第5章　再帰法と帰納法

胡　振江

自然数の上の再帰法

- べき乗の再帰的な定義　Base

  x^0 = 1　　<^.1>

  x^(n+1) = x * x^n　　<^.2>

- Fibonacciの再帰的な定義　Recursion

  fib 0 = 0　　<fib.1>

  fib 1 = 1　　<fib.2>

  fib (n+2) = fib n + fib (n+1)　　<fib.3>

自然数の上の帰納法による証明

命題p(n)が任意の自然数nについて成立

- P(0)が成立
- P(1)が成立
- P(n),p(n-1)が成立➔P(n+1)が成立

x^(m+n) = (x^m)*(x^n)

- mについて帰納法で証明する。
  - 0の場合

    x^(0+n) = x^n　　<^.1>

    = 1 * x^n　　<*の法則>

    = x^0 * x^m　　<^.1>
  - m+1の場合

    x^((m+1)+n)

    = x^((m+n)+1)　　<+の法則>

    = x * x^(m+n)　　<^.2>

    = x * x^m * x^n　　<仮定>

    = x^(m+1) * x^n　　<^.2>

リストの上の再帰法

- リストの長さを求める関数

  length [] = 0　　base

  length (x:xs) = 1 + length xs　　recursion

- リストの連接

  [] ++ ys = ys

  (x:xs) ++ ys = x : (xs++ys)

帰納法による証明

任意の有限リストxsについてP(xs)が成立

- P[] が成立
- P[x]が成立
- P(xs)が成立➔P(x:xs)が成立

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 [] + length ys <length.1>
- x:xsの場合

  length ((x:xs)++ys)
  - = length (x:(xs++ys)) <++.2>
  - = 1 + length (xs++ys) <length.2>
  - = 1 + length xs + length ys <仮定>
  - = length (x:xs) + length ys <length.2>

リスト演算

- 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

- head/tail の定義

  head (x:xs) = x

  tail (x:xs) = xs

  head [] = ⊥

  tail [] = ⊥

  空でないリストxsに対して

  [head xs]++tail xs = xs

- Init/last

  init [x] = []

  init (x:x’:xs) = x : init (x’:xs)

  非空リストの二つの場合

  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。

補助関数

- 補助関数

  xs ¥¥ [] = xs

  xs ¥¥ (y:ys) = remove xs y ¥¥ ys

  remove [] y = []

  remove (x:xs) y | x==y = xs

  | otherwise = x : remove xs y

補助定理と一般化

- 補助定理

  reverse [] = []

  reverse (x:xs) = reverse xs ++ [x]

すべての有限xsに対して

reverse (reverse xs) = xs

すべてのxと有限リストysに対して

reverse (ys++[x]) = x : reverse ys

一般化した
補助定理

reverse (reverse xs) = xs

xsに関する帰納法で証明する。

- 場合[]:

  reverse (reverse [])

  = reverse [] <rev.1>

  = [] <rev.1>

- 場合(x:xs)

  reverse (reverse (x:xs))

  = reverse (reverse xs ++ [x]) <rev.2>

  = x : reverse (reverse xs) <ほしい>

  = x : xs <仮定>

プログラムの合成

- プログラムの証明:
  - プログラム

    ➔プログラムの性質を満たすことを示す
- プログラムの合成
  - 仕様(プログラムが満たすべき性質)

    ➔プログラムを組み立てる

Initの合成

仕様:init xs = take (length xs – 1) xs

導出:

- init [x] = take (length [x] –1) [x]

  = take 0 [x]

  = []

- init (x:x’:xs) = take (length (x:x’:xs)-1) (x:x’:xs)

  = take (2+length xs-1) (x:x’:xs)

  = take (length xs + 1) (x:x’:xs)

  = x : take (length xs) (x’:xs)

  = x : take (length (x’:xs)-1)) (x’:xs)

  = x : init (x’:xs)

証明の手順と
似ている。

高速Fibonacci計算

fib 0 = 0

fib 1 = 1

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

fib’ n = fst (twofib n)

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

➔
Twofib
の合成

```
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
```

- 効率のよいプログラム

```
fib' n = fst (twofib n)
twofib 0 = (0,1)
twofib (n+1) = (b,a+b)
      where (a,b) = twofib n
```

お知らせ

- 次回
  - 日時：12月17日、8:30-10:00
  - 場所：教育用計算機センター5階
  - 内容：演習＋中間テスト(20%)
    - テストの内容：教科書の演習問題