08.01.2018, um 10:10, Hörsaal)

1. Aufgabe (6 Punkte)

Eine **unfold** Funktion, die ein einfaches rekursives Pattern, um eine Liste zu produzieren, implizit darstellt, kann wie folgt definiert werden.

```
unfold p f g x | p x = []
              | otherwise = f x : unfold p f g (g x)
```

Redefinieren Sie unter Verwendung der **unfold**-Funktion folgende Funktionen:

(map f), (iterate f) und dec2bin

2. Aufgabe (6 Punkte)

Analysieren sie die Komplexität der Funktionen **makeTree** und **codeTable** aus dem Hoffman-Algorithmus (der Algorithmus wird in der Vorlesung erläutert).

3. Aufgabe (4 Punkte)

Beweisen Sie mittels struktureller Induktion über **xs** folgende Eigenschaft:

```
filter p (xs ++ ys)  ==  filter p xs  ++  filter p ys
```

4. Aufgabe (5 Punkte)

Betrachten Sie folgende Funktionsdefinition:

```
f = flip g
```

Beweisen Sie mittels struktureller Induktion über **xs**, dass

```
foldl g z xs = foldr f z (reverse xs)
```

5. Aufgabe (4 Bonuspunkte)

```
Sei   h  :: a -> a -> a
      g  :: a -> a -> a
und   z  :: a
```

Nehmen Sie an, dass für alle x::a folgendes gilt:

```
h x (g y z) = g z (h x y)
```

Beweisen Sie mittels strukturelle Induktion über die Liste **xs**, dass für alle x, y :: a, wenn g kommutativ ist, folgende Eigenschaft gilt:

```
h x (foldl g y xs) = foldl g (h x y) xs
```

6. Aufgabe (6 Punkte)

Gegeben seien die folgenden Standard Haskell-Funktionen:

```
takeWhile :: (a -> Bool) -> [a] -> [a]
takeWhile p []     = []
takeWhile p (x:xs)  | p x          = x : takeWhile p xs
                    | otherwise = []
```

```
dropWhile :: (a -> Bool) -> [a] -> [a]
dropWhile p []     = []
dropWhile p (x:xs)  |  p x         = dropWhile p xs
                    | otherwise = (x:xs)

span :: (a -> Bool) -> [a] -> ([a], [a])
span p [] = ([], [])
span p (x:xs) | p x       = (x:ys, zs)
              | otherwise = ([], x:xs)
                            where (ys, zs) = span p xs
```

Beweisen Sie durch strukturelle Induktion, dass für alle endlichen Listen **xs** und beliebige Prädikate p (mit passender Signatur) gilt:

span p xs = (takeWhile p xs, dropWhile p xs)

7. Aufgabe (6 Punkte)

Gegeben seien die folgenden algebraischen Datentypen und Funktionsdefinitionen:

```
data Tree a = Nil | Leaf  a | Node  a  (Tree a)  (Tree a)
                                                    deriving Eq
mapTree :: (a -> b) -> Tree a -> Tree b
mapTree f Nil = Nil
mapTree f (Leaf x) = Leaf (f x)
mapTree f (Node x lt rt) = (Node (f x) (mapTree f lt) (mapTree f rt))

tree2List :: Tree a -> [a]
tree2List Nil = []
tree2List (Leaf x) = [x]
tree2List (Node x rt lt) = tree2List rt ++ [x] ++ tree2List lt
```

Beweisen Sie durch strukturelle Induktion über **t**, dass für alle endlichen Bäume **t** folgende Gleichung gilt:

map f (tree2List **t**) = tree2List (mapTree **t**)