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

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

TAの居室は文学部東館4階奥乃1研，2研

平山 直樹 if mod(学籍番号の下3桁, 3) ≡ 0

黄 楊暘 if mod(学籍番号の下3桁, 3) ≡ 1

山口 雄紀 if mod(学籍番号の下3桁, 3) ≡ 2

10月23日・本日のメニュー

- 数学的準備
- 1-1-6 Conditional Expressions and Predicates
- 1-1-7 Example: Square Roots by Newton's Method
- 1-1-8 Procedures as Black-Box Abstractions
- 1.2.1 Linear Recursion and Iteration（復習）
- 1.2.2 Tree Recursion（復習）
- 1.2.3 Growth of Order
- 1.2.4 Exponentiation

スターリングの公式（Sterling's formula）

- n!は $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\lambda_n}$ ただし $\frac{1}{12n+1} < \lambda_n < \frac{1}{12n}$
- スターリング級数は

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1+\frac{1}{12n}+\frac{1}{288n^2}-\frac{1}{51840n^3}-\frac{1}{2488320n^4}+\cdots\right)$$

- 対数をとると

$$\ln n! = n\ln n - n + \frac{1}{2}\ln(2\pi n)+\frac{1}{12n}-\frac{1}{360n^3}+\frac{1}{1260n^5}-\frac{1}{1680n^7}+\cdots$$

for n>>0

$$\ln n! \approx n\ln n - n$$

ビネの公式:フィボナッチ数の一般項

- Fibonacci数の漸化式

$F_0 = 0 \quad F_1 = 1 \quad F_{n+2} = F_n + F_{n+1}$

- ビネの公式

$$F_n = \frac{1}{\sqrt{5}}\left\{\left(\boxed{\frac{1+\sqrt{5}}{2}}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right\}$$

黄金比(Golden Ratio)

- 帰納法で証明せよ.

1. *n* = 0, 1 を検証

2. *k* のとき成立を仮定し, *k+1* で成立を示す

NoStudent Left Behind

ビネの公式を導出する

NoStudent Left Behind

Fibonacci数の漸化式

$F_0 = 0 \quad F_1 = 1 \quad F_{n+2} = F_n + F_{n+1}$

1. 線形差分方程式の解法

1) 特性方程式を作成・解を求める.

2) その根をα, βとすると元の差分方程式の解は

$F_n = C\alpha^n + D\beta^n$と表せる。

ただし, C, D は定数

3) 初期値等から、定数項の値を求める

2. 母関数(generating function)の解法

Fibの呼ばれる回数 *C(n)*

NoStudent Left Behind

- *C*(n):n に対して fib の呼ばれる回数
- *C*(0)=*C*(1)=1
- n≧2 に対して *C*(n)=*C*(n-1)+*C*(n-2)+1
- *C*(2)=3, *C*(3)=5, *C*(4)=9, *C*(5)=15, …
- 一般に、*C*(k) > $2^{k/2}$

C(n) を求める

1. F(n)=*C*(n)+1 とおくと
2. F(n)=F(n-1)+F(n-2) for n≧2
3. F(0)=2, F(1)=2

Fibの呼ばれる回数 $C(n)=F(n)-1$

- n≧2 に対して $C(n)=C(n-1)+C(n-2)+1$
- $F(n)=C(n)+1$ とおくと
- $F(n)=F(n-1)+F(n-2)$　for n≧2
- $F(0)=2$, $F(1)=2$

