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

NoStudent Left Behind

ノーベル経済学賞

NoStudent Left Behind

- アルビン・ロス氏(Harvard Univ. 左)と
- ロイド・シャプレー (UCLA)
- 「安定配分理論とマーケットデザインの実践」
- Stable marriage
- 研修医の配属先決定
- 小中高生の公立校選び
  (日経新聞10月16日朝刊より)
- 岩間先生が日本ではプロ
- 看護婦の勤務シフトに応用
- [cf] サイモン氏(CMU)は人工知能でノーベル賞

LaTeX でレポートを書く

NoStudent Left Behind

```
¥documentclass[a4paper,12pt]{article}
¥usepackage{listings}
¥title{タイトル} ¥author{学籍番号 氏名}
¥begin{document}
¥maketitle
¥section{反復型階乗}
¥lstset{numbers=left,basicstyle=¥small}
¥lstinputlisting{fact.scm}
```

以上で, ファイル "fact.scm" に書かれたプログラムのリスティングが得られる. その下にプログラムの説明を書く.

http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/11/IntroAlgDs/listing.tar.gz にサンプルあり.

```
¥section{再帰型フィボナッチ数}
¥end{document}
```

PDFのファイル名： 学籍番号-名前-回数.pdf
例: 1029233333-HiroshiGOkuno-2.pdf

LaTeX でレポートを書く

NoStudent Left Behind

```
¥documentclass[a4paper,12pt]{article}
¥usepackage{listings}
¥begin{document}
¥section{階乗の定義}
¥begin{equation}
  n! =
    ¥left¥{
      ¥begin{array}{ll}
        1  & ¥mbox{if } n = 0 ¥¥
        n ¥times (n-1)! & ¥mbox{otherwise}
      ¥end{array}
    ¥right.
¥end{equation}
¥end{document}
```

$$n! = \left\{ \begin{array}{ll} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{otherwise} \end{array} \right.$$

4

メディアセンターでの Jakld 使用法

NoStudent Left Behind

[インストール方法]

1. TAのページから **jalkd-on-line.tar.gz** をダウンロード
2. ファイルを展開し，展開して出来たトップディレクトリ（このREADMEが あるディレクトリ）を環境変数 **JAKLD_HOME** にセットする．
   $ export JAKLD_HOME=$HOME/jakld
3. 環境変数 PATH にサブディレクトリ bin を追加する．
   $ export PATH=$JAKLD_HOME/bin:$PATH
4. 環境変数を設定する上記2行を，シェル初期化ファイル（**~/.profile**，**~/.bash_profile** 等）に追加すると次回からのログイン後，すぐに使えます

[使用方法]

1. 端末に「jakld」と打てば起動します（行編集機能つき）．

5

Jakld の 出力方法

NoStudent Left Behind

1. **Command Prompt，cygwin で copy&paste**
2. **Shell の機能を使う**
3. **Output file を明示的に使用**
   - (define out (open-output-file "outfile.txt"))
     out　　同じファイル名のファイルがあると上書き
   - out
     #<port to outfile>
   - (display (fact 7) out)　結果を"outfile.txt"に書き込む
     5040
   - (newline out)
     #t
   - (close-output-port out)
4. call-with-output-file **Output file の非明示的使用**
   - (call-with-output-file "fact101.txt"
     (lambda (out)
     (newline out)
     (display (fact 101) out)
     (newline out)
     ))

6

10月16日・本日のメニュー

1.1 The Elements of Programming

1.1.1 Expressions

1.1.2 Naming and the Environment

1.1.3 Evaluating Combinations

1.1.4 Compound Procedures

1.1.5 The Substitution Model for Procedure Application（復習）

1.1.6 Conditional Expressions and Predicates （復習）

1.2 Procedures and the Processes They Generate

1.2.1 Linear Recursion and Iteration

1.2.2 Tree Recursion

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.

