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/09/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

12月8日中間試験（8号館3階大会議室）

http://winnie.kuis.kyoto-u.ac.jp/~fukubaya/AlgDsWiki/

10月20日・本日のメニュー

- 1-1-5 The Substitution Model for Procedure Application
- 1-1-6 Conditional Expressions and Predicates （復習）
- 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

1.1 言語の要素

- expressions（式）
  primitives（基本式）と combinations（合成式）で構成
- means of abstraction（抽象化法）
- Creating procedure objects (手続きの作成法)
- Viewing the rules of evaluation from a computational perspective (計算という観点からの評価法)

Scheme (Lisp) の基本

- 式(expression)は単純なものから構築.
- ほぼすべての式は, 値(value)を持つ.
- 式は評価(evaluate)されて値を返す.
- すべての値には型(type)がある.

このやり方は日常生活では普通にやっている.
ものは分類されて, 使われる.
e.g., 用途別の道具.

4

言語構文・構築子(language constructs)

- Primitives (基本式)
- Means of combination (合成法)
- Means of abstraction (抽象化法)

5

基本式(primitives)

- Self-evaluating primitives(評価すると自分自身の値が返る)
  - Numbers (数): 38, 3.80, 1.4141, 2.3e-4, 3/5
  - Strings (文字列): "moji", "a"
  - Booleans (論理式): #t, #f
- Built-in procedure (組込み手続き)
  - 基本オブジェクト(primitive objects)の処理
  - Numbers(数): +, -, *, /, <, =, <=, …
  - Strings(文字列): string-length, string=?
  - Booleans(論理式): and, or, not, xor, nand
- Names for built-in procedures
  - + ⇒ ＋という組込み手続き

6

合成法(combinations)

- Primitives を使って式を合成
- (+ 3 5)
  (演算子 引数 . . .)

- 評価法
  1. 部分式(subexpressions)を評価し，値を得る.
  2. 演算子(operator)に引数(arguments)を適用.
  - procedure application (手続き適用)という

7

名前と値との連携

- `define` を使って式を合成
- `(define foo (+ 3 5))`
  `foo` の値は `8`
- `(define bar +)`
  `bar` は `+` と同じ手続き
- `(bar 3 5)`

8

言語の要素

- primitive expressions(基本式),
  which represent the simplest entities the language is concerned with,
- means of combination(組合せの方法),
  by which compound elements are built from simpler ones, and
- means of abstraction(抽象化の方法),
  by which compound elements can be named and manipulated as units.

9

1.1.4 Compound Procedures（合成手続き）

- "To square something, multiply it by itself."
- **(define (square x) (* x x))**

  To square something, multiply it by itself.
- This is a compound procedure, of which name is "square". 手続き名前
- (define (<*name*> <*formal parameters*>)
  <*body*>) 仮パラメータ, 本体
- (<*name*> <*parameters*>)
  procedure application 手続き適用

10

1-1-5 The Substitution Model for Procedure Application (置換モデル)

- Vocabulary (語彙) ⇒Primitives
- Syntax (構文) ⇒means of abstractions
- Semantics (意味) ⇒ Viewing the rules of evaluation from a computational perspective (計算という観点からの評価法)

- 手続き適用の評価法として「置換モデル」

11

置換モデルの例による説明

```
(define (square x) (* x x))
(define (sum-of-squares x y)
   (+ (square x) (square y)) )
(define (f a) (sum-of-squares (+ a 1) (* a 2)))
(f 5)
     (sum-of-squares (+ a 1) (* a 2)) に a = 5 を適用
(sum-of-squares (+ 5 1) (* 5 2)) で a を 5 で置換(代入)
     (+ (square x) (square y)) に x = 6, y = 10 を適用
(+ (square 6) (square 10)) で x を 6 で, y を 10 で置換
     (* x x) に x = 6, (* x x) に x = 10 を適用
(+ (* 6 6) (* 10 10)) で x を 10 で置換
(+ 36 100)
136
```

12

2種類の階乗のプログラムを書いてください.

