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
1.2 手続きとその生成するプロセス
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
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/10/IntroAlgDs/
okuno@i.kyoto-u.ac.jp, okuno@nue.org
阿曽 慎平(M1) 奥乃研・音楽G (10号館4階奥乃1研)
平澤 恭治(M1) 奥乃研・ロボット聴覚G (10号館4階奥乃1研)

11月2日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- 1.3 Formulating Abstractions with Higher-Order Procedures
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using 'Lambda'
- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values

12月21日 中間試験を実施

Ex.1.5

```
(define (p) (p))
(define (test x y)
   (if (= x 0)
       0
       y ))
```

```
(test 0 (p))
```

一般にapplicative order で値が求まれば, normal order でも同じ値が求まる.

Applicative order	
1	(if (= x 0) 0 (p))
2	(if (= x 0) 0 (p))
3	(if (= x 0) 0 (p))
∞	...

Normal order	
1	(if (= x 0) 0 (p))
2	0

normal order は, 無駄な実行をすることがある. (e.g., my-square)

normal order をCall by name (名前呼び出し)

applicative order をCall by value (値呼び出し)

Ex.1.6 のポイント: Special Form

```
(define (p) (p))
(define (test x y)
   (if (= x 0)
       0
       y )))
```

通常のSchemeではどの様な値が返りますか？

```
1.(if (= 0 0)
      0
      (p) )

2.(new-if (= 0 0)
      0
      (p) )
```

```
(define (new-if pred then-c else-c)
   (cond (pred then-c)
         (else else-c) )))
```


小学校の復習

1. 143と546の最大公約数を求めよ.
2. 次のような**最小正整数** x を求めよ.
   - 3で割った余りが1 ($x \bmod 3 \equiv 1$),
   - 5で割った余りが2 ($x \bmod 5 \equiv 2$),
   - 7で割った余りが3 ($x \bmod 7 \equiv 3$).
3. 次のような**最小正整数** x を求めよ.
   - 3で割った余りが2 ($x \bmod 3 \equiv 2$),
   - 5で割った余りが3 ($x \bmod 5 \equiv 3$),
   - 7で割った余りが1 ($x \bmod 7 \equiv 1$).

Greatest Common Divisors (最大公約数)

ユークリッドの互除法

- $a \bmod b = r$ (modulo 剰余)とすると
- $GCD(a, b) = GCD(b, r)$ が成立。

```
(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (remainder a b)) ))
```

Modularity Calculus(合同式)

- *a* ≡ *b* mod *n*（congruent modulo *n*)
  「*a* mod *n* と *b* mod *n* が等しい」
- (reminder of) *x* modulo *n* は剰余
- *a+b* mod *n*
  ≡ *(a* mod *n* + *b* mod *n)* mod *n*
- *a*b* mod *n*
  ≡ *(a* mod *n* * *b* mod *n)* mod *n*
- *a* mod *(m*n)* は中国人剰余定理で求める
- $a^{p-1} \equiv 1 \mod p$ if *p* が素数(prime)

Chinese Remainder Theorem

連立1次合同式

$x \equiv b_1 \pmod{d_1}$
$x \equiv b_2 \pmod{d_2}$
...
$x \equiv b_t \pmod{d_t}$

の場合，$d_1, d_2, \ldots d_t$ が互いに素であれば，

$n = d_1 d_2 \ldots d_t$

を法として、ただ1つの解がある.

まず，$n/d_i = n_i$ とおけば，d_i と n_i は互いに素であるから，

$n_i x_i \equiv 1 \pmod{d_i}$

の解 x_i を求めることができる. ここで，

$x \equiv b_1 n_1 x_1 + b_2 n_2 x_2 + \ldots + b_t n_t x_t \pmod{n}$

とすれば，この x は明らかにもとの合同式をすべて満足する.

