関数型プログラミング

第8回　遅延評価

萩野　達也
hagino@sfc.keio.ac.jp

Slide URL

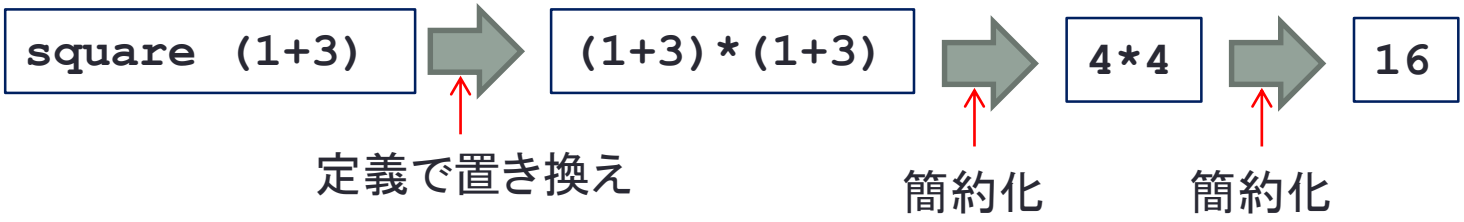
https://vu5.sfc.keio.ac.jp/slide/

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.

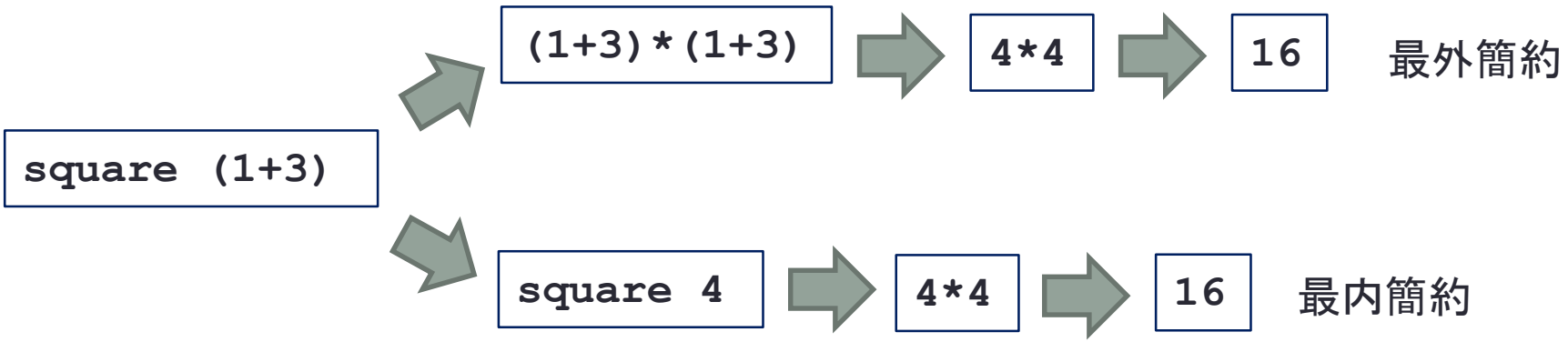
```
square n = n * n
```

関数名　　定義



最内簡約と最外簡約

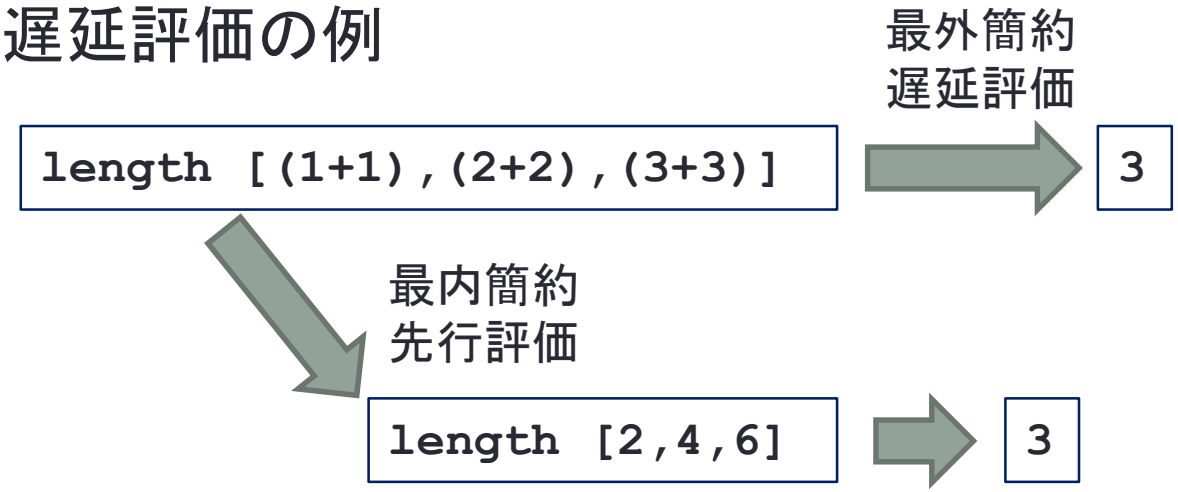
- 最内簡約
  - 内側から先に簡約化する.
  - 関数の引数を簡約化してから，関数の適用を展開する.
- 最外簡約
  - 外側から先に簡約化する.
  - 引数を簡約化せずに，関数の適用を展開する.



