アルゴリズムとデータ構造入門 2007年11月6日

アルゴリズムとデータ構造入門

1.手続きによる抽象の構築

1.3 高階手続きによる抽象化

奥 乃 博

大学院情報学研究科知能情報学専攻

知能メディア講座 音声メディア分野

工学部情報学科計算機科学コース

http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/07/IntroAlgDs/

okuno@nue.org

TAの居室は10号館4階奥乃1研，2研

TAのページがオープン，質問箱もあります

1

11月6日・本日のメニュー

- Fermat's test の補足
- 中間試験の説明

高階手続きによる抽象化

- 1.3.2 Constructing Procedures Using \`Lambda'
- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values

2

Fermat's Last Theorem

$$x^n + y^n = z^n$$

n>2 で *x, y, z* を満たす非負整

I have discovered a truly remarkable proof which this margin is too small to contain.

Euler's Conjecture

$$a^4 + b^4 + c^4 \neq d^4$$

が成立するだろう。

$$95800^4 + 217519^4 + 414560^4 = 422481^4$$

1987年発見

4

11月6日・本日のメニュー

- Fermat's test の補足
- 中間試験の説明

高階手続きによる抽象化

- **1.3.2 Constructing Procedures Using `Lambda'**
- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values

19

lambda: 無名(匿名)手続き

```
(define (plus4 x) (+ x 4))
```

は次式と等価

```
(define plus4 (lambda (x) (+ x 4)))
```

ラムダ式の読み方

本体(body)

```
(lambda                        (x)      (+    x      4))
    |                           |        |    |      |
the procedure of an argument x that adds x and 4
```

仮引数(formal parameters)

ラムダ式の適用

```
((lambda (x y z) (+ x y (square z))) 1 2 3)
        x = 1, y = 2, z = 3 を代入(置換)
12
```

20

Lambda as anonymous procedure

```
(lambda (x) (+ x 4))          無名(匿名)手続き
((lambda (x) (+ x 4)) 5)     手続き適用
```

```
(define (pi-sum a b)
  (define (pi-term x)
    (/ 1.0 (* x (+ x 2))) )
  (define (pi-next x) (+ x 4) )
  (sum pi-term a pi-next b) )
```

局所的な無駄な名前 Pi-term, Pi-next をなくす

```
(define (pi-sum a b)
  (sum (lambda (x) (/ 1.0 (* x (+ x 2))
       a
       (lambda (x) (+ x 4))
       b ))
```

21

lambda: Anonymous procedure

```
(define (fact n)
  (if (= n 0)
      1
      (* n (fact (- n 1))) ))
```

は次の式と等価

```
(define fact
  (lambda (n)
    (if (= n 0)
        1
        (* n (fact (- n 1))) )))
```

Using let to create local variables

$$f(x,y) = x(1+xy)^2 + y(1-y) + (1+xy)(1-y)$$

補助変数 a, b を使いたい

```
(define (f x y)
  (define (f-helper a b)
    (+ (* x (square a))
       (* y b)
       (* a b) ))
  (f-helper
    (+ 1 (* x y))
    (- 1 y) ))
```

$a = 1 + xy$

$b = 1 - y$

$f(x,y) = xa^2 + yb + ab$

1.3.2 Local Variables with let

```
(define (f x y)
  (define (f-helper a b)
    (+ (* x (square a))
       (* y b)
       (* a b) ))
  (f-helper
    (+ 1 (* x y))
    (- 1 y) ))
```

```
(define (f x y)
  (let ((a (+ 1 (* x y)))
        (b (- 1 y)) )
    (+ (* x (square a))
       (* y b)
       (* a b) )))
```

```
(define (f x y)
  ((lambda (a b)
     (+ (* x (square a))
        (* y b)
        (* a b) ))
   (+ 1 (* x y))
   (- 1 y) ))
```

(let ((<v_1> <e_1>)
      (<v_2> <e_2>)
      …
      (<v_n> <e_n>))
  <*body*>)

シンタックス・シュガー

scope of variables(有効範囲)

```
(let ((x 7))
  (+ (let ((x 3))
       (+ x (* x 10)) )
     x) )
```

Substition model

33

40

```
((lambda (x)
  (+
     x ))
 7 )
```

λ式に展開して考える

scope of variables(有効範囲)

```
(let ((x 5))
  (let ((x 3)
          (y (+ x 2)) )
    (* x y) ))
```