Chinese Remainder Theoremの応用

x mod *105* は？

- 105 = 3 * 5 * 7 より
  - *x* ≡ 1 (mod 3)
  - *x* ≡ 2 (mod 5)
  - *x* ≡ 3 (mod 7)
- 各素数に対する逆数を求める
  - 35*2 ≡ 1 (mod 3)
  - 21*1 ≡ 1 (mod 5)
  - 15*1 ≡ 1 (mod 7) より、
- Chinese Remainder Theorem より

  x mod 105 ≡ 1*35*2 +2*21*1+3*15*1 mod 105
  = 157 mod 105 ≡ 52 mod 105

Chinese Remainder Theoremの応用

x mod *105* は？

- 105 = 3 * 5 * 7 より
  - *x* ≡ 2 (mod 3)
  - *x* ≡ 3 (mod 5)
  - *x* ≡ 1 (mod 7)
- 各素数に対する逆数を求める
  - 35*2 ≡ 1 (mod 3)
  - 21*1 ≡ 1 (mod 5)
  - 15*1 ≡ 1 (mod 7) より、
- Chinese Remainder Theorem より

  x mod 105 ≡ 2*35*2 +3*21*1+1*15*1 mod 105
  = 218 mod 105 ≡ 8 mod 105

Chinese Remainder Theoremの応用 (2)

2^{90} mod *91* は？

- 91 = 7 * 13
- $2^3 \equiv 1 \pmod 7$ より，$2^{90} \equiv 1 \pmod 7$
- $2^6 \equiv -1 \pmod{13}$ より，
- $2^{84} \equiv 1 \pmod{13} \Rightarrow 2^{90} \equiv -1 \pmod{13}$
- 13*6 ≡ 1 (mod 7)
- 7*2 ≡ 1 (mod 13) より，
- 2^{90} mod *91* ≡ 1*13*6 -1*7*2 = 64

Lameの定理

- GCD(*a*, *b*)（ただし、*b* < *a*)の計算に*k* step 必要なら、*b* ≥ fib(*k*)
- 例えば、GCD(*m*, *n*)（ただし、*n* < *m* ）が *k* step かかるとすると、*n* ≥ fib(*k*) ≒ $\Phi^k / \sqrt{5}$

$$\phi = \frac{1}{2}\left(1+\sqrt{5}\right) \qquad fib(n) \cong \frac{\phi^n}{\sqrt{5}}$$

- つまり、ステップ数は、*n* の対数的に増加。
- $\Theta(\log n)$ steps

11月2日・本日のメニュー

- 1.2.5 Great Common Divisors
- **1.2.6 Examples: Testing for Primality**
- 1.3 Formulating Abstractions with Higher-Order Procedures
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using 'Lambda'
- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values

Testing for Primality （素数の定義は？）

```
(define (smallest-divisor n)
   (find-divisor n 2))

(define (find-devisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
        ((divides? test-divisor n) test-divisor)
        (else (find-divisor n
                  (+ test-divisor 1) )) ))

(define (divides? a b)
   (= (remainder b a) 0) )

(define (prime? n)
   (= n (smallest-divisor n)) )
```


Improvement by HGO

```
(define (smallest-divisor n)
  (find-divisor n 3))

(define (find-devisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
        ((divides? test-divisor n) test-divisor)
        (else (find-divisor n
                  (+ test-divisor 2) )) ))

(define (divides? a b)
  (= (remainder b a) 0) )

(define (prime? n)
  (if (even? n)
      2
      (= n (smallest-divisor n)) ))
```


The Fermat Test

$a^{p-1} \equiv 1 \bmod p$ if p が素数(prime)

```
(define (expmod base exp m)
  (cond ((= exp 0) 1)
        ((even? exp)
         (remainder (square (expmod base (/ exp 2) m)) m) )
        (else
         (remainder (* base (expmod base (- exp 1) m)) m) )
  ))

(define (fermat-test n)
  (define (try-it a)
    (= (expmod a n n) a) )
  (try-it (+ 1 (random (- n 1)))))

(define (fast-prime? n times)
  (cond ((= times 0) true)
        ((fermat-test n) (fast-prime? n (- times 1)))
        (else false) ))
```


Probabilistic Algorithms(確率的アルゴリズム)

- Carmichael numbers: 561, 1105, 1072, 2465,
  $a^{560}=a^2\ a^{10}\ a^{16}$
  $a^2 \equiv 1 \bmod 3$, $a^{10} \equiv 1 \bmod 11$, $a^{16} \equiv 1 \bmod 17$
  $\Rightarrow a^{560} \equiv 1 \bmod 561 = 3 * 11 * 17$
- Fermat's testは，エラーの機会を任意に小さくできる。
  →*probabilistic algorithm*
  必要条件のみ満足
- *Algorithm*: Wilson's test
  p is a prime precisely when $(p-1)! \equiv -1 \bmod p$
  必要十分条件
  $n! \sim (2\pi n)^{\frac{1}{2}} (n/e)^n$

