Haskell ゼミ

プログラミング Haskell

第 5 章 リスト内包表記

Fumiri USAMI

理学部　情報科学科
小口研究室

2016 年 11 月 1 日

アウトライン

アウトライン

よく見る数学の記法をそのまま使える！

例) 1 から 5 までの平方数の集合

$$\{x^2 \mid x \in \{1...5\}\} = \{1, 4, 9, 16, 25\}$$

そのまま Haskell で書いてみましょう

例) 1 から 5 までの平方数の集合

```
> [x^2 | x <- [1..5]]
[1, 4, 9, 16, 25]
```

「x ← [1..5]」を 生成器 (generator) という
つまり ∈ = ← みたいなもの

生成器いろいろ

- カンマで区切れば複数の generator を列挙できる
- generator の順番を入れ替えると、要素の順番が変わる
- 後ろの方が入れ子が深い
- 後ろの generator は前方の generator の使う変数を利用できる

例) [1,2,3] の要素と [4,5] の要素からなる組の集合

```
> [(x,y) | x <- [1..5], y <- [4,5]]
[(1,4),(1,5),(2,4),(2,5),(3,4),(3,5),(4,4),
(4,5),(5,4),(5,5)]

> [(x,y) | y <- [4,5], x <- [1..5]]
[(1,4),(2,4),(3,4),(4,4),(5,4),(1,5),(2,5),
(3,5),(4,5),(5,5)]
```

生成器いろいろ

例) [1..3] の要素から重複のない組の順列

```
> [(x,y) | x <- [1..3], y <- [x..3]]
[(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)]
```

ライブラリ関数 concat

```
concat     :: [[a]] -> [a]
concat xss = [x | xs <- xss, x <- xs]
```

concat 使用例

```
> concat [[x^2 | x <- [1..5]], [1..5]]
[1,4,9,16,25,1,2,3,4,5]
```

ワイルドカードで捨てちゃおう

組のリストから組の先頭のリストを作る関数 firsts

```
firsts    :: [(a, b)] -> [a]
firsts ps = [x | (x, _) <- ps]
```

リストの長さを計算する関数 length

```
length    :: [a] -> Int
length xs = sum [1 | _ <- xs]
```

ちなみに OCaml **だと**

```
let firsts lst = List.map fst lst
let length lst = List.fold_left
                   (fun n _ -> n+1) 0 lst
```

アウトライン

ガードとは

例) ある自然数 n の約数の集合

$$\{x \mid x \in \{1...n\} \ \wedge \ n \equiv 0 \ (mod \ x)\}$$

約数のリストを作る関数 factors

```
factors   :: Int -> [Int]
factors n = [x | x <- [1..n], n `mod` x == 0]
```

生成された値を間引くための論理式をガード (guard) **という**

… 条件を満たす場合にしか先の処理に進ませてくれない衛兵さん

上の場合、ガードは 「 `n `mod` x == 0` **」の部分**

factors を使って、素数判定関数を考えよう

素数の定義: 1 **より大きな整数で、約数が** 1 **と自分自身のみ**

$$\{x \in \mathbb{Z} \mid x > 1 \ \wedge \ x \text{ の約数} = \{1, x\}\}$$

これを Haskell **で書くと**

```
prime   :: Int -> Bool
prime n = factors n == [1, n]
```

使用例

```
> prime 15
  -- 遅延評価なので、約数 3 が出てきた瞬間に False が返る
False
> prime 7
True
```

素数判定をする関数 prime **を使って、**
整数 n **までの素数全てを生成する関数** primes **を定義しよう**

整数 n **までの素数の集合は次のように表せる**:

$$\{x \in \mathbb{Z} \mid x \in \{2, ..., n\} \ \wedge \ x \textbf{は素数}\}$$

Haskell **で書くと**

```
primes  :: Int -> Bool
primes n = [x | x <- [2..n], prime x]
```

使用例

```
> primes 40
[2,3,5,7,11,13,17,19,23,29,31,37]
```

cf. 12 **章「エラトステネスのふるい」**

ガードを使って探索しよう

キーと値からなる組のリスト lst **から、検索キー** k **と等しいキーを持つ組を全て探してきて値のリストを返す関数** find

Haskell **で書くと**

```
find          :: Eq a => a -> [(a, b)] -> [b]
find k lst = [v | (k', v) <- lst, k == k']
```

OCaml **の場合 (一例)**

```
# let find k lst = List.map snd
    (List.filter (fun (k',v) -> k == k') lst)
  in
  find 'b'
    [('a',1); ('b',2); ('c',3); ('b',4)];;
- : int list = [2; 4]
```

アウトライン

zip とは

関数 zip

2 つのリストを取り、対応する要素を組にして、1 つのリストを作る
2 引数のうち、短い方のリストの長さになる

```
> :t zip
zip :: [a] -> [b] -> [(a, b)]
> zip ['a','b', 'c'] [1, 2, 3, 4]
[('a',1),('b',2),('c',3)]
```