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

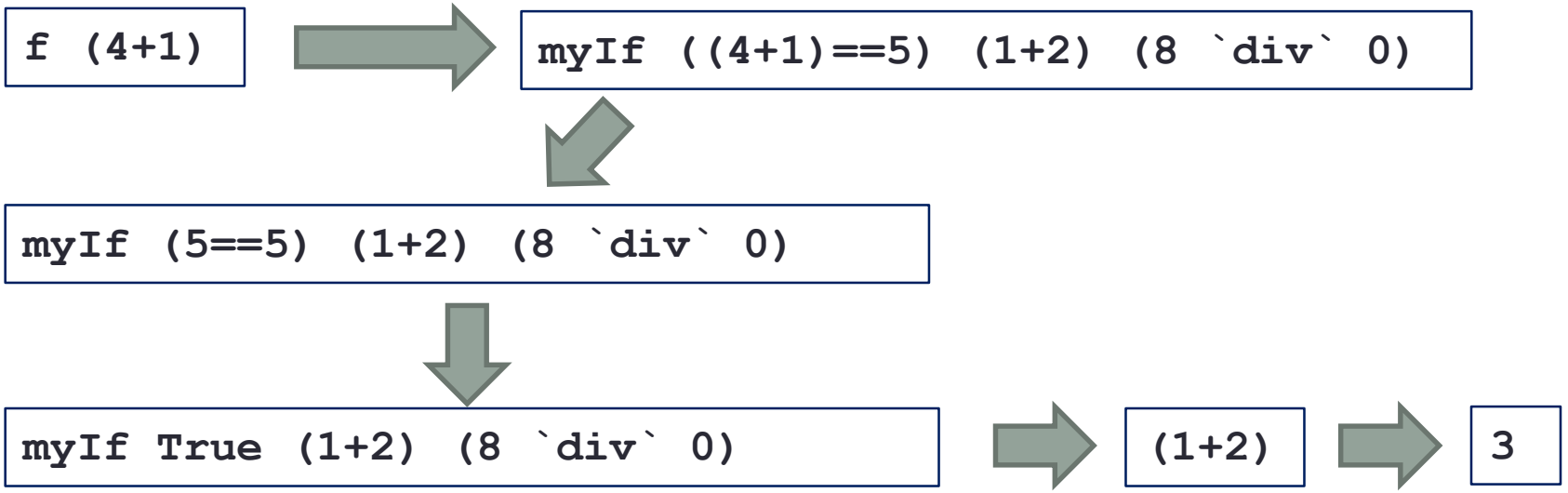
- 遅延評価の例



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 `div` 0)
```



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言語のようにスタックトレースを出すことができない

練習問題8－1

- 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8－2

- 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8－3

- 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 ...
```

- [1,3..] と書くことも可能です.

練習問題8－4

- 次のように整数の無限数列を作りなさい.
  - $0, 1, -1, 2, -2, 3, -3, 4, -4, \cdots\cdots$

```
infSeq ...

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

練習問題8－5

- 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8－6

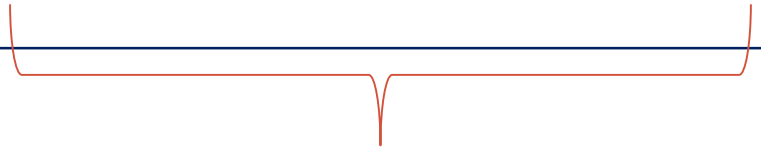
- フィボナッチ数列 $f_0, f_1, f_2, \cdots$ は
  - $(f_2, f_3, f_4, \cdots) = (f_0, f_1, f_2, \cdots) + (f_1, f_2, f_3, \cdots)$

  であることを使って，フィボナッチ数列の無限リストを効率よく計算するようにしなさい．

fib2.hs

```
main = print $ take 100 fib

fib = 1 : 1 : ...
```



$(f_2, f_3, f_4, \cdots)$

練習問題8－7

- 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..]
  where {
    isprime n = ...
  }
```

```
% stack runghc prime.hs
[2,3,5,7,...,541]
```

エラトステネスのふるい

- 素数のリストを作る効率の良い方法としてエラトステネスのふるいがあります.
  - まず，2から始まる数字を書き並べます.

2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,…

  - 先頭の2を素数として○を付け，残りの2の倍数は素数でないとして斜線を引きます.

  - 次の3を素数として○を付け，残りの3の倍数は素数でないとして斜線を引きます.

  - 次の5を素数として○を付け，残りの5の倍数は素数でないとして斜線を引きます.

  - このように，残った数の先頭を素数として○を付け，残りからその倍数を素数でないとして消すという操作を繰り返します.
  - 残ったのが，素数のリストです.

練習問題8－8

- エラトステネスのふるいの方法で素数の無限リストを作成しなさい.

seive.hs

```
main = print $ take 100 $ primes [2..]

sieve x xs = filter notdiv xs
  where notdiv y = ...

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

```
% stack runghc sieve.hs
[2,3,5,7,...,541]
```

練習問題8－9

- 双子素数を求めてみましょう.
  - 3と5は両方素数で，差が2なので，双子素数です.
  - 5と7も両方素数で，差が2なので，双子素数です.
  - 7と11は両方素数ですが，差が4なので，双子ではありません.
- 小さい方から20組の双子素数を出力するプログラムを作成しなさい.

twin.hs

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

primes (x:xs) = ... エラトステネスのふるい
```

- 素数のリストはエラトステネスのふるいを用いて生成したものを用いましょう.

```
% stack runghc twin.hs
[(3,5), (5,7), (11,13), (17,19),..., (311,313)]
```