1 map concat

```
(.) :: (b -> c) -> (a -> b) -> a -> c
f . g = \x -> f (g x)

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

map :: (a -> b) -> [a] -> [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')
                  =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'))
                  = 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
`map f (concat xs) = concat (map (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'))
                  = 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')
                    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 bagx)
    |ele 'elem' map fst bagx = ListBag (map (inc (ele,count)) bagx)
    |otherwise               = ListBag (bagx++[(ele,count)])

inc :: Eq a => (a,Integer) -> (a,Integer) -> (a,Integer)
inc (e,c) (x,z)
    |e==x    = (e,c+z)
    |othwise = (x,z)

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

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

Claim 3.1.

```
bag (xs++ys) = unite (bag xs) (bag ys)
```

used but unproven statements:

```
S1: map f (xs++ys) = (map f xs)++(map f ys)
S2: map f (map g xs) = map (f.g) xs
S2: (inc (e,z)).(inc (e,c)) = inc (e,z+c)
S4: (inc (f,z)).(inc (e,c)) = (inc (e,c)).(inc (f,z))
```

Lemma 3.2. *if* `A1: not e 'elem' map fst xs` *inc does not do anything*

```
xs = map (inc (e,c)) xs
```

Proof. of Lemma 3.2 by structural induction on xs
`A1: not e 'elem' map fst xs`
`xs=[]`

```
xs = []
   = map (inc (e,c)) []
   = map (inc (e,c)) xs
```

`xs=((f,z):xs')`

```
xs = (f,z):xs'
   =IH (f,z):(map (inc (e,c)) xs')
   =A1 (inc (e,c) (f,z)):(map (inc (e,c)) xs')
   = map (inc (e,c)) ((f,z),xs')
   = map (inc (e,c)) xs
```

□

Lemma 3.3. *adding same element*

`add (e,z) (add (e,c) (ListBag xs)) = add (e,c+z) (ListBag xs)`

Proof. of Lemma 3.3
Case 1 : `A1: not e 'elem' map fst xs`

`add (e,z) (add (e,c) (ListBag xs))`
$\overset{A1}{=}$ `add (e,z) (ListBag xs++[(e,c)])`
$=$ `ListBag (map (inc (e,z)) (xs++[(e,c)]))`
$\overset{S1}{=}$ `ListBag (map (inc (e,z)) xs)++(map (inc (e,z)) [(e,c)])`
$\overset{3.2,A1}{=}$ `xs++(map (inc (e,z)) [(e,c)])`
$=$ `xs++((inc (e,z) (e,c)):(map (inc (e,z)) []))`
$=$ `xs++((inc (e,z) (e,c)):[])`
$=$ `xs++((e,z+c):[])`
$=$ `xs++[(e,z+c)]`
$\overset{A1}{=}$ `add (e,c+z) (ListBag xs)`

Case 2 : `A1: e 'elem' map fst xs`

`add (e,z) (add (e,c) (ListBag xs))`
$\overset{A1}{=}$ `add (e,z) (ListBag (map (inc (e,c)) xs))`
$\overset{A1}{=}$ `ListBag (map (inc (e,z)) (map (inc (e,c)) xs))`
$\overset{S2}{=}$ `ListBag (map (inc (e,z)).(inc (e,c)) xs)`
$\overset{S3}{=}$ `map (inc (e,c+z)) xs`
$\overset{A1}{=}$ `add (e,c+z) (ListBag xs)`

□

Lemma 3.4. *adding different element*
`A1: e 'elem' map fst xs`
`A2: f!=e`

`add (f,z) (add (e,c) (ListBag xs)) = add (e,c) (add (f,z) (ListBag xs))`

Proof. of Lemma 3.4
`A1: not e 'elem' map fst xs`
Case 1 : `A3: not f 'elem' map fst xs`

`add (f,z) (add (e,c) (ListBag xs))`
$\overset{A1}{=}$ `add (f,z) (ListBag (map (inc (e,c)) xs))`
$\overset{A3}{=}$ `ListBag ((map (inc (e,c)) xs)++[(f,z)])`
$\overset{A3}{=}$ `ListBag ((map (inc (e,c)) xs)++[inc (e,c) (f,z)])`
$=$ `ListBag ((map (inc (e,c)) xs)++(map (inc (e,c)) [(f,z)]))`
$\overset{S1}{=}$ `ListBag (map (inc (e,c)) (xs++[(f,z)]))`
$\overset{A1,A2}{=}$ `add (e,c) (ListBag (xs++[(f,z)]))`
$\overset{A3}{=}$ `add (e,c) (add (f,z) (ListBag xs))`

Case 2 : `A3: f 'elem' map fst xs`