リスト内包表記と一緒に使うと便利

例) リストを受け取り、隣り合う要素を組にしたリストを作る

隣り合う要素を組にしたリストを返す関数 pairs

```
pairs    :: [a] -> [(a, a)]
pairs xs = zip xs (tail xs)

> pairs [1,2,3,4]
[(1,2),(2,3),(3,4)]
```

pairs **を利用すると、リストの整列を判定できる**

※ **順序クラスに属する任意の型の要素のリスト**

→ 隣接する要素が順番通りか調べればよい

リストの中が並んでいるか判定する関数 sorted

```
sorted    :: Ord a => [a] -> Bool
sorted xs = and [x <= y | (x,y) <- pairs xs]

> sorted [1,2,3,4]
True
> sorted [1,3,2,4]
False   -- 順番が違っていたらすぐ False を返す。ここでは (3,2) のとき
```

ある値が、リストの何番目にあるかを知ることもできる

例えば、[0,0,1,0,1] **から** 1 **の位置を知りたいとき、**
3 **番目と** 5 **番目に** 1 **があるので、**[3,5] **が返ってくる**

やり方としては

1. zip で [0,0,1,0,1] と [0..4] をペアにする
2. ペアのリスト [(0,0), (0,1), .. , (1,4)] から、先頭要素が 1 であるペアの第二要素を取り出す

ある値がリストのどの位置にあるか調べて、その位置すべてをリストとして返す関数 positions

```
positions      :: Eq a => a -> [a] -> [Int]
positions x xs =
  [i | (x', i) <- zip xs [0..n], x == x']
  where n = length xs - 1

> positions 1 [0,0,1,0,1]
[2,4]
```

アウトライン

Haskell の文字列は文字のリスト

Haskell の文字列は単なる 文字のリスト

```
> "abc" == ['a','b','c']
True
```

リストを扱う多層関数は、文字列も扱える

```
> "abcde"  2  -- リストのインデックスが 2 の要素を返す
'c'
> take 3 "abcde"  -- リストの先頭から 3 番目までを返す
"abc"
> length "abcde"  -- リストの長さを返す
5
> zip "abc" [1..5]  -- 二つのリストをペアのリストにする
[('a',1),('b',2),('c',3)]
```

文字列とリスト内包表記

文字列のうち小文字の個数を数える関数 lowers

```
lowers    :: String -> Int
lowers xs = length [x | x <- xs, isLower x]
  -- Char 型の小文字判定関数 isLower を使う時は import Data.Char すること
> lowers "OCaml"
3
```

特定の文字 x の個数を数える関数 count

```
count       :: Char -> String -> Int
count x xs = length [x' | x' <- xs, x == x']
> count 'も' "すももももももものうち"
8
```

アウトライン

シーザー暗号とは

シーザー暗号

平文の各文字を辞書順に 3 **文字ずらして暗号文を作る方法**
この方法を使用した Julius Caesar **にちなむ**

シーザー暗号の例

- PPAP → RRDR (3 **個ずらす**)
- PenPineappleApplePen → VktVotkgvvrkGvvrkVkt (6 **個**)
- PenPineappleApplePen → ApyAtyplaawpLaawpApy (11 **個**)

Haskell **でシーザー暗号の実装を考えていく**

シーザー暗号の実装

アルファベット小文字の暗号化を考える

- **文字を 0〜25 に変換する関数 l2i と、逆関数 i2l の定義**

```
l2i c = ord c - ord 'a'          -- let2int
    -- ord:  ascii コードの番号を返す関数
i2l n = chr (ord 'a' + n)   -- int2let
    -- chr:  ord の逆。対応する Char 型を返す
```

- **小文字をシフト数だけずらす関数 shift**

  文字を 0〜25 に変換したあと、n だけずらして 26 で割ったものを Char に戻す

  シフト数 n が負の場合にも対応

```
shift n c | isLower c = i2l ((l2i c + n) `mod` 26)
          | otherwise = c
```

文字列の内包表記で shift を使えば、シーザー暗号が作れる

- **与えられたシフト数で文字列を暗号化する関数** encode

```
encode n xs = [shift n x | x <- xs]
```

- **使用例**

```
> encode 3 "pen␣pineapple␣apple␣pen"
"shq␣slqhdssoh␣dssoh␣shq"
> encode (-3) "shq␣slqhdssoh␣dssoh␣shq"
"pen␣pineapple␣apple␣pen"
```

文字の出現頻度表

英語の文章では、使用頻度が特定の文字に偏っている
一般に e の使用頻度が最高、q や z の使用頻度が最低

```
-- fromIntegral は整数を浮動小数点数に変換する関数
-- 百分率を計算する関数 percent
percent n m =
  (fromIntegral n / fromIntegral m) * 100
-- 任意の文字列に対して文字の出現頻度表を返す関数 freqs
freqs xs =
 [percent (count x xs) n | x <- ['a'..'z']]
 where n = lowers xs

> freqs "eggs"
[0.0,0.0,0.0,0.0,25.0,0.0,50.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,25.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
```