1.1 言語の要素

1. expressions（式）
   - primitives(基本式)と
   - combinations(合成式)で構成
2. means of combination（合成法）
3. Means of abstraction(抽象化法)
4. Creating procedure objects (手続きの作成法)
5. 以上が言語構文 (language syntax)，構築子（language constructs）
6. Viewing the rules of evaluation from a computational perspective (計算という観点からの評価法)
7. 6.が言語意味論（language semantics）

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

11

合成式(combinations)

- Primitives を使って式を合成
  - (+ 3 5)
    (演算子 引数 . . .)
- 評価法(実行方法)
  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

Combination & Abstraction

- `(* 5 5)`
- `(* 38 38)`
- `(* 389.2 389.2)`

Abstraction of values

- `(define foo 389.2)`
- `(* foo foo)`
- `(* foo (+ foo 5) 123)`

Combination

- `(+ 3 (* 53 12.4) 9)`

Abstraction of combinations

```
(define (square x)
    (* x x))
```

- 正書法は下記

```
(define square
    (lambda (x) (* x x)) )
```

- 最初のは簡便記法: Syntax sugar (糖衣錠のこと)

式(expression)は評価されると値を返す

13

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)

14

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

1-1-5 The Substitution Model 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」

16

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
```

17

1-2-1 Linear Recursion and Iteration(復習)

- 階乗の定義(その1)

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

To define n!, if it is non-positive, return 1

otherwise, multiply it by (n-1)!

- どう実行されるか。Substation model で実行
- Linear recursive process (線形再帰的プロセス)
  Nに比例した数の再帰プロセスが生じる
- 積は deferred operations (遅延演算)

factorial の置換モデルによる実行

Linear recursive process (線形再帰的プロセス) Nに比例した数の再帰プロセスが生じる

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 (factorial 0)))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

19

factorial の計算量(Complexity)

NoStudent Left Behind

尺度が重要

factorialの呼ばれる回数
n回 for n! (time complexity)
未実行の * の量
最大 n回 for n! (space complexity)

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 (factorial 0)))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

deferred operationsを先に実行すれば，効率化が可能

末尾再帰(tail recursion) という

22

階乗の別の定義を思い出そう.

1-2-1 Linear Recursion and Iteration

末尾再帰(tail recursion) の効率実行

階乗の定義(その2)

To define n!, n! = 1 * 2 * ··· * n

*product = counter * product*
counter = counter + 1

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

どう実行されるか。Substation model (置換モデル) で実行

NoStudent Left Behind

factorial の置換モデルによる実行

```
(fact 6)
(fact-iter   1 1 6)
(fact-iter   1 2 6)
(fact-iter   2 3 6)
(fact-iter   6 4 6)
(fact-iter  24 5 6)
(fact-iter 120 6 6)
(fact-iter 720 7 6)
720
```

Linear iterative process
(線形反復プロセス)

tail recursion（末尾再帰） がiteration（反復）に!!!

fact から fact-iter へ 自動変換が可能

fact-iter の呼ばれる回数
n回 for n! (time complexity)
未実行の演算の量はない
余分な仮パラメータ
2個 (space complexity)

以下のfactの構造 - Block Structure

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

- 手続きiterは, fact の中で有効。
- 仮パラメータ product,counterは, iterの中で有効
- 外部からは情報隠蔽(information hiding)
- これはオブジェクト指向の1つの特徴.

27

Tail recursion の補足説明

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

- このプログラムは次の翻訳
  n! = (n-1)! * n
- 下記のfactorialとの違いは

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

28

factの置換モデルによる実行

```
(f 6)
(* (f 5) 6)
(* (* (f 4) 5) 6)
(* (* (* (f 3) 4) 5) 6)
(* (* (* (* (f 2) 3) 4) 5
(* (* (* (* (* (f 1) 2) 3
(* (* (* (* (* (* (f 0) 1) 2) 3) 4) 5) 6)
(* (* (* (* (* (* 1 1) 2) 3) 4) 5) 6)
(* (* (* (* (* 1 2) 3) 4) 5) 6)
(* (* (* (* 2 3) 4) 5) 6)
(* (* (* 6 4) 5) 6)
(* (* 24 5))
(* 120)
720
```

末尾再帰ではない.

こういうプログラムは虫の一種!!!

処理系にやさしいプログラムを書くことが不可欠

Tail recursion による高速化

```
SC> (time (null? (factorial 5000)))
total time: 0.729999999999563 secs
user time: 0.690993 secs
system time: 0 secs
#f
SC> (time (null? (f 5000)))
total time: 1.34000000000015 secs
user time: 1.321901 secs
system time: 0 secs
#f
SC> (time (null? (fact 5000)))
total time: 0.720000000001164 secs
user time: 0.701008 secs
system time: 0 secs
#f
```

コンパイルすると factorial（末尾再帰）は，fact（反復）は同じコードに変換される。インタプリタでも可.

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) )
```