Fermat's test は十分条件ではない

11月2日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- **1.3 Formulating Abstractions with Higher-Order Procedures**
- **1.3.1 Procedures as Arguments**
- 1.3.2 Constructing Procedures Using 'Lambda'
- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values

1.3.1 Procedures as Arguments

```
(define (sum-integers a b)
  (if (> a b)
      0
      (+ a (sum-integers (+ a 1) b)) ))

(define (sum-cubes a b)
  (if (> a b)
      0
      (+ (cube a) (sum-cubes (+ a 1) b)) ))

(define (cube x) (* x x x))

(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1.0 (* a (+ a 2))) (pi-sum (+ a 4) b)) ))
```

$$\sum_{i=a}^{b} i \qquad \sum_{i=a}^{b} i^3 \qquad \sum_{i=a,i+4}^{b} \frac{1}{i(i+2)}$$

```
(define (<name> a b)
  (if (> a b)
      0
      (+ (<term> a)
         (<name> (<next> a) b)) ))
```

手続きの共通パターンを抽象化
⇒ 高階手続き

汎用総和手続きを共通パターンを基に定義する

```
(define (<name> a b)
  (if (> a b)
      0
      (+ (<term> a)
         (<name> (<next> a) b)) ))

(define (sum f a next b)
  (if (> a b)
      0
      (+ (f a)
         (sum f (next a) next b)) ))

(define (inc n) (+ n 1))
(define (cube x) (* x x x))
(define (sum-cubes a b)
  (sum cube a inc b) )
(define (identity x) x)
(define (sum-integers a b)
  (sum identity a inc b) )
```

$$\sum_{i=a,next(i)}^{b} f(i) \qquad \sum_{i=a,i+1}^{b} cube(i) \qquad \sum_{i=a,i+1}^{b} i$$

Ex. 1.32 総和 (sum), 積 (product)を抽象化

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a) next b)) )

(define (product term a next b)
  (if (> a b)
      1
      (* (term a)
         (product term (next a) next b)) )
```

$$\sum_{i=a,next(i)}^{b} f(i) \qquad \prod_{i=a,next(i)}^{b} f(i)$$

```
(define (<combiner> <name> <term> a <next> b)
  (if (> a b)
      <null-value>
      (<combiner> (<term> a)
         (<name> <term> (<next> a) <next> b))
 ))
```


accumulate ⇦ sum, product

```
(define (accumulate combiner null-value
                    term a next b )
  (if (> a b)
      null-value
      (combiner (term a)
          (accumulate combiner null-value
                      term (next a) next b ))))

(define (sum term a next b)
   (accumulate + 0 term a next b))

(define (product term a next b)
   (accumulate * 1 term a next b))
```

$$\sum_{i=a,next(i)}^{b} f(i) \qquad \prod_{i=a,next(i)}^{b} f(i)$$

11月2日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- 1.3 Formulating Abstractions with Higher-Order Procedures
- 1.3.1 Procedures as Arguments
- **1.3.2 Constructing Procedures Using 'Lambda'**
- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values

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

```
(lambda      (x)     (+   x     4))
```

the procedure of an argument x that adds x and 4

仮引数 (formal parameters)　　本体 (body)

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)

```
(+ 1 2 (square 3))
(+ 1 2 9)
12
```

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 ))
```


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

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

$$a = 1+xy$$
$$b = 1-y$$
$$f(x,y) = xa^2 + yb + ab$$

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) ))
```


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 ((<v1> <e1>)
      (<v2> <e2>)
      ...
      (<vn> <en>) )
  <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) )
((lambda (x)
  (+ ((lambda (x)
       (+ x (* x 10)) )
     3 )
     x ))
 7 )
```

Substition model

33

40

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

Substition model

x=3, y=7

21

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2日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- 1.3 Formulating Abstractions with Higher-Order Procedures
  - 1.3.1 Procedures as Arguments
  - 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: Θ(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，

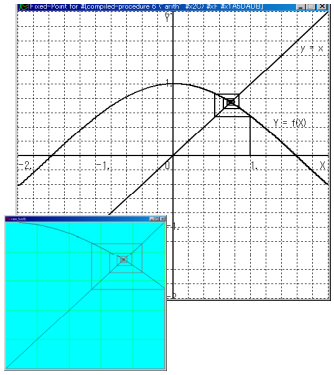
次の関数の不動点探索となる

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

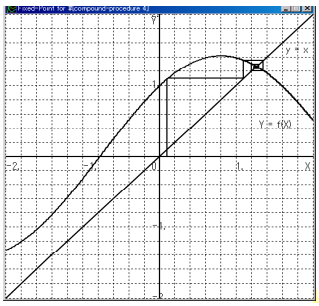
```
(define (sqrt x)
  (fixed-point (lambda (y) (/ x y))
               1.0))
```

