

```
length (xs++ys) = length xs + length ys
xsに関する帰納法で証明する
   ■ []の場合
        length ([]++ys)
             = length ys
                                       <++.1>
             = 0 + length ys
                                       <length.1>
             = length [] + length ys
   x:xsの場合
        length ((x:xs)++ys)
             = length (x:(xs++ys))
                                       <++.2>
                                       <length.2>
             = 1 + length (xs++ys)
             = 1 + length xs + length ys
             = length (x:xs) + length ys
                                       <length.2>
```

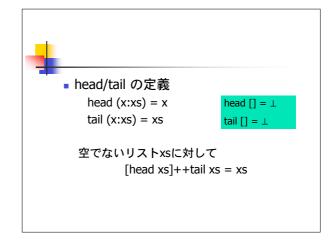
```
    ■ Zip 2引数関数:3つの場合
    zip [] ys = []
    zip (x:xs) [] = []
    zip (x:xs) (y:ys) = (x,y): zip xs ys

    ■ length (zip xs ys) = min (length xs) (length ys)
    ■ 証明: 場合1:xs=[], ys
    場合2:(x:xs), ys=[]
    場合3:(x:xs), (y:ys)
```

```
■ Take/dropの再帰的な定義
take 0 xs = []
take (n+1) [] = []
take (n+1) (x:xs) = x : take n xs

drop 0 xs = xs
drop (n+1) [] = []
drop (n+1) (x:xs) = drop n xs

■ 証明: take n xs ++ drop n xs = xs
```



```
Init/last
init [x] = []
init (x:x':xs) = x : init (x':xs)

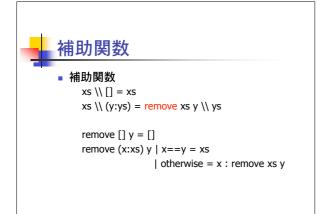
last [x] = x
last (x:x':xs) = last (x':xs)

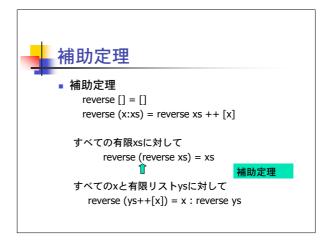
init xs = take (length xs -1 ) xs
xsに関する帰納法で証明する。
```

```
Map/filter
map f [] = []
map f (x:xs) = f x : map f xs

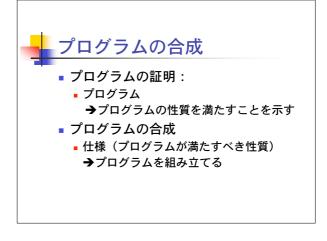
filter p [] = []
filter p (x:xs) | p x = x : filter p xs
| otherwise = filter p xs

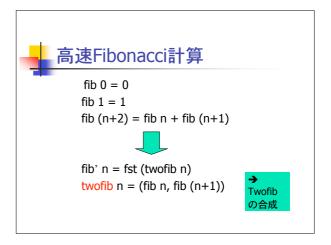
filter p (map f xs) = map f (filter (p . f) xs)
xsに関する帰納法で証明する。
```











```
twofib 0 = (fib \ 0, fib \ 1)

= (0,1)

twofib (n+1)

= (fib \ (n+1), fib \ (n+2))

= (fib \ (n+1), fib \ n + fib \ (n+1))

= (b,a+b)

where (a,b) = twofib \ n
```