Substition model

x=3, y=7

21

```
((lambda (x)

 5 )
```

λ式に展開して考える

let* は、順番に変数を評価

```
(let ((x 5))
  (let* ((x 3)
          (y (+ x 2)) )
    (* x y) ))
((lambda (x)
   ((lambda (x)
      ((lambda (y)
         (* x y) )
       (+ x 2) )
    3 )
 5 )
```

Substition model

x=3, y=5

15

λ式に展開して考える

11月6日・本日のメニュー

- Fermat's test　の補足
- 中間試験の説明

高階手続きによる抽象化

- 1.3.2 Constructing Procedures Using `Lambda'
- **1.3.3 Procedures as General Methods**
- 1.3.4 Procedures as Returned Values

1.3.3 Procedures as General Methods

Finding roots of equations by the half-interval method（区間二分法）

```
(define (search f neg-point pos-point)
  (let ((midpoint (average neg-point pos-point)))
    (if (close-enough? neg-point pos-point)
        midpoint
        (let ((test-value (f midpoint)))
          (cond ((positive? test-value)
                 (search f neg-point midpoint))
                ((negative? test-value)
                 (search f midpoint pos-point))
                (else midpoint))))))
```


Finding roots of equations by the half-interval method

```
(define (close-enough? x y)
  (< (abs (- x y)) 0.001))

(define (half-interval-method f a b)
  (let ((a-value (f a))
        (b-value (f b)))
    (cond ((and (negative? a-value) (positive? b-value))
           (search f a b))
          ((and (negative? b-value) (positive? a-value))
           (search f b a))
          (else
           (error "Values are not of opposite sign" a b))
  )))
```

2点の値の符号が異なるかのチェックを行う

L:開始時の区間長、T:誤差許容度、
ステップ数: $\Theta(\log(L/T))$

Finding fixed points of functions（不動点）

抽象化すると

```
(define tolerance 0.00001)
(define (fixed-point f first-guess)
  (define (close-enough? v1 v2)
    (< (abs (- v1 v2)) tolerance))
  (define (try guess)
    (let ((next (f guess)))
      (if (close-enough? guess next)
          next
          (try next))))
  (try first-guess))
```

xが不動点 $x = f(x)$ $f(x), f(f(x)), f(f(f(x))), \cdots$

Finding fixed points of functions（不動点）

```
(fixed-point cos 1.0)
(fixed-point
  (lambda (y) (+ (sin y) (cos y)))
  1.0 )
```

$y = \cos y$

$y = \sin y + \cos y$

$y * y = x$ より $y = \frac{x}{y}$ と書くと、

次の関数の不動点探索となる $y \mapsto \frac{x}{y}$

```
(define (sqrt x)
  (fixed-point (lambda (y) (/ x y))
               1.0))
```

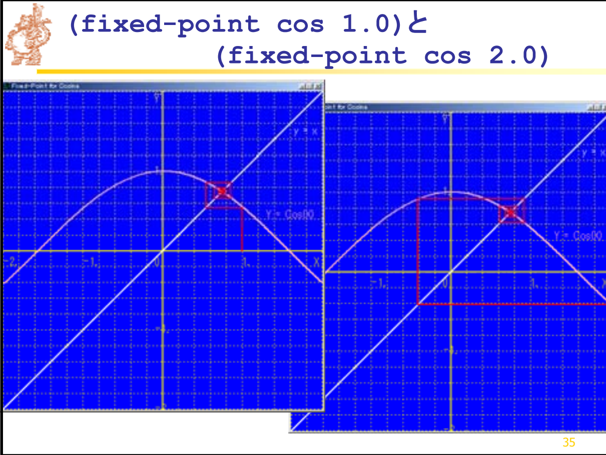
32

Finding fixed points of functions（不動点）

```
(fixed-point cos 0.2)
```

```
(fixed-point
  (lambda (y)
    (+ (sin y) (cos y)))
  0.1 )
```

(fixed-point cos 1.0)と (fixed-point cos 2.0)



35

不動点が求まらない場合がある $\sqrt{x}$

```
(define (sqrt x)
  (fixed-point (lambda (y) (/ x y))
               1.0))
```

$$y \mapsto \frac{x}{y}$$

(sqrt 2)を実行すると

1 → 2 → 1

(y_1 → y_2 → y_1)

Naïve Fixed Point (sqrt 2)

$$y \mapsto \frac{x}{y}$$

急いては事を仕損じる

アイデア倒れ

37

Average damping（平均緩和法）

One way to control such ocillations:

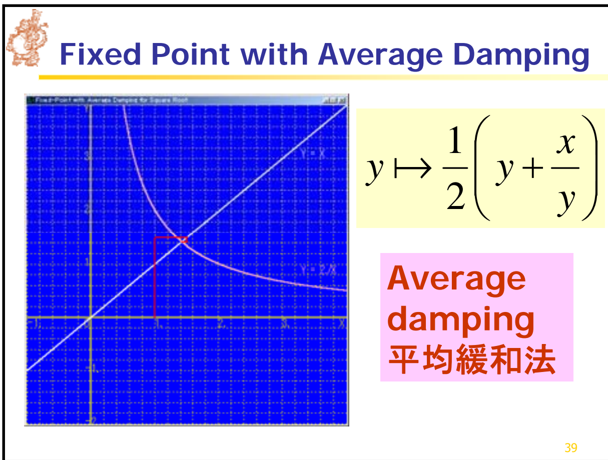
Redefine a new function

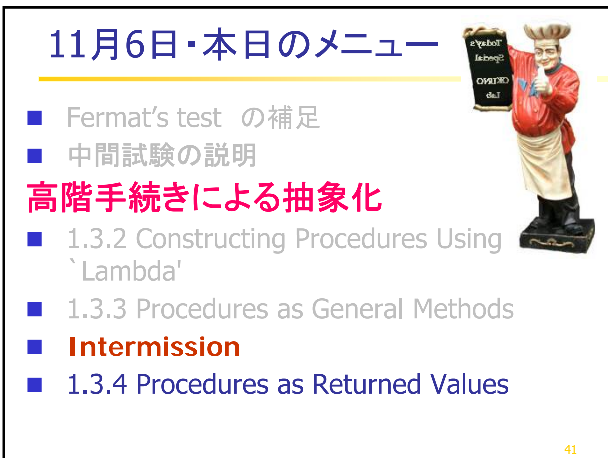
$$y \mapsto \frac{1}{2}\left(y + \frac{x}{y}\right)$$

```
(define (sqrt x)
  (fixed-point
      (lambda (y) (average y (/ x y)))
      1.0) )
```

Average damping（平均緩和法）

38





What is this instrument?

- タイガー計算機

http://www.tiger-inc.co.jp/temawashi/temawashi.html

42

タイガー計算機の操作

- 回転（＋、－）
- 左シフト（10をかける）
- 右シフト（10で割る）

2進数の場合

- 加算はフルアダー
- 減算は負の数に変化後加算
- 左シフト（2をかける）
- 右シフト（2で割る）

44

プログラミングの進め方

1．プログラムファイルを作成
   - emacs，meadow，mule を使う
   - メモ帳を使う（改行コードが悪さをする）
2．tus2，tustk2，Edwin(MIT)でプログラムファイルを読み込む．

   (load "c:/my-tus/file.lsp")
3．プログラムを動かす

   (foo 1 2 3)

45

emacs/meadow/mule

1. すべての入力は評価される
   - 単純な文字 ⇒ 自分が返される(self-evaluating)
2. コマンドは連想型
   - f(orward), b(ackward), e(nd), a(初め)
   - p(revious), n(ext)
   - d(delete), k(ill)
   - control-key(C-): 文字単位のコマンド
   - meta-key(M-) : 単語単位のコマンド(モードに依存)
   - control-meta-key(C-M-):S式単位のコマンド
3. Incremental search(C-s):逐次探索
4. C-x は拡張コマンド C-xC-s (save), C-xC-f (find)
5. ファイルの属性(ext)によりモード自動設定
6. タブで自動字下げ, 閉じ括弧で対応する開き括弧が点滅
7. M-x apropos で関連情報を検索するのがよい.

46

11月6日・本日のメニュー

- 中間試験の説明

高階手続きによる抽象化

- 1.3.2 Constructing Procedures Using `Lambda'
- 1.3.3 Procedures as General Methods
- Intermission
- 1.3.4 Procedures as Returned Values

47

1.3.4 Procedures as Returned Values

```
(define (sqrt x)
  (fixed-point (lambda (y) (average y (/ x y)))
               1.0))
```

平均緩和法を不動点の観点から眺めると

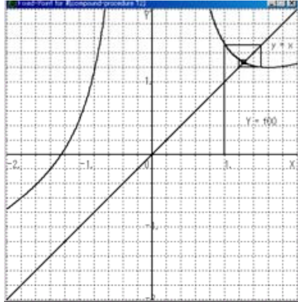
```
(define (average-damp f)
  (lambda (x) (average x (f x))))

((average-damp square) 10)

(define (sqrt x)
  (fixed-point
    (average-damp (lambda (y) (/ x y)))
    1.0))

(define (cube-root x)
  (fixed-point
    (average-damp (lambda (y) (/ x (square y))))
    1.0))
```

average-damp で統一的に捉えることが可能

48

Cubic-root の実行過程

```
(define (cube-root x)
  (fixed-point
    (average-damp (lambda (y) (/ x (square y))))
    1.0 ))
```