さあ、解いてみてください。

1.1.7 Square Root by Newton's Method

$\sqrt{x}$ *is the* y *such that* $y^2 = x$ *and* $y \geq 0$

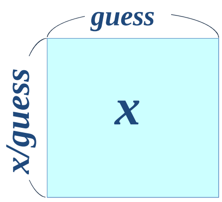
```
(define (sqrt-iter guess x)
  (if (good-enough? guess x)
      guess
     (sqrt-iter (improve guess x) x) ))
(define (improve guess x)
  (average guess (/ x guess)) )
(define (average x y)
  (/ (+ x y) 2) )
(define (good-enough? guess x)
   (< (abs (- (square guess) x)) 0.001)
(define (sqrt x) (sqrt-iter 1.0 x))
```

improveの設計

$\sqrt{x}$ *is the* y *such that* $y^2 = x$ *and* $y \geq 0$

```
(define (improve guess x)
  (average guess (/ x guess)) )
```

```
(sqrt 2.0)
(sqrt-iter 1.0 2.0)
(sqrt-iter 1.5 2.0)
(sqrt-iter 1.416667 2.0)
(sqrt-iter 1.414215 2.0)
(sqrt-iter 1.414213 2.0)
```



宮田君による

1.1.7 Square Root by Newton's Method

```
(define (sqrt x)
  (sqrt-iter 1.0 x) )
```

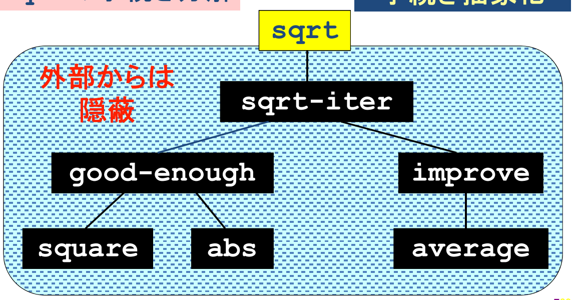
と定義すれば,

```
(sqrt 9)

(sqrt (+ 100 37))

(sqrt (+ (sqrt 2) (sqrt 3)))

(sqrt (sqrt 1000))
```

1.1.8 Procedures as Black-Box Abstraction (手続き：ブラックボックス抽象化)

Sqrtの手続き分解

手続き抽象化



手続き抽象化の効用 Square の定義

1．内部実装(implementation)の隠蔽

- `(define (square x) (* x x))`
- `(define (square x)`
  `(exp (double (log x))) )`
  `(define (double x) (+ x x))`

x^2

$e^{2\ln x}$

2．局所名(local names)の隠蔽

- `(define (square x) (* x x))`
- `(define (square y) (* y y))`

束縛変数 (bound variables)と 自由変数(free variables)

bound (束縛)

```
(define (sqrt-iter guess x)
  (if (good-enough? guess x)
      guess
      (sqrt-iter (improve guess x) x) ))
(define (good-enough? guess x)
   (< (abs (- (square guess) x)) 0.001)
(define (good-enough? v target)
   (< (abs (- (square v) target)) 0.001)
```

- 束縛変数:仮パラメータは手続きで束縛
- 自由変数:束縛・captureされていない
- 有効範囲(scope)変数の束縛されている式の範囲

Block Structure(ブロック構造): xのscope(有効範囲)は

```
(define (sqrt x)
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001) )
  (define (improve guess)
    (average guess (/ x guess)) )
  (define (sqrt-iter guess)
     (if (good-enough? guess)
         guess
         (sqrt-iter (improve guess)) ))
  (sqrt-iter 1.0) )
```

静的有効範囲(lexical scoping)

Tree Recursion: Tower of Hanoi

ハノイの塔を解こう.

1. 一度には1枚の円盤しか動かせない.
2. 小さい円盤の上には大きな円盤は置けない.

Java で実行

Tower of Hanoi

```
(define (move-tower size from to extra)
   (cond ((= size 0) #true)
         (else
            (move-tower (- size 1) from extra to)
            (print-move from to)
            (move-tower (- size 1) extra to from))
    ))
(define (print-move from to)
   (newline)
   (display "move top disk from ")
   (display from)
   (display " to ") (display to) )

(define (solve-tower-of-hanoi size from to)
    (move-tower size from to (- 6 from to)) )
```

