関数型プログラミング
第5回　遅延評価

萩野　達也

hagino@sfc.keio.ac.jp

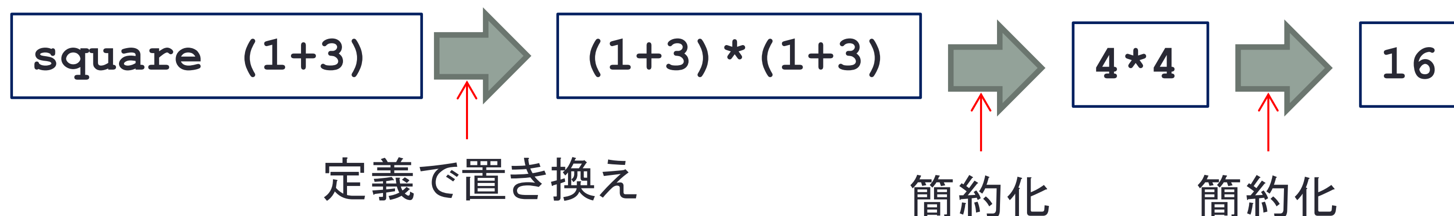
Haskellにおける評価

- 評価とは
  - Haskellでは与えられた式を**評価**することがプログラムの実行に相当します.
  - 評価とは，規則に従い値を計算することです.

- 置き換えモデル
  - 評価は，関数の適用を定義で**置き換え**ていくことで行われます.

```
square n = n * n
```

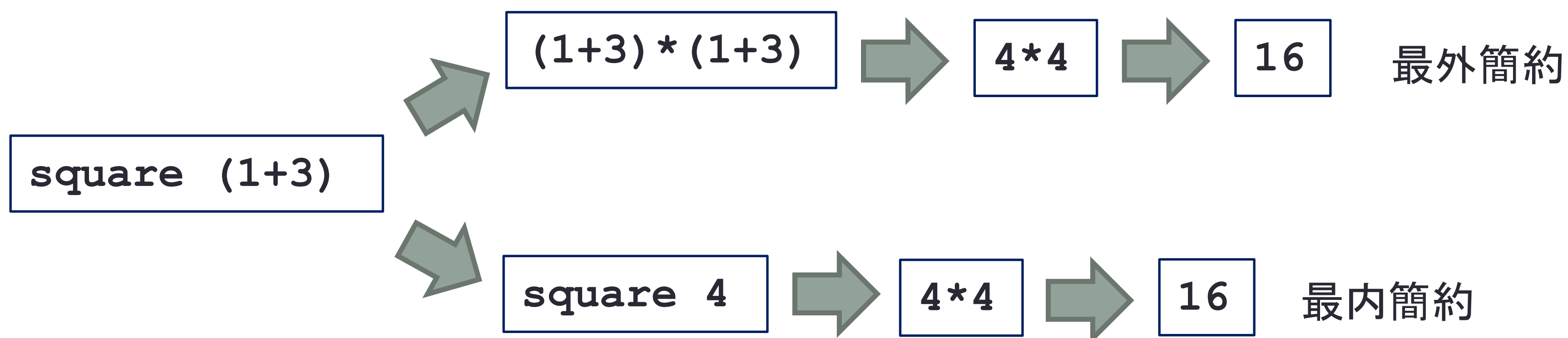
関数名　　定義



最内簡約と最外簡約

- 最内簡約
  - 内側から先に簡約化する.
  - 関数の引数を簡約化してから，関数の適用を展開する.