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

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

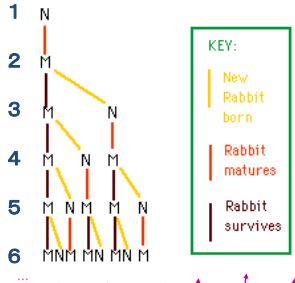
for n>>0

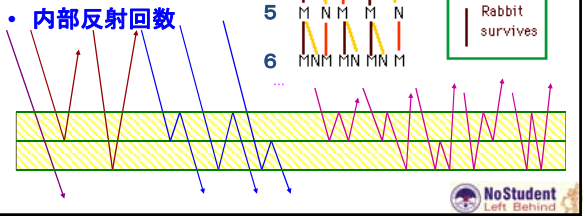
$$\ln n! \approx n\ln n - n$$

NoStudent Left Behind

フィボナッチ関数（Fibonacci Function）

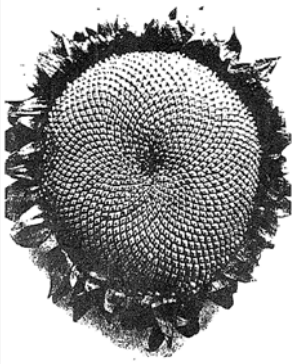
- ウサギのつがい（二羽）の数
- 内部反射回数

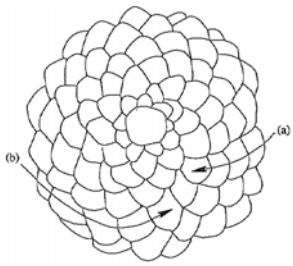




NoStudent Left Behind

フィボナッチ関数（Fibonacci Function）





(a) は13個の時計回り
(b) は8個の反時計回り

NoStudent Left Behind

1.2.2 Fibonacci Function (recursion)

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

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

1.2.2 Fibonacci Function (iteration)

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

$$b_{i+1} = a_i \qquad \text{where} \qquad b_0 = 0$$

$$fib(n) = b_n$$

```
(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) )
```