Ex.1.10 アッカーマン(Ackermann)関数

$$Ack(m,n)=\begin{cases} n+1, & \text{if } m=0 \\ Ack(m-1,1), & \text{if } n=0 \\ Ack(m-1, Ack(m,n-1)), & \text{otherwise} \end{cases}$$

```
(define (ack m n)
    (cond ((= m 0) (+ n 1))
          ((= n 0) (ack (- m 1) 1))
          (else (ack (- m 1)
                     (ack m (- n 1)) ))))
(ack 0 2)
(ack 1 2)
(ack 2 2)
(ack 3 2)
```

Ackermann関数は線形再帰ではない！

宿題1：アッカーマン関数の値

練習問題

- (ack 0 2)
- (ack 1 2)
- (ack 2 2)
- (ack 3 2)

計算過程を書くこと. 以下随意課題

1. (ack 0 n) ≡ n+1 (理由も)
2. (ack 1 n) ≡ ？ (理由も)
3. (ack 2 n) ≡ ？ (理由も)
4. (ack 3 n) ≡ ？ (理由も)
5. (ack 4 n) ≡ ？ (理由も)

宿題：10月30日正午締切

1. Ackermann関数のファイルを作成せよ．　ack.scm
2. Ackermann関数を実行し，出力結果を求めよ．
   (ack 0 2), (ack 1 2), (ack 2 2), (ack 3 2)
3. 教科書練習問題 Ex1.5（**置換プロセスを明記のこと**）
4. Program ファイルとレポート（pdf）を
   SICP-4@zeus.kuis.kyoto-u.ac.jp に送付
5. 【随意】 次の一般式を求めよ．理由，計算過程明記のこと．
   (ack 0 n) ≡ n+1, (ack 1 n) ≡ ?, (ack 2 n) ≡ ?,
   (ack 3 n) ≡ ? , (ack 4 n) ≡ ?

- **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）

Ex. Counting Change

1. **1ドルの両替の方法は何通り？**
2. **50セント (half dollar), 25セント (quarter), 10セント (dime), 5セント(nickel), 1セント(penny)**

3. **一般化： 分割数を求める**

n種類の硬貨，金額aの両替の場合の数は

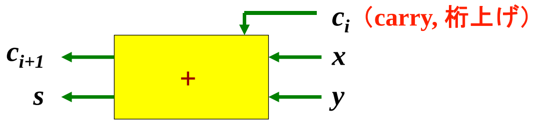
- 使える硬貨をある順番で並べておくと
- n種類の硬貨で金額aの両替の場合の数は
  1. 先頭の種類を除いたすべての硬貨を使って金額aを両替する場合の数
     ＋
  2. 先頭の種類の硬貨（額面d）とすると，a-dの額を全n種の硬貨を使って両替する場合の数
  3. 初期値：a=0の時1，a<0の時かn=0の時0

分割統治法 (divide-and-conquor)

Ex. Counting Change

```
(define (count-change amount)
  (cc amount 5) )

(define (cc amount kinds-of-coins)
  (cond ((= amount 0) 1)
        ((or (< amount 0) (= kinds-of-coins 0)) 0)
        (else
           (+ (cc amount (- kinds-of-coins 1))
              (cc (- amount (first-denomination
                                  kinds-of-coins))
                  kinds-of-coins )))))

(define (first-denomination kinds-of-coins)
  (cond ((= kinds-of-coins 1) 1)
        ((= kinds-of-coins 2) 5)
        ((= kinds-of-coins 3) 10)
        ((= kinds-of-coins 4) 25)
        ((= kinds-of-coins 5) 50) ))
```

Abacus & Binary Adder (2進加算器)




```
(define (adder x y c)
   (define (carry x y c)
      (if (or (and (= x 1) (= y 1))
              (and (= y 1) (= c 1))
              (and (= c 1) (= x 1)) )
          1 0 ))
   (define (sum x y c)
      (xor x y c) )
   (cons (sum x y c) (carry x y c)) )

(define (xor x y z)
  (if (= x 0)
      (if (= y 0) z (if (= z 0) 1 0))
      (if (= y 0) (if (= z 0) 1 0) z) ))
```

What is this instrument?

タイガー計算機

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