1 foldl scanl

```
inits :: [a] -> [[a]]
inits []     = [[]]
inits (x:xs) = []:(map (x:) (inits xs))

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

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

Claim 1.1.

```
map (foldl f a).inits = scanl f a
```

Lemma 1.2.

```
map (foldl f a) (map (x:) ys) = map (foldl f (f a x)) ys
```

Proof. of Lemma 1.2 by structural induction on ys
ys=[]

```
map (foldl f a) (map (x:) ys)
          = map (foldl f a) (map (x:) [])
          = map (foldl f a) []
          = []
          = map (foldl f (f a x)) []
          = map (foldl f (f a x)) ys
```

ys=(y:ys')

```
map (foldl f a) (map (x:) ys)
          = map (foldl f a) (map (x:) (y:ys'))
          = map (foldl f a) ((x:y):(map (x:) ys'))
          = (foldl f a (x:y)):(map (foldl f a) (map (x:) ys'))
         IH
          = (foldl f a (x:y)):(map (foldl f (f a x)) ys')
          = (foldl f (f a x) y):(map (foldl f (f a x)) ys')
          = map (foldl f (f a x)) (y:ys')
          = map (foldl f (f a x)) ys
```

□

Proof. of Claim 1.1
we show the equivalent claim
`map (foldl f a) (inits xs) = scanl f a xs`
by structural induction on xs
`xs=[]`

```
map (foldl f a) (inits xs)
          = map (foldl f a) (inits [])
          = map (foldl f a) [[]]
          = [foldl f a []]
          = [a]
          = scanl f a []
          = scanl f a xs
```

`xs=(x:xs')`

```
map (foldl f a) (inits xs)
          = map (foldl f a) (inits (x:xs'))
          = map (foldl f a) ([]:(map (x:) (inits xs')))
          = (foldl f a []):(map (foldl f a) (map (x:) (inits xs')))
          = a:(map (foldl f a) (map (x:) (inits xs')))
        1.2
          = a:(map (foldl f (f a x)) (inits xs'))
         IH
          = a:(scanl f (f a x) xs')
          = scanl f a (x:xs')
          = scanl f a xs
```

□

2 sum prod tails

```
tails :: [a] -> [[a]]
tails []        = [[]]
tails xxs@(_:xs) = xxs:(tails xs)

sum = foldl + 0

product = foldl * 1
```

Claim 2.1.

```
sum.map product.tails = foldl f 1
                f x y = x*y+1
```

Lemma 2.2.

```
sum (x:ys) = x+(sum ys)
```

Proof. of Lemma 2.2 by structural induction on ys
ys=[]

ys=(y:ys')

□

Lemma 2.3.

```
product (x:ys) = x(product ys)
```

Proof. of Lemma 2.3 by structural induction on ys
ys=[]

ys=(y:ys')

□

Lemma 2.4.

```
foldl f (x+1) ys = x(product ys) + (foldl f 1 ys)
```

Proof. of Lemma 2.4 by structural induction on ys
ys=[]

```
foldl f (x+1) ys
           = foldl f (x+1) []
           = x+1
           = x(product []) + (foldl f 1 [])
           = x(product ys) + (foldl f 1 ys)
```

ys=(y:ys')

```
foldl f (x+1) ys
          = foldl f (x+1) (y:ys')
          = foldl f (f (x+1) y) ys'
          = foldl f (xy+y+1) ys'
         IH
          = (xy+y)(product ys') + (foldl f 1 ys')
          = xy(product ys') + y(product ys') + (foldl f 1 ys')
         IH
          = xy(product ys') + (foldl f (y+1) ys')
          = xy(product ys') + (foldl f (f 1 y) ys')
          = xy(product ys') + (foldl f 1 (y:ys'))
         2.3
          = x(product (y:ys')) + (foldl f 1 (y:ys'))
          = x(product ys) + (foldl f 1 ys)
```

□

Proof. of Claim 2.1

we show the equivalent claim

`sum (map product (tails xs)) = foldl f 1 xs`

`f x y = x*y+1`

by structural induction on xs

`xs=[]`

```
sum (map product (tails xs))
             = sum (map product (tails []))
             = sum (map product [[]])
             = sum [product []]
             = sum [1]
             = 1
             = foldl f 1 []
             = foldl f 1 xs
```

`xs=(x:xs')`

```
sum (map product (tails xs))
             = sum (map product (tails (x:xs')))
             = sum (map product ((x:xs'):(tails xs')))
             = sum ((product (x:xs')):(map product (tails xs')))
            2.2
             = (product (x:xs')) + (sum (map product (tails xs')))
            IH
             = (product (x:xs')) + (foldl f 1 xs')
            2.3
             = x(product xs') + (foldl f 1 xs')
            2.4
             = foldl f (x+1) xs'
             = foldl f (f 1 x) xs'
             = foldl f 1 (x:xs')
             = foldl f 1 xs
```

□