再帰の鳥瞰図

山本和彦
@kazu_yamamoto

お品書き

- 末尾再帰
- 一般的な再帰
- 節度

末尾再帰

- 場合分けの末端で、自分自身を呼び出す
  - Haskell だと = の右が自分自身

```
gcd :: Int -> Int -> Int
gcd a 0 = a
gcd a b = gcd b (a ‘mod‘ b)
```

一般的な再帰

- 場合分けの末端で、他の関数を呼び出す

```
sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs
```

- 最後の行はこう書ける

```
sum (x:xs) = (+) x (sum xs)
```

末尾再帰あれこれ

- 末尾再帰はループと同じ力を持つ
  - ループで書けるものは末尾再帰で書ける
- 正格評価でも末尾再帰呼び出しはジャンプに変換できる
  - コンパイラーがサポートしていればの話
  - スタックは消費されない

一般的な再帰から末尾再帰へ (1)

- ループで書けるようなコードなのに
  一般的な再帰になっている場合は
  引数を増やすことで末尾再帰にできる
- 一般的な再帰での和

```
sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs
```

- 末尾再帰での和

```
sum :: Num a => [a] -> a
sum is = sum’ is 0
  where
    sum’ []     a = a
    sum’ (x:xs) a = sum’ xs (x+a)
```

一般的な再帰から末尾再帰へ (2)

- 一般的な再帰でのフィボナッチ数
  - 注) = の右側は (+)

```
fib :: Int -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

- 末尾再帰でのフィボナッチ数

```
fib :: Int -> Integer
fib n = fib’ n 0 1
  where
    fib’ 0 x _ = x
    fib’ m x y = fib’ (m - 1) y (x + y)
```

末尾再帰では書けない一般的な再帰 (1)

- 一つ前ができているとすると、次はどうする？
- リストの連結

```
(++) :: [a] -> [a] -> [a]
(++) []     ys = ys
(++) (x:xs) ys = x : (xs ++ ys)
```

末尾再帰では書けない一般的な再帰 (2)

- 利用例

```
break (> 3) [1,2,3,4,1,2,3,4]
→ ([1,2,3],[4,1,2,3,4])
break (< 9) [1,2,3]
→ ([],[1,2,3])
break (> 9) [1,2,3]
→ ([1,2,3],[])
```

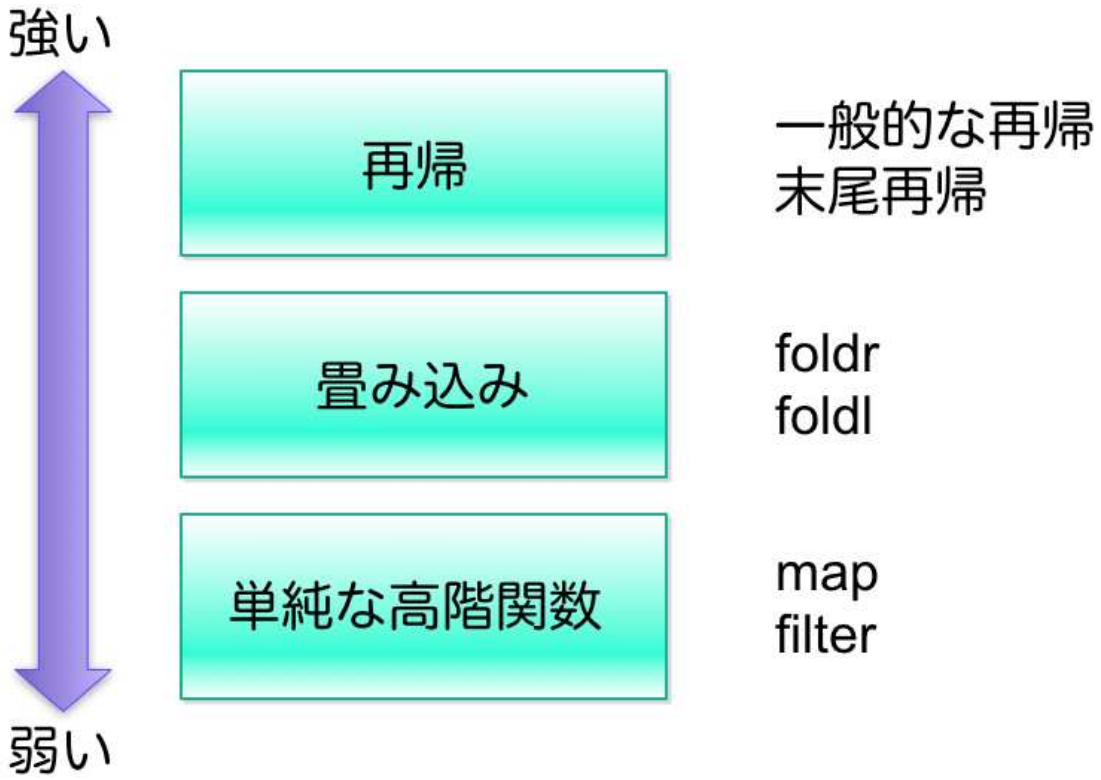
- リストの分割

```
break :: (a -> Bool) -> [a] -> ([a], [a])
break _ []     =  ([], [])
break p xxs@(x:xs)
  | p x        =  ([],xxs)
  | otherwise  =  (x:ys,zs)
  where
    (ys,zs) = break p xs
```

面接に出る一般的な再帰

- コインの両替問題
- 二分木のパターン数 (カタラン数)

力の階層と節度



なるべく力の弱いものを使え！

どちらが分かりやすい？

- 再帰で書く

```
sumEven :: [Int] -> Int
sumEven xs = sumEven' xs 0
  where
    sumEven' []     r = r
    sumEven' (y:ys) r
      | even y        = sumEven' ys (y + r)
      | otherwise     = sumEven' ys r
```

- 節度を守って書く

```
sumEven :: [Int] -> Int
sumEven = foldl (+) 0 . filter even
```