49

Newton's method & differentiation

```
(define (deriv g)
  (lambda (x)(/ (- (g (+ x dx)) (g x)) dx)) )
(define dx 0.00001)

(define (cube x) (* x x x))
((deriv cube) 5)

(define (newton-transform g)
  (lambda (x)(- x (/ (g x) ((deriv g) x)))) )

(define (newtons-method g guess)
  (fixed-point (newton-transform g) guess) )

(define (sqrt x)
  (newtons-method (lambda (y) (- (square y) x))
                  1.0))
```

$$y = x - \frac{g(x)}{g'(x)}$$

ニュートン法

50

更なる抽象化・first-class procedures

```
(define (fixed-point-of-transform g transform
   guess)
  (fixed-point (transform g) guess) )
```

1st method

```
(define (sqrt x)
  (fixed-point-of-transform
     (lambda (y) (/ x y))
     average-damp
     1.0 ))
```

2nd method

```
(define (sqrt x)
  (fixed-point-of-transform
     (lambda (y) (- (square y) x))
     newton-transform
     1.0 ))
```

手続きの構築で何ら差別がない

51

First-class citizen （第1級市民）

第1級市民の “権利と特権”

- 変数で名前をつけることができる.
- 手続きへ引数として渡すことができる.
- 手続きの結果として返すことができる.
- データ構造の中に含めることができる.

Microsoft Longhorn will make RAW ‘first class citizen.’
The Inquirer, Wed. Jun-8, 2005

52

手続き(関数)への演算： 導関数

- `(define dx 0.0001)`
- `(define (ddx f x) (/ (- (f (+ x dx)) (f x)) dx) )`
- `(ddx square 3)` ⇒ 6.00010000001205
- 我々はもっとスマートだった！導関数という考え方を採用
- `(define (deriv f) (lambda (x) (/ (- (f (+ x dx)) (f x)) dx) ))`
- `((deriv square) 3)` ⇒ 6.00010000001205
- `((deriv (deriv square)) 3)` ⇒ 1.99999998
- `(define (new-ddx f x) ((deriv f) x) )`

53

手続き(関数)の合成： 高階導関数

- この考え方を発展させ、高階導関数が構築できる
- `(define (compose f g) (lambda (x) (f (g x)) ))`
- `(define 2nd-deriv (compose deriv deriv))`
- `((2nd-deriv square) 3)` ⇒ 1.9999999878

もちろん手続きの合成も

- `((compose square sqrt) 7)` ⇒ 7.0
- `((2nd-deriv cos) pi)` ⇒ 0.999999993922529
- `(define 3rd-deriv (compose deriv 2nd-deriv))`
- `((3rd-deriv sin) pi)` ⇒ 0.999999960615838
- `((4th-deriv cos) pi)` ⇒ 1.11022302462516

54

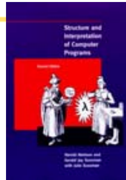
Let's Play JMC with your num.

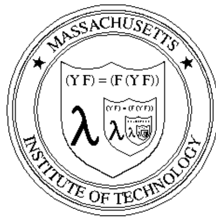
```
(define (jmc n)
  (if (> n 100)
      (- n 10)
     (jmc (jmc (+ n 11)))
  ))
```

- 各自、次の式を求めよ

(jmc (modulo 学籍番号 100))

補足: Fixed Point






```
(define (jmc n)
   (if (> n 100)
       (- n 10)
       (jmc (jmc (+ n 11))) ))

(fixed-point jmc 1) ⇒ ?
(Y F) = (F (Y F))   Y operator
                    (不動点となる手続きを作成)
(Y jmc) = (F (Y jmc))
        = (lambda (n)
            (if (> n 100) (- n 10) ?) )
```

Fixed Point Operator F

```
(define (Y F)
  (lambda (s)
    (F (lambda (x) (lambda (x) ((s s) x)))
       (lambda (s) (F (lambda (x) ((s s) x))))
    )))
```

再帰呼び出しに無名手続きを使いたい
(Y F) = (F (Y F))

詳しくは、Church numeralの項で説明。

宿題:11月13日正午締切

- 不動点の考え方を習得すること
- 宿題は、次の2題:
- Ex.1.35, 1.36.

61