リスト

胡　振江

リストの表記法

- リスト：線形に順序のついた同じ型の値の集まり

  [1,2,3] :: [Int]

  [‘h’,’e’,’l’,’l’,’o’] :: [Char]

  [[1,2],[3]] :: [[Int]]

  [(+),(-)] :: [Int→Int→Int]

  [ ] :: [a]

  [1,3..5] :: [Int]

  [1..] :: [Int]

  [1,”fine day”]　X

リストの内包表記

- 集合を記述する数学の形式を元にした構文:

  [x*x | x <- [1..10], even x]

式　　生成式　　論理式

限定式

内包表記の例

– [(a,b) | a<-[1..3], b<-[1..2]]

  ➔ [(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]

– [(i,j) | i<-[1..4], j<-[i+1..4]]

  ➔ [(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)]

– [(i,j) | i<-[1..4], even i,j<-[i+1..4],odd j]

  ➔ [(2,3)]

– [3 | j<-[1..4]]

  ➔ [3,3,3,3]

– [‘ ‘ | j <- [1..5]]

  ➔ “   “

例題

- 正の整数の約数のリストを生成する関数

```
divisors n = [d | d<-[1..n], n `mod` d==0]
```

- 2つの正整数の最大公約数を求める関数

```
gcd a b = maximum [ d | d <- divisors a,
                              b `mod` d == 0]
```

- 素数を判定する関数

```
prime n = (divisors n == [1,n])
```

例題

- 与えられた範囲内のx^2+y^2=z^2のすべて（本質的違うような）x,y,zを求める関数

```
triads n = [(x,y,z) | x<-[1..n],
                      y<-[x..n],
                      z<-[y..n],
                      x^2+y^2==z^2 ]
```

リストの演算

- リストの連接

  [1,2,3] ++ [4,5] ➔ [1,2,3,4,5]

  [1,2] ++ [] ++ [1] ➔ [1,2,1]

  – (++) :: [a] → [a] → [a]
  – 性質:
    - 結合的: (xs++ys)++zs = xs++(ys++zs)
    - 単位元: [] ++ xs = xs ++ [] = xs
  – concat :: [[a]] → [a]

    concat xss = [x | xs<-xss, x<-xs]

リスト上の関数

- リストの長さ
  - length [1,2,3] ➔ 3
  - length [] ➔ 0
  - 性質　length (xs++ys) = length xs + length ys
- リストの先頭要素と後部
  - head [1,2,3] ➔ 1　　　　　head [] = ⊥
  - tail [1,2,3] ➔ [2,3]
  - 性質　xs = [head xs] ++ tail xs

リスト上の関数

- リストの前部と末尾要素
  - init [1,2,3] ➔ [1,2]
  - last [1,2,3] ➔ 3
  - 性質　xs = init xs ++ [last xs]
- 部分リストの取り出し
  - take 3 [1..10] ➔ [1,2,3]
  - take 3 [1,2] ➔ [1,2]
  - drop 3 [1..10] ➔ [4,5,6,7,8,9,10]
  - 性質1　take m . drop n = drop n . take (m+n)
  - 性質2　drop m . drop n = drop (m+n)

リスト上の関数

- 部分リストの取り出し
  – takeWhile even [2,4,6,1,5,6] ➔ [2,4,6]
  – dropWhile even [2,4,6,1,5,6] ➔ [1,5,6]
- リストの反転
  – reverse [1,2,3,4] ➔ [4,3,2,1]
  – reverse “hello” ➔ “olleh”

リスト上の関数

- リストの綴じ合わせ
  - zip [1..3] [‘a’,’b’,’c’] ➔ [(1,’a’),(2,’b’),(3,’c’)]
  - zipWith f xs ys = [f x y | (x,y) <- zip xs ys]

例（内積の計算）:

    sp (xs,ys) = sum [x*y | (x,y) <- zip xs ys]
    sp (xs,ys) = sum (zipWith (*) xs ys)

例（位置の計算）

    position xs x = [i | (i,y) <- zip [0..length xs-1] xs, x==y]

リスト上の関数

- リストの番号づけ
  - [2,4,6,8] !! 2 ➔ 6

  例（非減少判定）

```
nondec xs = and [ xs!!k <= xs !! (k+1)
                  | k <- [0 .. length xs – 2 ]]
```

- リストの差
  - [1,2,1,3,1,3] \\ [1,3] ➔ [2,1,1,3]
  - 注：List.hsをloadする必要がある。

高階関数map

- 関数mapは関数をリストのそれぞれの要素に適用する。
  - 定義： map f xs = [f x | x<-xs]
  - 例： map square [1,2,3] ➔ [1,4,9]

    sum (map square [1..100]) ➔ ?
  - 性質：

    map (f.g) = map f . map g

    map f (xs++ys) = map f xs ++ map f ys

    map f . concat = concat . map (map f)