暗号解読

暗号文から、元の平文に復号したい！
→ 文字の出現頻度の偏りに着目

```
table = [8,2,3,4,12,2,2,6,7,0.2,0.8,
4,2.4,7.2,8,2,0.1,6,6,9,3,1,2,0.2,2,0.1]
```

暗号文をいくつシフトさせれば復号できるか？

- 暗号文に対する文字の出現頻度表を作る
- 出現頻度表を左に回転させながら、期待される文字の表に対するカイ二乗検定の値を計算する
- 算出した値のリストのうち、最小の値の位置をシフト数とする

(カイ二乗検定: 一般に、実測値と理論値にどのくらい差があるのか知りたいときに使われる)

カイ二乗検定

os:**観測頻度のリスト**, es:**期待頻度のリスト**, n:**リストの長さ**

値が小さければ小さいほど二つのリストは似ている

$$\chi^2 = \sum_{i=0}^{n-1} \frac{(os_i - es_i)^2}{es_{ij}}$$

Haskell **で書くと**

```
chisqr os es =
  sum [((o-e)^2)/ e | (o,e) <- zip os es]
```

リストの要素を n **だけ左に回転させる関数** rotate

```
rotate n xs = drop n xs ++ take n xs
-- drop n xs:  xs の n 番目までを捨てて残った後ろのリストを返す
```

暗号解読例

```
table' = freqs "kdvnhoo lv ixq"
> chisqr (rotate n table') table | n <- [0..25]]
[1406.509788359788,639.314021164021,625.9487213403879,205.69960317460308,
1446.4679012345678,4249.598412698412,633.5269841269841,1158.0805555555555,
970.4152116402114,1013.4046296296294,487.05983245149906,1505.3027777777775,
2283.6592592592588,1359.800132275132,1492.7641975308638,3027.569091710757,
670.3160052910052,2846.9308641975304,1012.4401234567899,799.2842592592591,
1262.5227954144618,852.8832451499115,2899.9787037037026,952.9163139329804,
5320.200264550263,647.1678571428572]
```

-- どうやら 3 番目が一番値が小さいので、3 つシフトさせる

```
> encode (-3) "kdvnhoo lv ixq"
"haskell is fun"
```

いままでの手順をまとめると

```
crack xs = encode (-factor) xs
  where
    factor = head (positions (minimum chitab) chitab)
    chitab = [chisqr (rotate n table') table | n <- [0..25]]
    table' = freqs xs
```

ただし、文字列が短かったり、文字の出現頻度が例外的だと解読できない

```
> crack (encode 3 "haskell")
> crack (encode 5 "boxing␣wizards␣jump␣quickly")
```

APPENDIX

アウトライン

第五章 答え合わせ

```
q1 = sum [x^2|x<-[1..100]] -- 338350

-- q2
replicate n x = [x | _ <- [1..n]]
-- > replicate 0 0
-- []
-- > replicate 8 20
-- [8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8]

-- q3
pyth n = [(x,y,z) |
    x <- [1..n], y <- [1..n], z <- [1..n], x*x+y*y==z*z]
-- 重複をさけるもの
pyth' n = [(x,y,z) |
    x <- [1..n], y <- [x..n], z <- [y..n], x*x+y*y==z*z]
-- > pyth' 20
-- [(3,4,5),(5,12,13),(6,8,10),(8,15,17),(9,12,15),(12,16,20)]

-- q4
yakusu n = [x | x <- [1..n], n `mod` x == 0]
perfects n = [x | x <- [1..n], x+x == sum (yakusu x)]
-- > perfects 500
-- [6,28,496]
```

第五章 答え合わせ

```
-- q5
lst1 = [(x,y) | x <- [1,2,3], y <- [4,5,6]]
lst2 = concat [[(x,y)|y <- [4,5,6]] | x <- [1,2,3]]
q5 = lst1 == lst2

-- q6
find k lst = [v | (k', v) <- lst, k == k']
positions x xs =
    [n | n <- (find x (zip xs [1..(length xs)]))]

-- q7
scalarproduct xs ys = sum [x*y | (x,y) <- (zip xs ys)]

-- q8
l2i' x c = ord c - ord x       -- let2int
i2l' x n = chr (ord x + n)  -- int2let
shift' n c | isLower c = i2l' 'a' ((l2i' 'a' c + n) `mod` 26)
           | isUpper c = i2l' 'A' ((l2i' 'A' c + n) `mod` 26)
           | otherwise = c
encode' n xs = [shift' n x | x <- xs]
ppap = "PenPineappleApplePen"
```

pdf や hs ファイルなど
https://github.com/fuminashi/asai/blob/master/haskell/haskell5.pdf

東北大学住井研究室の tex **ファイルをお借りしました。**
ありがとうございます！
https://github.com/fetburner/sumiilab-tex