```
add (f,z) (add (e,c) (ListBag xs))
            A1
            =  add (f,z) (ListBag (map (inc (e,c)) xs))
            A3
            =  ListBag (map (inc (f,z)) (map (inc (e,c)) xs))
            S2
            =  ListBag (map (inc (f,z)).(inc (e,c)) xs)
            S4
            =  ListBag (map (inc (e,c)).(inc (f,z)) xs)
            S2
            =  ListBag (map (inc (e,c)) (map (inc (f,z)) xs))
            A1
            =  add (e,c) (ListBag (map (inc (f,z)) xs))
            A3
            =  add (e,c) (add (f,z) (ListBag xs))
```

□

Lemma 3.5. *if* `A1: not e 'elem' map fst xs`

```
unite (ListBag ((e,c):(xs++[(e,z)]++ys))) bagy
                = unite (ListBag (e,c+z):(xs++ys))) bagy
```

Proof. of Lemma 3.5 by structural induction on xs
`A1: not e 'elem' map fst xs`
`xs=[]`

```
unite (ListBag ((e,c):(xs++[(e,z)]++ys))) bagy
                = unite (ListBag ((e,c):([]++[(e,z)]++ys))) bagy
                = unite (ListBag ((e,c):([(e,z)]++ys))) bagy
                = unite (ListBag [(e,z)]++ys) (add (e,c) bagy)
                = unite (ListBag ys) (add (e,z) (add (e,c) bagy))
                 3.3
                =  unite (ListBag ys) (add (e,c+z) bagy)
                = unite (ListBag ((e,c+z):ys)) bagy
                = unite (ListBag ((e,c+z):(xs++ys))) bagy
```

`xs=xs'++[x]`

```
unite (ListBag ((e,c):(xs++[(e,z)]++ys))) bagy
                = unite (ListBag (xs++[(e,z)]++ys)) (add (e,c) bagy)
                = unite (ListBag (xs'++[x]++[(e,z)]++ys)) (add (e,c) bagy)
                = unite (ListBag ys) (add (e,z) (add x (add ... (add (e,c) bagy)...)))
                 3.4
                =  unite (ListBag ys) (add x (add (e,z) (add ... (add (e,c) bagy)...)))
                = unite (ListBag (xs'++[(e,z)]++[x]++ys)) (add (e,c) bagy)
                = unite (ListBag ((e,c):(xs'++[(e,z)]++[x]++ys))) bagy
                 IH
                =  unite (ListBag ((e,c+z):(xs'++[x]++ys))) bagy
                = unite (ListBag ((e,c+z):(xs++ys))) bagy
```

□

Lemma 3.6.

```
add (e,1) (unite (ListBag xs) bagy)
                = unite (add (e,1) (ListBag xs)) bagy
```

Proof. of Lemma 3.6 by structural induction on xs

```
xs=[]

add (e,1) (unite (ListBag xs) bagy)
            = add (e,1) (unite (ListBag []) bagy)
            = add (e,1) bagy
            = unite (ListBag []) (add (e,1) bagy)
            = unite (ListBag [(e,1)]) bagy
            = unite (add (e,1) (ListBag [])) bagy
            = unite (add (e,1) (ListBag xs)) bagy

xs=(x:xs')

add (e,1) (unite (ListBag xs) bagy)
            = add (e,1) (unite (ListBag (x:xs')) bagy)
            = add (e,1) (unite (ListBag xs') (add x bagy))
           IH= unite (add (e,1) (ListBag xs')) (add x bagy)
```

Case 1 : `A1: not e 'elem' map fst xs`

