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10号館4階奥乃1研，2研
阿曽 慎平(M1) 奥乃研・音楽G
平澤 恭治(M1) 奥乃研・ロボット聴覚G

10月26日・本日のメニュー

- 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

Procedure（手続き）vs. Process(プロセス)

- 手続きが再帰的とは、構文上から定義。
  自分の中で自分を直接・間接に呼び出す。
- 再帰的手続きの実行
  - 再帰プロセスで実行
  - 反復プロセスで実行
- 線形再帰プロセスは線形反復プロセスに変換可能
  「tail recursion（末尾再帰的）」
- 再帰プロセスでは、deferred operation用にプロセスを保持する必要あり ⇒ スペース量が余分に必要
- Scheme のループ構造はsyntactic sugar
  - do, repeat, until, for, while

Applicative order vs. normal order (適用順序と正規順序)

- 「適用順序(作用順序, Applicative order)」
  今まで見てみた置換モデルの評価順序:
- 別の順序:「正規順序(normal-order)」:
  仮パラメータをすべて置換してから，簡約する.

```
1.  (f 5)
2.       ((sum-of-squares (+ a 1) (* a 2)) 5)
3.  (sum-of-squares (+ 5 1) (* 5 2))
4.       ((+ (square x) (square y)) (+ 5 1) (* 5 2))
5.  (+ (square (+ 5 1)) (square (* 5 2))
6.       (+ ((* x x) (+ 5 1)) ((* x x) (* 5 2))
7.  (+ (* (+ 5 1) (+ 5 1)) (* (* 5 2) (* 5 2)))
8.  (+ (* 6 6) (* 10 10))
9.  (+ 36 100)
10. 136
```

2回同じものを計算

1.1.6 Condtional Expressions and Predicates (条件式と述語)

- 条件式の一般形; cond は特殊形式 (special form)
- (cond (<p_1> <e_{11}> ...<e_{1m}>)
  (<P_2> <e_{21}>...<e_{2K}>)
  ...
  (<p_n> <e_{n1}>...<e_{np}>))
- 式の対 (<p> <e> ... <e>) : 節(clause)
- <p>: 述語(predicate)
- 述語の値: true (#t) か false (#f).
- <e>: 帰結式(consequent expression)
- 特別の<p>: else (#t を返す)
- 節の評価は，<p>が#tなら<e>が順に評価される.
- 一旦述語が#tを返すと，それ以降の節は評価されない.

1.1.6 Condtional Expressions and Predicates (条件式と述語)

- 絶対値をcase analysis (場合分け)で定義

$$|x| = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -x & \text{if } x < 0 \end{cases}$$

cond, if の実行は、置換モデルで、特別扱い
Special form(特殊形式)

```
1.(define (abs x)
    (cond ((> x 0) x)
          ((= x 0) 0)
          ((< x 0) (- x)) ))
```

if : Syntax sugar
糖衣

```
2.(define (abs x)
    (cond ((< x 0) (- x))
          ((>= x 0) x) ))
```

```
(or (> x 0) (= x 0))
```

```
3.(define (abs x)
    (cond ((< x 0) (- x))
          (else x) ))
```

```
4.(define (abs x)
    (if (< x 0)
        (- x)
        x ))
```

1.1.6 Predicates (述語)

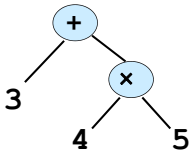
- (and <e1> ... <en>) 論理積 (左から評価)
- (or <e1> ... <en>) 論理和 (左から評価)
- (not <e>) 論理否定

例:

- 5 < x < 10 ⇒
- (define (>= x y)
  (or (> x y) (= x y)))
- (define (>= x y)
  (not (< x y)))

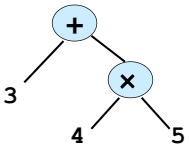
Ex.1.2 前置記法(prefix notation)

- 式 (演算子 被演算子 ...)
  operator operands
- 式の記法
  - 前置記法 (prefix notation , Pollish notation, ポーランド記法)
    + 3 * 4 5
  - 中置記法 (infix notation)
    3 + (4 * 5)
  - 後置記法 (postfix notation, reverse Pollish notation、逆ポーランド記法)
    3 4 5 * +
- 木表現はどれも同じ



木の辿り方から3つの記法への変換

- 木の辿り方
  - 前順走査 (pre-order traversal)
    ノード⇒左部分木⇒右部分木
    + ⇒ 3 ⇒ * ⇒ 4 ⇒ 5
  - 間順走査 (in-order tr.)
    左部分木⇒ノード⇒右部分木
    3 ⇒ + ⇒ 4 ⇒ * ⇒ 5
  - 後順走査 (post-order tr.)
    左部分木⇒右部分木⇒ノード
    3 ⇒ 4 ⇒ 5 ⇒ * ⇒ +



