5. Übung (04.–8. Mai 2020)

Programmierung

Hinweise In den folgenden Aufgaben sollen Beweise von Programmeigenschaften für Haskell-Programme durchgeführt werden. Sie können Ihre Lösungen im Klartext bzw. formatiert in Markdown über Opal einreichen. Halten Sie sich an die folgende Form (wobei die Randbemerkungen wie z.B. `(#2)` Zeilennummern im gegebenem Haskell-Code angeben):

```
# Aufgabe 42
## Induktionsbasis

Sei xs = [].

length (dup xs) = length (dup [])
                = length []         (#2)
                = …
                = pow xs - 1

## Induktionsschritt

Sei xs :: [Int], sodass length (dup xs) = pow xs - 1 (IV).
Weiterhin, sei x :: Int eine beliebige Zahl.
Wir zeigen nun, dass length (dup (x:xs)) = pow (x:xs) - 1.

length (dup (x:xs)) = length (x : (dup xs ++ dup xs))       (#3)
                    = …
                    = pow (x:xs) - 1                         (#7)
```

Aufgabe 1 (AGS 12.3.20)

Zeigen Sie unter Verwendung der folgenden Definitionen durch strukturelle Induktion die Gültigkeit der Gleichung `sum (foo xs)` = `2 * sum xs - length xs` für jedes `xs :: [Int]`.

```
 1  foo :: [Int] -> [Int]
 2  foo []     = []
 3  foo (x:xs) = x : x : (-1) : foo xs
 4
 5  sum :: [Int] -> Int
 6  sum []     = 0
 7  sum (x:xs) = x + sum xs
 8
 9  length :: [Int] -> Int
10  length []     = 0
11  length (x:xs) = 1 + length xs
```

Zeigen Sie dazu den Induktionsanfang und den Induktionsschritt; geben Sie beim Induktionsschritt die Induktionsvoraussetzung an. Geben Sie bei jeder Umformung die benutzte *Definition*, *Eigenschaft* bzw. *Induktionsvoraussetzung* an. Quantifizieren Sie alle Variablen.

Aufgabe 2 (AGS 12.3.29)

Folgende Definitionen seien gegeben:

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

preOrder :: BinTree a -> [a]
preOrder (Leaf x) = [x]
preOrder (Node x l r) = [x] ++ preOrder l ++ preOrder r

mPostOrder :: BinTree a -> [a]
mPostOrder (Leaf x) = [x]
mPostOrder (Node x l r) = mPostOrder r ++ mPostOrder l ++ [x]
```

Sei außerdem `rev :: [a] -> [a]` eine Funktion, sodass für jeden Typ `a` folgende zwei Eigenschaften gelten:

$$\forall \texttt{x :: a}: \qquad \texttt{rev [x]} = \texttt{[x]} \tag{H1}$$

$$\forall \texttt{xs, ys :: [a]}: \qquad \texttt{rev (xs ++ ys)} = \texttt{rev ys ++ rev xs} \tag{H2}$$

Gehen Sie davon aus, dass die Funktion `(++) :: [a] -> [a] -> [a]` assoziativ ist.

(a) Sei `a` ein Typ, `x :: a` und `xs, ys :: [a]`. Zeigen Sie, dass folgende Gleichung gilt:

$$\texttt{[x] ++ rev ys ++ rev xs} = \texttt{rev (xs ++ ys ++ [x])} \tag{H3}$$

Hinweis: Sie dürfen (H1) und (H2) verwenden. Für den Beweis der Gültigkeit dieser Gleichung ist *keine* Induktion nötig.

(b) Zeigen Sie durch strukturelle Induktion, dass die Aussage

$$\texttt{preOrder t} = \texttt{rev (mPostOrder t)}$$

Für jeden Typ `a` und jeden Baum `t :: BinTree a` gilt. Zeigen Sie dazu den Induktionsanfang und den Induktionsschritt; geben Sie beim Induktionsschritt die Induktionsvoraussetzung an. Geben Sie bei jeder Umformung die benutzte *Definition*, *Eigenschaft* bzw. *Induktionsvoraussetzung* an. Quantifizieren Sie alle Variablen.

Hinweis: Sie dürfen dafür die Eigenschaften (H1), (H2) und (H3) verwenden.

Zusatzaufgabe 1 (AGS 12.3.28)

Folgende Definitionen seien gegeben:

```
data IntTree = Node Int [IntTree]

yield :: IntTree -> [Int]
```

```
yield (Node i []) = [i]
yield (Node i ts) = concat (map yield ts)

yieldProd :: IntTree -> Int
yieldProd (Node i []) = i
yieldProd (Node i ts) = product (map yieldProd ts)
```

Zeigen Sie unter Verwendung der obigen Definitionen durch strukturelle Induktion die Gültigkeit der Gleichung

$$\texttt{product (yield t)} = \texttt{yieldProd t}$$

für jeden Baum `t :: IntTree`. Sie dürfen dabei nutzen, dass für alle Typen `a`, `b`, positive Integer $\texttt{k} > 0$, Integer `i :: Int`, Funktionen `f :: a -> b`, Werte `a1, ..., ak :: a` und Listen `l1, ..., lk :: [Int]` folgende Eigenschaften gelten:

$$\texttt{product [i]} = \texttt{i} \tag{H1}$$

$$\texttt{map f [a1, ..., ak]} = \texttt{[f a1, ..., f ak]} \tag{H2}$$

$$\texttt{product (concat [l1, ..., lk])} = \texttt{product [product l1, ..., product lk]} \tag{H3}$$

Zeigen Sie dazu den Induktionsanfang und den Induktionsschritt; geben Sie beim Induktionsschritt die Induktionsvoraussetzung an. Geben Sie bei jeder Umformung die benutzte *Definition*, *Eigenschaft* bzw. *Induktionsvoraussetzung* an. Quantifizieren Sie alle Variablen.

Zusatzaufgabe 2 (AGS 12.3.22)

```
data BinTree a = Branch a (BinTreee a) (BinTree a) | Leaf a

p :: BinTree a -> [a]
p (Leaf x)       = [x]
p (Branch x s t) = [x] ++ (p s ++ p t)

d :: BinTree a -> BinTree a -> BinTree a
d (Leaf x) u       = Branch x u u
d (Branch x s t) u = Branch x s (d t u)
```

Zeigen Sie unter Verwendung der obigen Definitionen durch strukturelle Induktion die Gültigkeit der Gleichung

$$\texttt{p (d t u)} = \texttt{p t ++ (p u ++ p u)}$$

für jeden Typ `a` und alle Bäume `t, u :: BinTree a`. Zeigen Sie dazu den Induktionsanfang und den Induktionsschritt; geben Sie beim Induktionsschritt die Induktionsvoraussetzung an. Geben Sie bei jeder Umformung die benutzte *Definition*, *Eigenschaft* bzw. *Induktionsvoraussetzung* an. Quantifizieren Sie alle Variablen. Gehen Sie davon aus, dass die Funktion `(++) :: [a] -> [a] -> [a]` assoziativ ist.

Hinweis: Für den Beweis genügt die Induktion über der Struktur des Baumes `t`.