```
           IH= unite (add (e,1) (ListBag xs')) (add x bagy)
           A1= unite (ListBag (xs'++[(e,1)])) (add x bagy)
            = unite (ListBag (x:(xs'++[(e,1)]))) bagy
            = unite (ListBag ((x:xs')++[(e,1)])) bagy
            = unite (ListBag (xs++[(e,1)])) bagy
           A1= unite (add (e,1) (ListBag xs)) bagy
```

Case 2.1 : `A1: e 'elem' map fst xs` and `A2: x=(f,c) f!=e`
`A1` and `A2` imply `A3: e 'elem' map fst xs'`

```
           IH= unite (add (e,1) (ListBag xs')) (add x bagy)
           A3= unite (ListBag (map (inc (e,1)) xs')) (add x bagy)
            = unite (ListBag (x:(map (inc (e,1)) xs'))) bagy
           A2= unite (ListBag ((inc (e,1) x):(map (inc (e,1)) xs'))) bagy
            = unite (ListBag (map (inc (e,1)) (x:xs'))) bagy
           A1= unite (add (e,1) (ListBag xs)) bagy
```

Case 2.2 : `A1: e 'elem' map fst xs` and `A2: x=(e,c)`
multiset, `A1` and `A2` imply `A3: not e 'elem' map fst xs'`

```
           IH= unite (add (e,1) (ListBag xs')) (add x bagy)
           A3= unite (ListBag (xs'++[(e,1)])) (add x bagy)
            = unite (ListBag (x:(xs'++[(e,1)]))) bagy
            = unite (ListBag ((e,c):(xs'++[(e,1)]))) bagy
         L3.5= unite (ListBag ((e,c+1):xs')) bagy
            = unite (ListBag ((inc (e,1) (e,c)):xs')) bagy
         L3.2= unite (ListBag ((inc (e,1) x):(map (inc (e,1)) xs'))) bagy
            = unite (ListBag (map (inc (e,1)) (x:xs'))) bagy
            = unite (ListBag (map (inc (e,1)) xs)) bagy
           A1= unite (add (e,1) (ListBag xs)) bagy
```

□

Lemma 3.7. *Associativity of* `unite`

```
unite (ListBag xs) (unite (ListBag ys) (ListBag zs))
             = unite (unite (ListBag xs) (ListBag ys)) (ListBag zs)
```

Proof. by structural induction on xs

`xs=[]`

```
unite (ListBag xs) (unite (ListBag ys) (ListBag zs))
             = unite (ListBag []) (unite (ListBag ys) (ListBag zs))
             = unite (ListBag ys) (ListBag zs)
             = unite (unite (ListBag []) (ListBag ys)) (ListBag zs)
             = unite (unite (ListBag xs) (ListBag ys)) (ListBag zs)
```

`xs=(x:xs')`

```
unite (ListBag xs) (unite (ListBag ys) (ListBag zs))
             = unite (ListBag (x:xs')) (unite (ListBag ys) (ListBag zs))
             = unite (ListBag xs') (add x (unite (ListBag ys) (ListBag zs)))
          L3.6
             = unite (ListBag xs') (unite (add x (ListBag ys)) (ListBag zs))
            IH
             = unite (unite (ListBag xs') (add x (ListBag ys))) (ListBag zs)
             = unite (unite (ListBag (x:xs')) (ListBag ys)) (ListBag zs)
             = unite (unite (ListBag xs) (ListBag ys)) (ListBag zs)
```

□

Proof. of Claim 3.1 by 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)
             = bag (x:(xs'++ys))
             = add (x,1) (bag (xs'++ys))
            IH
             = add (x,1) (unite (bag xs') (bag ys))
             = add (x,1) (unite (bag xs') (bag ys))
             = unite (ListBag []) (add (x,1) (unite (bag xs') (bag ys)))
             = unite (ListBag [(x,1)]) (unite (bag xs') (bag ys))
            Ass
             = unite (unite (ListBag [(x,1)]) (bag xs')) (bag ys)
             = unite (unite (ListBag []) (add (x,1) (bag xs'))) (bag ys)
             = unite (add (x,1) (bag xs')) (bag ys)
             = unite (bag (x:xs')) (bag ys)
             = unite (bag xs) (bag ys)
```

□