Javaプログラムのデモ

http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/10/IntroAlgDs/

10月26日・本日のメニュー

- 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
- 1.2.2 Tree Recursion
- 1.2.3 Growth of Order
- 1.2.4 Exponentiation

1.1.7 Square Root by Newton's Method

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

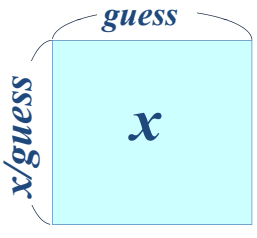
```
(define (sqrt-iter guess x)
  (if (good-enough? guess x)
      guess
      (sqrt-iter (improve guess x) x) ))
(define (improve guess x)
  (average guess (/ x guess)) )
(define (average x y)
  (/ (+ x y) 2) )
(define (good-enough? guess x)
   (< (abs (- (square guess) x)) 0.001)
(define (sqrt x) (sqrt-iter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



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))
```

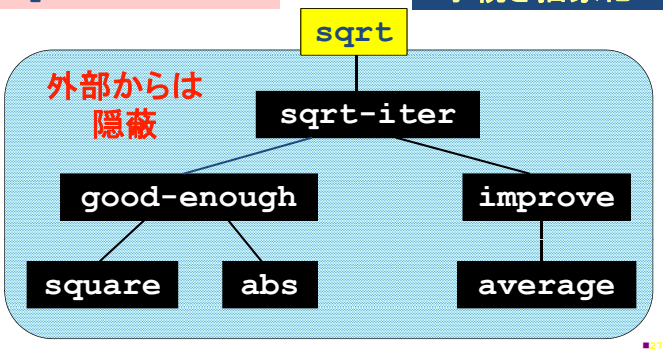

10月26日・本日のメニュー

- 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
- 1.2.2 Tree Recursion
- 1.2.3 Growth of Order
- 1.2.4 Exponentiation

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

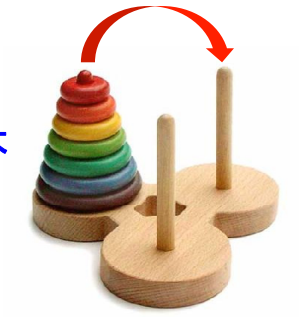
```
(define (sqrt x)
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001) )
  (define (improve guess)
    (average guess (/ x guess)) )
  (define (sqrt-iter guess)
     (if (good-enough? guess)
         guess
         (sqrt-iter (improve guess)) ))
  (sqrt-iter 1.0) )
```

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 size)) )
```


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:11月2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
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)

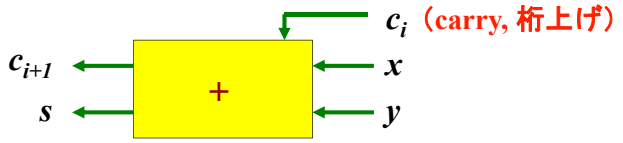
Ex. Counting Change

```
(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)

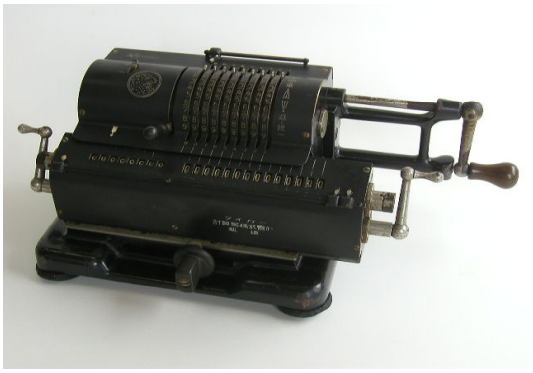



```
(define (adder x y c)
   (define (carry x y c)
      (if (or (and (= x 1) (= y 1))
              (and (= y 1) (= c 1))
              (and (= c 1) (= x 1)) )
          1 0 ))
   (define (sum x y c)
      (xor x y c) )
   (cons (sum x y c) (carry x y c)) )

(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

10月26日・本日のメニュー

- 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
- 1.2.2 Tree Recursion
- **1.2.3 Growth of Order**
- 1.2.4 Exponentiation

1.2.3 Order of Growth

R(n) は、ステップ数あるいはスペース量

For all $n > n_0$

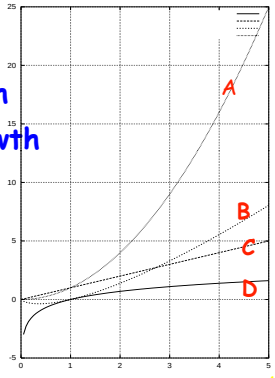
- R(n) が Θ(f(n)) 上下限 $\quad k_1 f(n) \geq R(n) \geq k_2 f(n)$
- R(n) が O(f(n)) 上限 $\quad R(n) \leq k f(n)$
- R(n) が Ω(f(n)) 下限 $\quad R(n) \geq k f(n)$

Order of Growth R(n) の例

- $\Theta(1)$: constant growth
- $\Theta(n)$: linear growth
- $\Theta(b^n)$: exponential growth
- $\Theta(\log n)$: logarithmic growth
- $\Theta(n^m)$: power law growth

下記はどの曲線？

1. $y = x$
2. $y = x^2$
3. $y = \log x$
4. $y = x \log x$



10月26日・本日のメニュー

- 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**
- **1.2.2 Tree Recursion**
- **1.2.3 Growth of Order**
- 1.2.4 Exponentiation

2種類の階乗(n!)

- トップダウン(top-down)式で計算–線形再帰