木構造再帰

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))
  ))
```

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

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

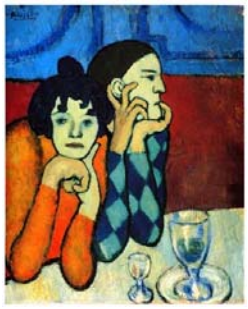
黄金比（Golden Ratio）

- 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

NoStudent Left Behind

THE POWER OF LOGARITHM

計算尺（slide rule, slipstick）
対数による積の計算
乗算→対数→加算
累乗→対数→乗算
2^{30} はいくら
2^{10} →対数→10log2 →3.01
$2^{10} \fallingdotseq 10^3$ →1K
$2^{30} \fallingdotseq 10^9$ →1G
音楽のピッチ
音の知覚



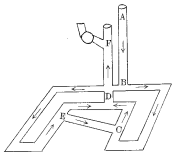
39

SLIDE RULE

NoStudent Left Behind

鉱山用通気計算尺
G.R.I.Mine Ventilation Slide Rule

一資源試型計算尺使用法説明書



ヘンミ計算尺株式会社
東京都千代田区神田駿河台４－４
３－２６３１（代）
（６９０６Ｎ）

- アポロ13
- ミクロの決死隊
- タイタニック

40

Scale Label	Value Relative to C/D
A	X^2
B	X^2
C	X
CF	X x pi
CF/M	X x $\log_{10}$
CI	1 / X
CIF	1 / (pi x X)
D	X
DF	X x pi
DF/M	X x $\log_{10}$
DFM	X x $\log_{10}$
DI	1 / X
DIF	1 / (pi x X)
E	e^x
H	-
H1, H2	Square root of $(1+X^2)$
HC	-

K	X^3
KZ	X x 360
L	log X
Lg	log X
Ln	ln X
LL0	$e^{0.001x}$
LL1 or ZZ1	$e^{0.01x}$
LL2 or ZZ2	$e^{0.1x}$
LL3 or ZZ3	e^x
LL00 or LL/0	e^{-x}
LL01 or LL/1	$e^{-0.1x}$
LL02 or LL/2	$e^{-0.01x}$
LL03 or LL/3	$e^{-0.001x}$
M	log X
P	Square root of $(1-X^2)$
p%	-
P1, P2	Square root of $(1-X^2)$

R	1 / X
R1, R2	Square root of X
S	sin X
Sh1, Sh2	sinh X
Sq1, Sq2	Square root of X
SRT	sin X, tan X
ST	sin, tan X
T	tan X, cotan X
T1, T3	tan X, cotan X
T2	tan X, cotan X
Th	tanh X
V	Volts
W1, W2	Square root of X
Z	X
ZZ1, ZZ2, ZZ3	e^x

大きな数・小さな数

NoStudent Left Behind

deca	da	$\times 10^1$
hecto	h	$\times 10^2$
kilo	K	$\times 10^3$
mega	M	$\times 10^6$
giga	G	$\times 10^9$
tera	T	$\times 10^{12}$
peta	P	$\times 10^{15}$
exa	E	$\times 10^{18}$
zetta	Z	$\times 10^{21}$
yotta	Y	$\times 10^{24}$

deci	d	$\times 10^{-1}$
centi	c	$\times 10^{-2}$
milli	m	$\times 10^{-3}$
micro	μ	$\times 10^{-6}$
nano	n	$\times 10^{-9}$
pico	p	$\times 10^{-12}$
femto	f	$\times 10^{-15}$
atto	a	$\times 10^{-18}$
zepto	z	$\times 10^{-21}$
yocto	y	$\times 10^{-24}$

42

10^1	ten *or* decad
10^2	hundred *or* hecatontad
10^3	thousand *or* chiliad
10^4	myriad
10^5	lac *or* lakh
10^6	million
10^7	crore
10^8	myriamyriad
10^9	milliard *or* billion
10^{12}	trillion
10^{15}	quadrillion
10^{18}	quintillion

10^{21}	sextillion
10^{24}	septillion
10^{33}	decillion
10^{63}	vigintillion
10^{303}	centillion
10^{100}	googol
10^{googol}	googolplex
10^N	*N*plex
10^{-N}	*N*minex

43

224plex 阿伽羅

88plex	無量大数	20plex	垓	4minex	絲 (糸)
80plex	不可思議	16plex	京	5minex	忽
72plex	那由他	12plex	兆	6minex	微
64plex	阿僧祇	8plex	億	7minex	纖 (繊)
56plex	恒河砂	4plex	萬(万)	8minex	沙
48plex	極	3plex	千	9minex	塵
44plex	載	2plex	百	10minex	埃
40plex	正	1plex	十	11minex	渺
36plex	澗	0plex	一	12minex	漠
32plex	溝	1minex	分	13minex	模糊
28plex	穣	2minex	厘	14minex	逡巡
24plex	杼 (禾偏)	3minex	毫(毛)	15minex	須臾

1minex	分	13minex	模糊
2minex	厘	14minex	逡巡 シュンジュン
3minex	毫 (毛) モウ	15minex	須臾シュユ
4minex	絲 (糸) シ	16minex	瞬息シュンソク
5minex	忽 コツ	17minex	弾指ダンシ
6minex	微 ビ	18minex	刹那
7minex	纖 (繊)セン	19minex	六徳 リットク
8minex	沙シャ	20minex	虚
9minex	塵ジン	21minex	空
10minex	埃アイ	22minex	清
11minex	渺ビョウ	23minex	浄
12minex	漠バク		

45

コンピュータ将棋「あらか2010」清水市代女流王将破る (10/11, 東大)

- **情報処理学会**の「トッププロ棋士に勝つ将棋プロジェクト」特製システム
- **阿伽羅(あから)**は10の224乗という数を表し、将棋の局面の数がこの数に近いことに因んで命名された
- **ハードウエア部**
  - 東京大学クラスターマシン:
    - -Intel Xeon 2.80GHz, 4 cores 109台
    - -Intel Xeon 2.40GHz, 4 cores 60台
    - 合計 169台 676 cores
  - バックアップマシン:4プログラムそれぞれについて1台ずつ
    - -CPU: Xeon W3680 3.33GHz 6cores