13

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

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

$$\ln n! \approx n\ln n - n$$ for n>>0

26

2種類の階乗(n!)

- トップダウン(top-down)式で計算–線形再帰

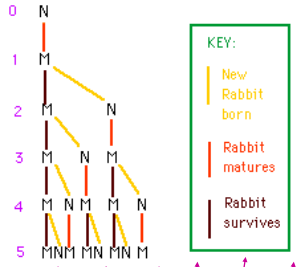
```
(define (factorial n)
  (if (<= n 0)
      1
      (* n (factorial (- n 1))) ))
```

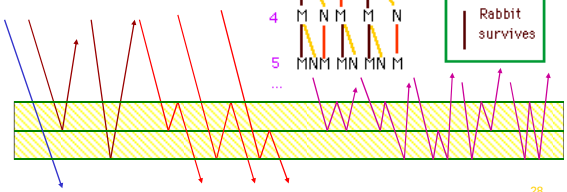
- ボトムアップ(bottom-up)式で計算–線形反復

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

フィボナッチ関数(Fibonacci Function)

- ウサギのつがい(二羽)の数
- 内部反射回数





1.2.2 Fibonacci Function

$$fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ fib(n-1) + fib(n-2), & \text{otherwise} \end{cases}$$

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


1.2.2 Fibonacci Function

$$a_{i+1} = a_i + b_i \qquad a_0 = 1$$

$$b_{i+1} = a_i \qquad \text{ただし,} \qquad b_0 = 0$$

$$fib(n) = b_n$$

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

1.2.2 Fibonacci Function

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

ブロック構造を使わないと,

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

32

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

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

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

ビネの公式を導出する

- Fibonacci数の漸化式

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

- 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
- 母関数(generating function)の解法

ビネの公式：差分方程式による導出

- 漸化式： $F_0 = 0$, $F_1 = 1$, $F_{n+2} = F_n + F_{n+1}$
- 差分方程式 $F_{n+2} - F_n - F_{n+1} = 0$ を解く.

1.特性方程式 $t^2 - t - 1 = 0$を作る

2.特性方程式の根を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

$F_0 = \mathrm{C} + \mathrm{D} = 0$

3.初期値等から、定数項の値を求める.

- 1より $\alpha, \beta = \dfrac{1 \pm \sqrt{5}}{2}$

$F_1 = \mathrm{C}\left(\dfrac{1+\sqrt{5}}{2}\right) + \mathrm{D}\left(\dfrac{1-\sqrt{5}}{2}\right) = 1$

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

$F_2 = \mathrm{C}\left(\dfrac{1+\sqrt{5}}{2}\right)^2 + \mathrm{D}\left(\dfrac{1-\sqrt{5}}{2}\right)^2 = 1$

ビネの公式：母関数による導出(1/3)

- フィボナッチ数列： 0, 1, 1, 2, 3, 5, 8, ...

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

- 母関数 $y = F_0 + F_1 x + F_2 x^2 + F_3 x^3 + F_4 x^4 + F_5 x^5 + \ldots$

$$xy = F_0 x + F_1 x^2 + F_2 x^3 + F_3 x^4 + F_4 x^5 + F_5 x^6 + \ldots$$
$$y - xy = F_0 + (F_1 - F_0)x + (F_2 - F_1)x^2 + (F_3 - F_2)x^3 + \ldots$$
$$\text{左辺} = y(1-x)$$
$$\text{右辺} = x + F_0 x^2 + F_1 x^3 + F_2 x^4 + F_3 x^5 +$$
$$= x + x^2(F_0 x + F_1 x^2 + F_2 x^3 + F_3 x^4 \ldots)$$
$$= x + x^2 y$$
$$\therefore (1-x)y = x + x^2 y$$
$$y = \frac{x}{1 - x - x^2}$$

