

1 map concat

$(.) :: (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$
 $f . g = \lambda x \rightarrow f (g x)$

$concat :: [[a]] \rightarrow [a]$
 $concat [] = []$
 $concat (x:xs) = x ++ (concat xs)$

$map :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$
 $map f [] = []$
 $map f (x:xs) = (f x) : (map f xs)$

Claim 1.1.

$map f . concat = concat . map (map f)$

Proof. structural induction on xs
 $xs = []$

$(map f . concat) xs = (map f . concat) []$
 $= map f (concat [])$
 $= map f []$
 $= []$
 $= concat []$
 $= concat (map (map f) [])$
 $= (concat . map (map f)) []$
 $= (concat . map (map f)) xs$

$xs = (x:xs')$ structural induction on x
 $x = []$

$(map f . concat) xs = (map f . concat) (x:xs')$
 $= map f (concat ([]:xs'))$
 $= map f ([] ++ (concat xs'))$
 $= map f (concat xs')$
 $\stackrel{IH}{=} (concat . map (map f)) xs'$
 $= concat (map (map f) xs')$
 $= [] ++ (concat (map (map f) xs'))$
 $= concat ([] : (map (map f) xs'))$
 $= concat ((map f []) : (map (map f) xs'))$
 $= concat (map (map f) ([]:xs'))$
 $= concat (map (map f) xs)$
 $= (concat . map (map f)) xs$

$x = (y:ys)$

$(map f . concat) xs = (map f . concat) (x:xs')$
 $= map f (concat ((y:ys):xs'))$
 $= map f ((y:ys) ++ (concat xs'))$
 $= (f y) : (map f (ys ++ (concat xs')))$
 $= (f y) : (map f (concat (ys:xs')))$

```

IH
= (f y):((concat.map (map f)) (ys:xs'))
= (f y):(concat (map (map f) (ys:xs')))
= (f y):(concat ((map f ys):(map (map f) xs'))))
= (f y):((map f ys)++(concat (map (map f) xs')))
= ((f y):(map f ys))++(concat (map (map f) xs'))
= (map f (y:ys))++(concat (map (map f) xs'))
= (map f x)++(concat (map (map f) xs'))
= concat ((map f x):(map (map f) xs'))
= concat (map (map f) (x:xs'))
= concat (map (map f) xs)
= (concat.map (map f)) xs

```

□

alternative proof

Proof. we show the equivalent claim

$\text{map } f (\text{concat } xs) = \text{concat } (\text{map } (\text{map } f) xs)$

structural induction on xs

$xs = []$

```
map f (concat xs) = map f (concat [])
                  = map f []
                  = []
                  = concat []
                  = concat (map (map f) [])
                  = concat (map (map f) xs)
```

$xs = (x:xs')$ structural induction on x

$x = []$

```
map f (concat xs) = map f (concat ([]:xs'))
                  = map f ([]++(concat xs'))
                  = map f (concat xs')
IH
                  = concat (map (map f) xs')
                  = []++(concat (map (map f) xs'))
                  = concat ([]:(map (map f) xs'))
                  = concat ((map f []):(map (map f) xs'))
                  = concat (map (map f) ([]:xs'))
                  = concat (map (map f) xs)
```

$x = (y:ys)$

```
map f (concat xs) = map f (concat ((y:ys):xs'))
                  = map f ((y:ys)++(concat xs'))
                  = (f y):(map f (ys++(concat xs')))
                  = (f y):(map f (concat (ys:xs')))
IH
                  = (f y):(concat (map (map f) (ys:xs')))
                  = (f y):(concat ((map f ys):(map (map f) xs'))))
                  = (f y):((map f ys)++(concat (map (map f) xs')))
                  = ((f y):(map f ys))++(concat (map (map f) xs'))
                  = (map f (y:ys))++(concat (map (map f) xs'))
                  = (map f x)++(concat (map (map f) xs'))
                  = concat ((map f x):(map (map f) xs'))
                  = concat (map (map f) (x:xs'))
                  = concat (map (map f) xs)
```

□

2 foldl map

```
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl f a []      = a
foldl f a (x:xs) = foldl f (f a x) xs
```

Claim 2.1.

```
foldl f a.map g = foldl h a
                  h b x = f b (g x)
```

Proof. we show the equivalent claim

```
foldl f a (map g xs) = foldl h a xs
                      h b x = f b (g x)
```

by structural induction on xs

xs=[]

```
foldl f a (map g xs) = foldl f a (map g [])
                     = foldl f a []
                     = a
                     = foldl h a []
                     = foldl h a xs
```

xs=(x:xs')

```
foldl f a (map g xs) = foldl f a (map g (x:xs'))
                     = foldl f a ([g x]++(map g xs'))
                     = foldl f (f a (g x)) (map g xs')
                      $\stackrel{IH}{=}$  foldl h (f a (g x)) xs'
                     = foldl h (h a x) xs'
                     = foldl h a (x:xs')
                     = foldl h a xs
```

□

3 bag

```
data Bag a = ListBag [(a,Integer)] deriving (Eq,Show)

add :: Eq a => (a,Integer) -> (Bag a) -> (Bag a)
add (ele,count) (ListBag bag)
  | ele 'elem' map fst bag =
    = ListBag (bag++[ele,count])
  | otherwise

unite :: (Bag a) -> (Bag a) -> (Bag a)
unite (ListBag [])      bagy = bagy
unite (ListBag (x:xs)) bagy = unite (ListBag xs) (add x bagy)

bag :: [a] -> (Bag a)
bag []      = ListBag []
bag (x:xs) = add (x,1) (bag xs)
```

Claim 3.1.

$\text{bag } (xs++ys) = \text{unite } (\text{bag } xs) (\text{bag } ys)$

Proof. structural induction on xs
xs=[]

```
bag (xs++ys) = bag ([]++ys)
              = bag ys
              = unite (ListBag []) (bag ys)
              = unite (bag []) (bag ys)
              = unite (bag xs) (bag ys)

xs=(x:xs')

bag (xs++ys) = bag ((x:xs')++ys)
              = add (x,1) (bag (xs'++ys))
               $\stackrel{\text{IH}}{=}$  add (x,1) (unite (bag xs') (bag ys) )

              = unite (add (x,1) (bag xs')) (bag ys)
              = unite (bag (x:xs')) (bag ys)
              = unite (bag xs) (bag ys)
```

□