```
(define (factorial n)
  (if (<= n 0)
      1
      (* n (factorial (- n 1))) ))
```

- ボトムアップ(bottom-up)式で計算–線形反復

```
(define (fact n)
  (define (iter product counter)
     (if (> counter n)
         product
         (iter (* counter product)
               (+ counter 1) )))
  (iter 1 1) )
```


練習問題: 表を埋めてください

手続き	*ステップ数* 各手続きが呼ばれた回数	*スペース* 遅延演算の個数
再帰型 (factorial n)		
繰り返し型 (fact n)		
テーブル参照型fact		

テーブル参照型fact

n	0	1	2	3	4	5	6	7	8
n!	1	1	2	6	24	120	720	5040	40320

2種類のFibonacci 関数

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2)) ))))
```

- トップダウン(top-down)式に計算 – 木構造再帰

```
(define (fib-iter n)
   (define (iter a b count)
       (if (= count 0)
           b
           (iter (+ a b) a (- count 1)) ))
   (iter 1 0 n) )
```

- ボトムアップ(bottom-up)式に計算 –

  線形再帰だが、線形反復プロセス

練習問題: 表を埋めてください

手続き	*ステップ数* 各手続きが呼ばれた回数	*スペース* 遅延演算の個数
再帰型(fib n)		
繰り返し型 (fib-iter n)		
テーブル参照型 fib		

テーブル参照型fib

n	0	1	2	3	4	5	6	7	8	9	10
fib(n)	0	1	1	2	3	5	8	13	21	34	55

10月26日・本日のメニュー

- 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
- 1.2.2 Tree Recursion
- 1.2.3 Growth of Order
- **1.2.4 Exponentiation**

1.2.4 Exponentiation（冪乗）

```
(define (expt b n)
  (if (= n 0)
      1
      (* b (expt b (- n 1))) ))
```

$b^n = b * \cdots * b$

- Linear recursive process Θ(n) steps, Θ(n) space

```
(define (expt b n)
  (define (iter counter product)
    (if (= counter 0)
        product
        (iter (- counter 1) (* b product)) ))
  (iter n 1) )
```

p = b * p
c = c - 1

- Linear iterative process Θ(n) steps, Θ(1) space

Exponentiation

```
(define (fast-expt b n)
  (cond ((= n 0) 1)
        ((even? n)
         (square (fast-expt b (/ n 2))))
        (else (* b (fast-expt b (- n 1))) ))
(define (even? n)
  (= (remainder n 2) 0) )
```

- recursive process Θ(log n) steps, Θ(log n) space

Exponentiation (べき乗) [Knuth: Computer Algorithms]

- X^{16}
- $16 \equiv 10000_2$ より2進数を4回左シフト

1. まず，1を“SX”，0を“S”で置換
2. 次に，先頭の“SX”を除く．
3. 得られたSとXを「Square」「xをかける」と読む．

- 例： X^{23}
- $23 \equiv 10111_2$
  1. SX S SX SX SX
  2. SSXSXSX
  3. $X^2 X^4 X^5 X^{10} X^{11} X^{22} X^{23}$

“Power Tree” [Knuth: Computer Algorithms]

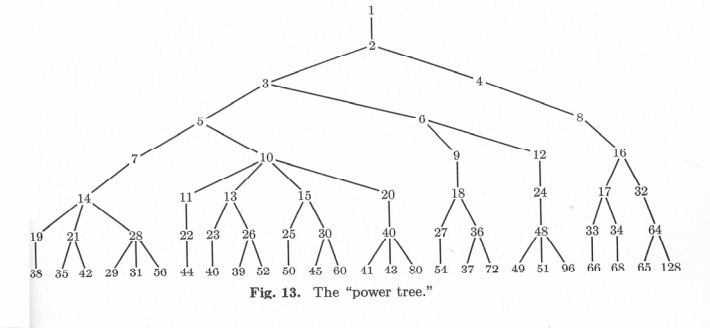


Fig. 13. The “power tree.”