高階関数filter

- 関数filterは述語pとリストxsを引数にとり、要素がpを満たすような部分リストを返す。
  - 定義: filter p xs = [x | x<-xs, p x]
  - 例:　filter even [1,2,4,5,32] ➔ [2,4,32]
  - 性質:

    filter p . filter q = filter q . filter p

    filter p (xs++ys) = filter p xs ++ filter p ys

    filter p . concat = concat . map (filter p)

内包表記の翻訳

- 内包表記 ➔ map, filter での表記
- 規則:
  - [x | x<-xs] ➔ xs
  - [f x | x <- xs] ➔ map f xs
  - [e | x<-xs, p x, …]

    ➔ [e | x <- filter p xs, …]
  - [e | x <-xs, y<-ys, …]

    ➔ concat [[e | y<-ys,…] | x<-xs]

翻訳の例

```
[ 1 | x <- xs ]
➔[ const 1 x | x <- xs ]          const k x = k
➔map (const 1)

[ x*x | x <- xs, even x]
➔[ x*x | x <- filter even xs ]
➔[ square x | x<-filter even xs]  square x = x*x
➔map square (filter even xs)
```

高階関数fold (1/2)

- 畳み込み関数foldはリストを他の種類の値に変える事が出来る。
  - 右側畳み込みfoldr

    foldr (⊕) a [x1,x2,…,xn] = x1⊕(x2⊕(…(xn⊕a)))

```
s = a;
for ( i=n; i>=1; i-- ) {
    s = x[i] + s
}
```

高階関数fold (2/2)

– 左側畳み込みfoldl

foldl (⊕) a [x1,x2,…,xn] = (((a⊕x1)⊕x2)…⊕ xn)

```
s = a;
for ( i=1; 1<=n, i++ ) {
    s = s + x[i]
}
```

例

- 簡単な例

```
sum = foldr (+) 0
product = foldr (*) 1
concat = foldr (++) []
and = foldr (&&) True
or = foldr (||) False
```

- リストの反転

```
reverse = foldr postfix []
    where postfix x xs = xs ++ [x]
```

- [xn,…,x0] ➔ xn*10^n + … + x0

```
pack xs = foldl oplus 0 xs
  where n `oplus` x = 10*n + x
```

畳み込み演算の性質

- 第一双対定理

  演算子⊕とeが単位半群:

  $$x \oplus (y \oplus z) = (x \oplus y) \oplus z$$

  $$e \oplus x = x \oplus e = x$$

  xsが有限リスト

$$\Downarrow$$

$$\text{foldr } (\oplus)\ e\ xs = \text{foldl } (\oplus)\ e\ xs$$

畳み込み演算の性質

- 第二双対定理

$$x \oplus (y \otimes z) = (x \oplus y) \otimes z$$

$$x \oplus e = e \otimes x$$

xsが有限リスト

$$\Downarrow$$

$$\text{foldr } (\oplus)\ e\ xs = \text{foldl } (\otimes)\ e\ xs$$

畳み込み演算の性質

- 第三双対定理

$$\mathrm{foldr}\ (\oplus)\ e\ xs = \mathrm{foldl}\ (\otimes)\ e\ (\mathrm{reverse}\ xs)$$

$$\text{where } x \otimes y = y \oplus x$$

空でないリストの畳み込み

- foldr1

  foldr1 (⊕) [x1,x2,…,xn] = x1⊕(x2⊕(…⊕xn))

- foldl1

  foldl1 (⊕) [x1,x2,…,xn] = ((x1⊕x2)…)⊕xn

- 例

  maximum xs = foldr1 max xs

  = foldl1 max xs

リストの走査関数scanl, scanr

- 左(右)側の走査

  scanl (⊕) a [x1,x2,…,xn]

    = [a,

      a ⊕ x1,

      (a ⊕ x1) ⊕ x2,

      …,

      ((a⊕x1)⊕x2)…⊕ xn]

- 例:
  - 累積和: scanl (+) 0 [12,3,4,5] ➔ [0,1,3,6,10,15]
  - 累乗積: scanl (*) 1 [1,2,3,4,5] ➔ [1,1,2,6,24,120]

リストのパターン

- リストの構成
  - 構成子　[]
    - 空リストを生成する
  - 構成子　(:)
    - リストの新たら第一要素として新しい値を挿入する

1:2:3:4:[]　↔　[1,2,3,4]

リスト上の関数の定義

```
null [] = True
null (x:xs) = False

length [] = 0
length (x:xs) = 1 + length xs

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