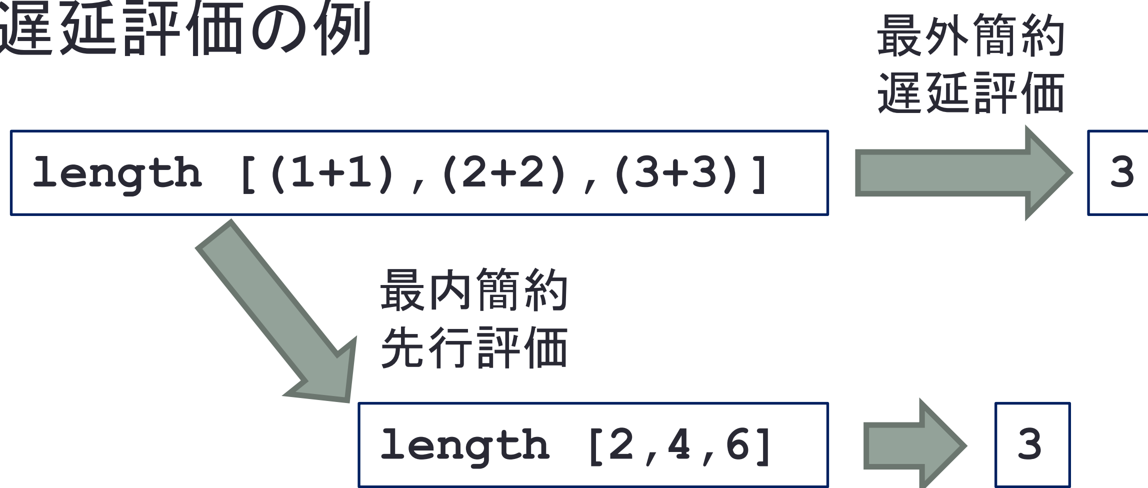
- 最外簡約
  - 外側から先に簡約化する.
  - 引数を簡約化せずに，関数の適用を展開する.



最外簡約と遅延評価

- Haskellでは最外簡約によって式の評価を行う.
  - なるべく評価を後で行うので遅延評価(lazy evaluation)になっている

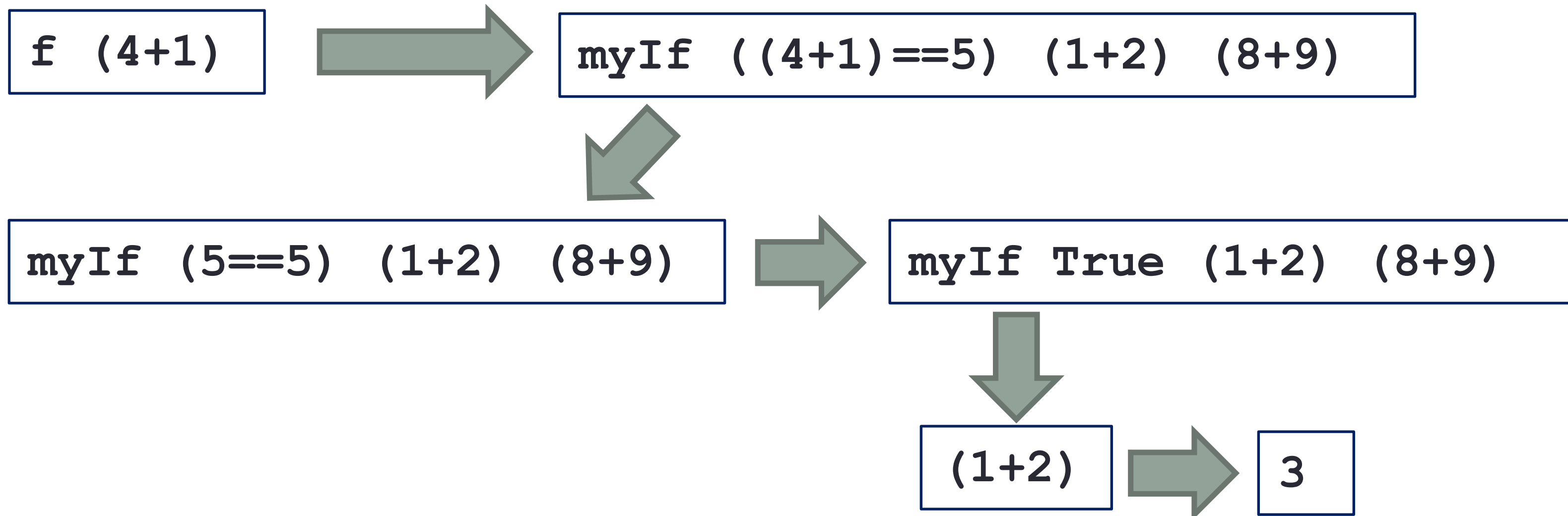
- 遅延評価の例



遅延評価の例

```
myIf :: Bool -> a -> a -> a
myIf True  t e = t
myIf False t e = e

f n = myIf (n==5) (1+2) (8+9)
```



遅延評価の利点・欠点

- 利点
  - 不要な計算を減らすことができる
  - 無限の長さのリストを扱うことができる
    - `ints n = n : (ints (n + 1))`
  - インターフェイスを統一できる
    - 「木構造→リスト→処理」としてもリストが実際に一度に作成されるわけではない
    - リスト処理に関するものを用意することで統一できる
- 欠点
  - 思った順番で操作を実行することが難しい
  - デバッグしにくい
    - C言語のようしスタックトレースを出すことができない

