Übung 7 - Aufgabe 2a(a)

Beweisen Sie mithilfe der Listeninduktion die Aussage

`xs ++ [] = xs`.

Induktionsanfang

`xs = []`

`[] ++ [] = []` (app_1)

Induktionsschritt

`xs = (x:xs’)`

Aussage gilt für `xs’`.

`(x:xs’) ++ [] = (x : (xs‘ ++ []))` (app_2)

`               = (x : xs’)` (IV)

Übung 7 - Aufgabe 2a(b)

Beweisen Sie mithilfe der Listeninduktion die Aussage

`concat (l1 ++ l2) = concat l1 ++ concat l2`.

Definition

```
concat:: [[a]] -> [a]
concat []     = []
concat (x:xs) = x ++ concat xs
```

Induktionsanfang

`xs = []`

`concat ([] ++ l2) = concat l2` (app_1)

`                  = [] ++ concat l2` (app_{1_R})

`                  = concat [] ++ concat l2` (con_{1_R})

Übung 7 - Aufgabe 2a(b)

Induktionsschritt

`xs = (x:xs’)`

Aussage gilt für `xs’`.

`concat ((x:xs’) ++ l2) = concat (x:xs’) ++ concat l2`

`                       = concat (x:(xs’ ++ l2))` (app_2)

`                       = x ++ concat (xs’ ++ l2)` (con_2)

`                       = x ++ concat xs’ ++ concat l2` (IV)

`                       = concat (x:xs’) ++ concat l2` (con_{2_R})

Übung 7 - Aufgabe 2a(c)

Beweisen Sie mithilfe der Listeninduktion die Aussage

`length (l1 ++ l2) = (length l1) + (length l2)`.

Induktionsanfang

`l1 = []`

`length ([] ++ l2) = length l2` (app_1)

`                  = 0 + (lenth l2)` $(math)$

`                  = (length []) + (length l2)` (len_{1_R})

Übung 7 - Aufgabe 2a(c)

Induktionsschritt

`l1 = (x:xs')`

Aussage gilt für `xs'`.

```
length ((x:xs') ++ l2) = length (x : (xs' ++ l2))  (app2)
                       = 1 + length (xs' ++ l2)  (len2)
                       = 1 + (length xs') + length l2  (IV)
                       = length (x : xs') + length l2  (len2R)
```


Übung 7 - Aufgabe 2a(d)

Beweisen Sie mithilfe der Listeninduktion die Aussage

`reverse (l1 ++ l2) = (reverse l2) + (reverse l1).`

Induktionsanfang

`l1 = []`

```
reverse ([] ++ l2) = reverse l2  (app1)
                   = (reverse l2) ++ []  (Ü7/2a-a)
                   = (reverse l2) ++ (reverse [])  (rev1R)
```


Übung 7 - Aufgabe 2a(d)

Induktionsschritt

`l1 = (x:xs')`

Aussage gilt für `xs'`.

```
reverse ((x:xs') ++ l2) = reverse (x : (xs' ++ l2))  (app2)
                        = reverse (xs' ++ l2) ++ [x] (rev2)
                        = ((reverse l2) ++ (reverse xs')) ++ [x]  (IV)
                        = (reverse l2) ++ ((reverse xs') ++ [x])  (ass.)
                        = (reverse l2) ++ (reverse (x:xs'))  (rev2R)
```


Übung 7 - Aufgabe 2b

Beweisen Sie die Eigenschaft der Funktion `foldr`

`foldr f e (xs ++ ys) = f (foldr f e xs) (foldr f e ys).`

Induktionsanfang

`xs = []`

```
foldr f e ([] ++ ys) = foldr f e ys (app1)
                     = f e (foldr  f e ys) (e-neutral)
                     = f (foldr f e []) (foldr f e ys) (foldr1)
```

Übung 7 - Aufgabe 2b

Induktionsschritt

`xs = (x:xs')`

Aussage gilt für `xs'`.

`foldr f e ((x:xs') ++ ys) = foldr f e (x : (xs' ++ ys))` (app_2)

`= f x (foldr f e (xs' ++ ys))` $(foldr_2)$

`= f x (f (foldr f e xs')(foldr f e ys))` (IV)

`= f (f x (foldr f e xs'))(foldr f e ys)` $(ass.)$

`= f (foldr f e (x : xs'))(foldr f e ys)` $(foldr_2)$

Übung 7 - Aufgabe 2c

Beweisen Sie, daß für beliebige Typen `a, b, c`, Werte `e :: c`, Funktionen `f :: a -> b` und `g :: b -> c -> c` die folgende Gleichung gilt

`(foldr g e) . (map f) = foldr h e .`

Induktionsanfang

`xs = []`

`foldr g e (map f []) = foldr g e []` (map_1)

`= e` $(foldr_1)$

`= foldr h e []` $(foldr_R)$

Übung 7 - Aufgabe 2c

Induktionsschritt

`xs = (x:xs')`

Aussage gilt für `xs'`.

`foldr g e (map f (x : xs')) = foldr g e ((f x) : map f xs')` (map_2)

`= g (f x) (foldr g e (map f xs'))` $(foldr_2)$

`= g (f x) (foldr h e xs')` (IV)

`= h x (foldr h e xs')` $(Def.h)$

`= foldr h e (x : xs')` $(foldr_{2_R})$

Übung 7 - Aufgabe 3

Zu zeigen

`flatten (myswap t) = reverse (flatten t) .`

Induktionsanfang

`flatten (myswap (Leaf x)) = flatten (Leaf x)` (mys_1)

`= [x]` (fla_1)

`= reverse ([] ++ [x])` $(Vo06)$

`= reverse [x]` (app_1)

`= reverse (flatten (Leaf x))` (fla_{1_R})

Übung 7 - Aufgabe 3

Induktionsschritt

```
flatten (myswap (Node l r))=flatten (Node (myswap l)(myswap r))
             = (flatten (myswap r)) ++ (flatten (myswap l))
             = (reverse (flatten r)) ++ (reverse (flatten l))
             = (reverse ((flatten l) ++ (flatten r))
```

Begründungen der Schritte: (mys_2), (fla_2), (IV), $(\ddot{U}7/A2a-d)$