ビネの公式：母関数による導出(2/3)

- 部分分数への分解 $\dfrac{x}{1-x-x^2} = \dfrac{1}{\alpha - \beta}\left(\dfrac{1}{1-\alpha x} - \dfrac{1}{1-\beta x}\right)$

$$\frac{x}{1-x-x^2} = \frac{x}{1-(\alpha+\beta)x + \alpha\beta x^2}$$

- 両辺を比べて $(\alpha+\beta)=1$, $\alpha\beta=-1$
- これは $t^2 - t - 1 = 0$ の解
- 級数展開をすると $\dfrac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + \ldots$

$$y = \frac{1}{\alpha - \beta}\left(\left(1 + \alpha x + \alpha^2 x^2 + \ldots\right) - \left(1 + \beta x + \beta^2 x^2 + \ldots\right)\right)$$
$$y = \frac{\alpha - \beta}{\alpha - \beta}x + \frac{\alpha^2 - \beta^2}{\alpha - \beta}x^2 + \frac{\alpha^3 - \beta^3}{\alpha - \beta}x^3 + \ldots$$

- x^n の係数はフィボナッチ数F_n

ビネの公式：母関数による導出(3/3)

- x^n の係数はフィボナッチ数F_n

$$F_n = \frac{\alpha^n - \beta^n}{\alpha - \beta}$$

- 一方，

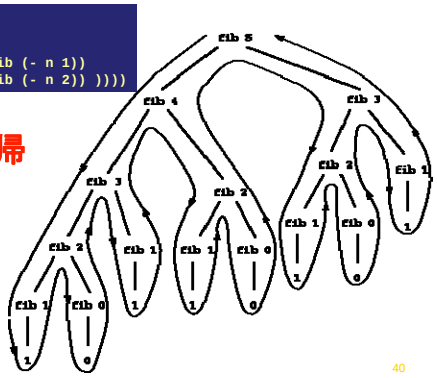
$$\alpha, \beta = \frac{1 \pm \sqrt{5}}{2}$$

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

1.2.2 Fibonacci – Tree Recursion

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

木構造再帰



fib-iter の置換モデルによる実行

```
(fib-i 6)
  (iter  1 0 6)
  (iter  1 1 5)
  (iter  2 1 4)
  (iter  3 2 3)
  (iter  5 3 2)
  (iter  8 5 1)
  (iter 13 8 0)
8
```

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

Linear iterative process
（線形反復プロセス）

Fibの呼ばれる回数　C(n)

- 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

42

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。

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しておく必要がある
  ⇒ スペース量が余分にいる。
- Scheme のループ構造はsyntactic sugar
  - do, repeat, until, for, while

53

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

- 今見てみた置換モデルの評価順序:
  「**適用順序(作用順序, Applicative order)**」
- 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

54

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.

55

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 ))
```

56

1.1.6 Predicates (述語)

- (and <e_1> … <e_n>) 論理積 （左から評価）
- (or <e_1> … <e_n>) 論理和 （左から評価）
- (not <e>) 論理否定

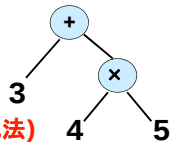
例:

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

57

Ex.1.2 前置記法（prefix notation）

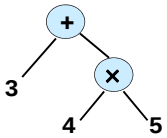
- 式 （演算子 被演算子 …）
  operator operands
- 式の記法
  - 前置記法 （prefix notation , Pollish notation, ポーランド記法)
    + 3 * 4 5
  - 中置記法 （infix notation)
    3 + (4 * 5)
  - 後置記法 （postfix notation, reverse Pollish notation、逆ポーランド記法)
    3 4 5 * +
- 木表現はどれも同じ



58

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/09/IntroAlgDs/

宿題：10月27日正午締切

1. **Fibonacci 関数のプログラムを説明せよ.**
2. **Fibonacci数の一般項を導け**

- Latex で listing style を使って書くこと.
- **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）