練習問題5ー1

- たらいまわし関数をHaskellとC言語で作ってみて実行時間を比較しなさい.

tarai.hs

```
main = print (tarai 20 10 5)

tarai :: Int -> Int -> Int -> Int
tarai x y z = if x <= y then y
                       else tarai (tarai (x-1) y z)
                                  (tarai (y-1) z x)
                                  (tarai (z-1) x y)
```

tarai.c

```
#include <stdio.h>

int tarai(int x, int y, int z) {
  if (x <= y) return y;
  else return tarai(tarai(x-1,y,z),tarai(y-1,z,x),tarai(z-1,x,y));
}

main() {
  printf("%d¥n", tarai(20,10,5));
}
```

練習問題5ー2

- 無限数列「1,2,3,4,5,6,7,8,9,10,.....」作成し，その先頭の20個を出力しなさい．

ints.hs

```
main = print $ take 20 $ ints 1

ints n = n:(ints(n+1))
```

- 「ints 1」は「[1..]」と書くことができる．

ints2.hs

```
main = print $ take 20 [1..]
```

練習問題5ー3

- 奇数だけの無限数列を作りなさい.
  - 1, 3, 5, 7, 9, 11, ...
- 先頭の20個を出力しなさい.

odds.hs

```
main = print $ take 20 $ odds 1

odds n =
```

- 無限数列 [1..]から奇数だけ選び出すことで同じことをしなさい.

odds2.hs

```
main = print $ take 20 $ filter ...
```

練習問題5ー4

- フィボナッチ数列 $f_0, f_1, f_2, \cdots$ とは
  - $f_0 = 1$
  - $f_1 = 1$
  - $f_n = f_{n-1} + f_{n-2}$
- フィボナッチ数列の先頭の20個を出力しなさい.

fib.hs

```
main = print $ take 20 $ map fib [0..]

fib 0 = 1
fib 1 = 1
fib n =
```

- 先頭の100個を出力するとどうなりますか？

練習問題5－5

- 練習問題5－4のフィボナッチ数列では個別に`fib`を呼び出すため，効率が悪くなっていました．
- フィボナッチ数列の無限数列を作ると良いかもしれません．

fibs.hs

```
main = print $ take 100 $ fibs ...

fibs x y = x:(fibs y (x+y))
```

練習問題5−6

- 与えられた数の約数を出力するプログラムを書いてみましょう.
- 数は引数として与えられます.

factor.hs

```
import System.Environment

main = do args <- getArgs
          print $ factors $ read $ head args

factors n = filter divisible [1..n]
  where divisible m = ...
```

練習問題5－7

- 素数は，自身と1以外では割り切れない数のことです．
- 2から始まる最初の100個の素数を出力しなさい．

prime.hs

```
main = print $ take 100 $ filter isprime [2..]

factors n =

isprime n =
```

練習問題5−8

- エラトステネスのふるいを使って素数表を効率よく作ってみましょう.
  - 最初は2以上のすべての数をリストとして用意しておきます.
    - 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, ...
  - 先頭の数2を素数としてマークし，残りのリストから2の倍数を取り除きます.
    - (2), 3, 5, 7, 9, 11, 13, 15, 17, ...
  - 残りのリストの先頭の数3を素数としてマークし，残りのリストから3の倍数を取り除きます.
    - (2), (3), 5, 7, 11, 13, 17, ...
  - 残りのリストの先頭の数5を素数としてマークし，残りのリストから5の倍数を取り除きます.
    - (2), (3), (5), 7, 11, 13, 17, ...
  - .....
- 素数の無限列を作り，その先頭の1000個を出力しなさい.

sieve.hs

```
main = print $ take 1000 $ primes [2..]

primes (x:xs) = ...
```

- `primes(x:xs)` は`x`を素数とし，`xs`から`x`の倍数を取り除き，次の`primes`に渡します.

練習問題5－9

- 与えられた数を素因数分解するプログラムを書きなさい．
- 例えば，300が与えられると

  [2,2,3,5,5]

  と出力します．

factoring.hs

```
import System.Environment

main = do args <- getArgs
          print $ factoring $ read $ head args

factoring n = ...
```