

再帰法と帰納法

胡 振江



自然数上の再帰法

- べき乗の再帰的な定義

$$x^0 = 1$$

$$x^{(n+1)} = x * x^n$$

Base

<^.2>

- Fibonacciの再帰的な定義

$$\text{fib } 0 = 0$$

$$\text{fib } 1 = 1$$

$$\text{fib } (n+2) = \text{fib } n + \text{fib } (n+1)$$

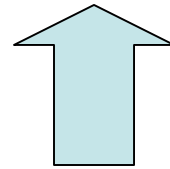
Recursion

<fib.2>

<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 $\rightarrow P(n+1)$ 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

$$\begin{aligned} x^{(0+n)} &= x^n &<^{\wedge}.1> \\ &= 1 * x^n &<*\text{の法則}> \\ &= x^0 * x^m &<^{\wedge}.1> \end{aligned}$$

- m+1の場合

$$\begin{aligned} x^{((m+1)+n)} & &<+\text{の法則}> \\ &= x^{((m+n)+1)} &<^{\wedge}.2> \\ &= x * x^{(m+n)} &<^{\wedge}.2> \\ &= x * x^m * x^n &<\text{仮定}> \\ &= x^{(m+1)} * x^n &<^{\wedge}.2> \end{aligned}$$



リスト上の再帰法

- リストの長さを求める関数

$\text{length } [] = 0$

$\text{length } (x:xs) = 1 + \text{length } xs$

base

recursion

- リストの接続

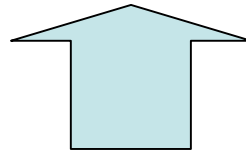
$[] ++ ys = ys$

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



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 $\Rightarrow P(x:xs)$ が成立

$$\text{length } (xs++ys) = \text{length } xs + \text{length } ys$$

xsに関する帰納法で証明する

– []の場合

$$\begin{aligned}\text{length } ([]++ys) &= \text{length } ys &<++.1> \\ &= 0 + \text{length } ys &<+> \\ &= \text{length } [] + \text{length } ys &<\text{length}.1>\end{aligned}$$

– x:xsの場合

$$\begin{aligned}\text{length } ((x:xs)++ys) &= \text{length } (x:(xs++ys)) &<++.2> \\ &= 1 + \text{length } (xs++ys) &<\text{length}.2> \\ &= 1 + \text{length } xs + \text{length } ys &<\text{仮定}> \\ &= \text{length } (x:xs) + \text{length } ys &<\text{length}.2>\end{aligned}$$



リスト演算

- Zip

2引数関数: 3つの場合

$\text{zip } [] \text{ } ys = []$

$\text{zip } (x:xs) [] = []$

$\text{zip } (x:xs) (y:ys) = (x,y) : \text{zip } xs \text{ } ys$

- $\text{length } (\text{zip } xs \text{ } ys) = \min (\text{length } xs) (\text{length } ys)$

- 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

$\text{take } 0 \text{ xs} = []$

$\text{take } (n+1) [] = []$

$\text{take } (n+1) (x:xs) = x : \text{take } n \text{ xs}$

$\text{drop } 0 \text{ xs} = \text{xs}$

$\text{drop } (n+1) [] = []$

$\text{drop } (n+1) (x:xs) = \text{drop } n \text{ xs}$



証明: $\text{take } n \text{ xs} ++ \text{drop } n \text{ xs} = \text{xs}$

- head/tail の定義

$\text{head } (x:xs) = x$

$\text{tail } (x:xs) = xs$

$\text{head } [] = \perp$

$\text{tail } [] = \perp$

空でないリストxsに対して

$[\text{head } xs]++\text{tail } xs = xs$



- 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。

非空リストの二
つの場合



- Map/filter

$\text{map } f [] = []$

$\text{map } f (x:xs) = f x : \text{map } f xs$

$\text{filter } p [] = []$

$\text{filter } p (x:xs) \mid p x = x : \text{filter } p xs$

$\mid \text{otherwise} = \text{filter } p xs$

$\text{filter } p (\text{map } f xs) = \text{map } f (\text{filter } (p \circ f) xs)$

xsに関する帰納法で証明する。



補助関数

- 補助関数

$xs \setminus [] = xs$

$xs \setminus (y:ys) = \text{remove } xs \ y \setminus ys$

$\text{remove } [] \ y = []$

$\text{remove } (x:xs) \ y \mid x==y = xs$

$\mid \text{otherwise} = x : \text{remove } xs \ y$



補助定理

- 補助定理

$\text{reverse } [] = []$

$\text{reverse } (x:xs) = \text{reverse } xs ++ [x]$

すべての有限xsに対して

$\text{reverse } (\text{reverse } xs) = xs$



すべてのxと有限リストysに対して

$\text{reverse } (ys++[x]) = x : \text{reverse } ys$

補助定理



$$\text{reverse} (\text{reverse } xs) = xs$$

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: $\text{init } xs = \text{take } (\text{length } xs - 1) \text{ } xs$

導出:

- $\text{init } [x] = \text{take } (\text{length } [x] - 1) [x]$
= $\text{take } 0 [x]$
= $[]$
- $\text{init } (x:x':xs) = \text{take } (\text{length } (x:x':xs) - 1) (x:x':xs)$
= $\text{take } (2 + \text{length } xs - 1) (x:x':xs)$
= $\text{take } (\text{length } xs + 1) (x:x':xs)$
= $x : \text{take } (\text{length } xs) (x':xs)$
= $x : \underline{\text{take } (\text{length } (x':xs) - 1) (x':xs)}$
= $x : \text{init } (x':xs)$



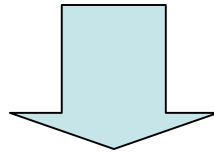
証明の手順と
似ている。

高速Fibonacci計算

$\text{fib } 0 = 0$

$\text{fib } 1 = 1$

$\text{fib } (n+2) = \text{fib } n + \text{fib } (n+1)$



$\text{fib}' n = \text{fst } (\text{twofib } n)$

$\text{twofib } n = (\text{fib } n, \text{fib } (n+1))$

→
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

$\text{fib}'\ n = \text{fst} (\text{twofib}\ n)$

$\text{twofib}\ 0 = (0, 1)$

$\text{twofib}\ (n+1) = (b, a+b)$

where $(a, b) = \text{twofib}\ n$