Finding fixed points of functions(不動点)

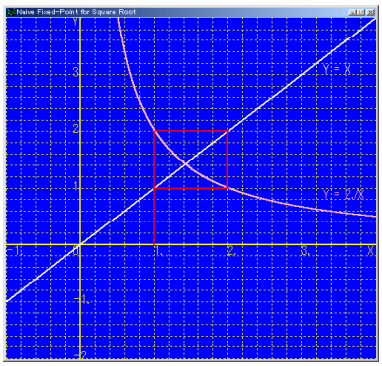
```
(fixed-point cos 1.0)
```



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



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

Average damping（平均緩和法）

One way to control such ocillations:
Redefine a new function

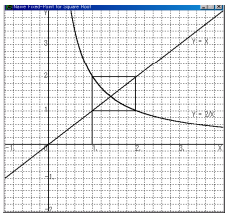
$$y \mapsto \frac{1}{2}\left(y + \frac{x}{y}\right)$$

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

Average damping（平均緩和法）

宿題:11月2日正午締切

1. 教科書練習問題 Ex1.29, 1.31
2. Program ファイルとレポート(pdf) を
   SICP-5@zeus.kuis.kyoto-u.ac.jp に送付
3. 【随意】不動点を求める可視化せよ.



- 友達に教えてもらったら, その人の名前を
  明記すること. Webは出展を明記. (otherwise『同じ』回答は減点)

Rules of Thumb

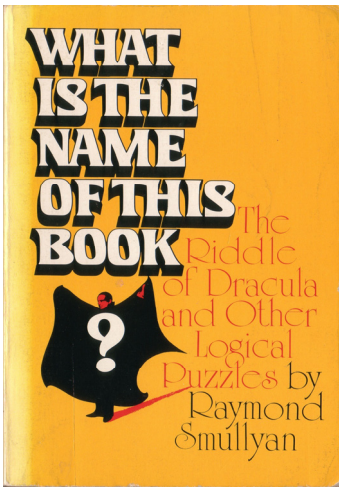
1. Rule of 72 : $r \times y = 72$
   For y years with an interest rate of r % per year ⇒ roughly double.
2. π *seconds is a nanocentury.*
3. 1 *year* $= 3.155 \times 10^7$ *seconds*

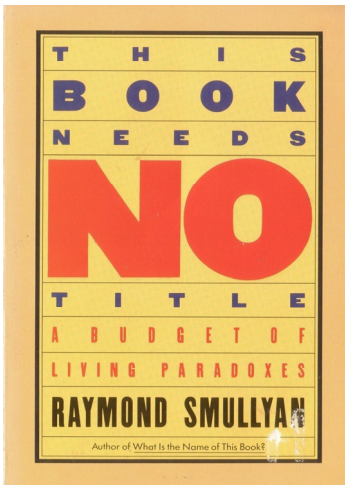
emacs/meadow/mule

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.

Interesting books

1. Douglas R. Hofstadter
2. Reymond Sumullyan
3. Doug Adams
4. Don E. Knuth
5. John H. Conway and Richard K. Guy





自己参照(Self-Reference)の例

1. What is the book of this book?
2. This book needs no title.
3. この文章は赤い色で書かれている.
4. この文章は17文字でできています.
5. 嘘つき村の住民は嘘つきです. 嘘つき村の住民が「私は嘘はつきません」と言った.
6. 「クレタ人は嘘つきである」とクレタ人は言った.(新約聖書「テトスへの手紙」)

自己参照(Self-Reference)

嘘つき村の住民は皆嘘つきです. 他の村の住民は嘘はつきません. 嘘つき村の住民が「私は嘘はつきません」と言った.

- 発話が嘘(住民) ⇒
- 発話が本当 ⇒

ラッセルのパラドックス

- Aのすべての部分集合: 2^A

例: $A=\{a, b, c\}$

$2^A=\{\{\}, \{a\},\{b\},\{c\},\{a,b\},\{b,c\},\{c,a\},\{a,b,c\}\}$

- すべての部分集合を含む集合Sを考える
- いずれが成立するか
  1. $S \in S$
  2. $S \notin S$