    - -Memory: 24GB (DDR3 UMB ECC 4GBx6)
    - ※ 使用ハードウェアは当日変更される可能性もあります。(2010年10月6日追記)
- **ソフトウエア部**
  - 構成:国内トップ4プログラムによる**多数決合議法**(4つのプレイヤープログラムに局面を渡し、指し手を受け取り、もっとも多い手を指し手として返す)
    - **合議マネージャ: 開発:電気通信大学伊藤研究室&保木邦仁**
    - **プレイヤー1:「激指」開発:激指開発チーム(鶴岡慶雅、横山大作)**
    - **プレイヤー2:「GPS将棋」開発:チームGPS(田中哲朗、金子知適ほか)**
    - **プレイヤー3:「Bonanza」開発:保木邦仁**
    - **プレイヤー4:「YSS」開発:山下宏**

46

ビネの公式を導出する

Fibonacci数の漸化式

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

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

ビネの公式:線形差分方程式による導出

- **漸化式**: $a_1 = 1$, $a_2 = 1$, $a_{n+2} = a_n + a_{n+1}$
- **一般的な解法** $a_{n+2} - a_n - a_{n+1} = 0$

1. 方程式 $t^2 - t - 1 = 0$ の根をφ, ψとすると

一般項は $a_n = \mathrm{A}\phi^n + \mathrm{B}\psi^n$ と表せる。

ただし、A, B は定数である。

2. 初期値等から、定数項の値を求める。

- 1. より

$$\phi = \frac{1+\sqrt{5}}{2}, \quad \psi = \frac{1-\sqrt{5}}{2}$$

- 2. より

$$a_1 = \mathrm{A}\phi + \mathrm{B}\psi = 1$$
$$a_2 = \mathrm{A}\phi^2 + \mathrm{B}\psi^2 = 1$$

$$a_n = \frac{1}{\sqrt{5}}\left(\phi^n - \psi^n\right)$$

58

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, ...

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

- **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** $\frac{x}{1-x-x^2}=\frac{1}{\alpha-\beta}\left(\frac{1}{1-\alpha x}-\frac{1}{1-\beta x}\right)$

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

- 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** $\frac{1}{1-x}=1+x+x^2+x^3+x^4+x^5+x^6+\ldots$

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

$$y=\frac{\alpha-\beta}{\alpha-\beta}x+\frac{\alpha^2-\beta^2}{\alpha-\beta}x^2+\frac{\alpha^3-\beta^3}{\alpha-\beta}x^3+\ldots$$

- **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，

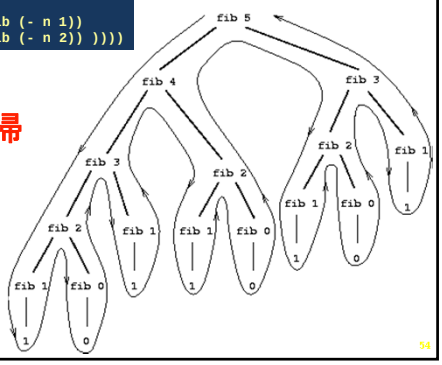
$$\alpha,\beta=\frac{1\pm\sqrt{5}}{2}$$

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

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）

55

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
- $C(0)=C(1)=1$
- $n \geqq 2$ 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 $n \geqq 2$
3. $F(0)=2$, $F(1)=2$

56

Fibの呼ばれる回数 $C(n)=F(n)-1$

- $n \geqq 2$ 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 $n \geqq 2$
- $F(0)=2$, $F(1)=2$

さあ、解いてみてください。

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
- $F(n) = \mathrm{A}\phi^n + \mathrm{B}\psi^n$, φ, ψ は $t^2 - t - 1 = 0$ の根

$$\mathrm{A} = 1 + \frac{1}{\sqrt{5}} = \frac{2}{\sqrt{5}}\phi \qquad \mathrm{B} = 1 - \frac{1}{\sqrt{5}} = -\frac{2}{\sqrt{5}}\psi$$

$$F(n) = \frac{2\phi}{\sqrt{5}}\phi^n - \frac{2\psi}{\sqrt{5}}\psi^n = \frac{2}{\sqrt{5}}(\phi^{n+1} - \psi^{n+1})$$

$$C(n) = \frac{2}{\sqrt{5}}(\phi^{n+1} - \psi^{n+1}) - 1$$

61

フィボナッチ数は

- Fib(n) は $a_n = \frac{1}{\sqrt{5}}\left(\phi^n - \psi^n\right)$
- $\psi = \frac{1-\sqrt{5}}{2}$ より $-0.62 < \psi < 0$
- 従って、fib(n) は $a_n = \frac{1}{\sqrt{5}}\phi^n$ に最も近い整数

62

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**

NoStudent Left Behind

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

NoStudent Left Behind

- 「**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

65

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

NoStudent Left Behind

- 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.

66

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 ))
```

67

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

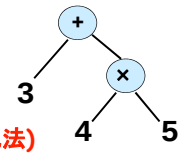
例:

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

68

Ex.1.2 前置記法(prefix notation)

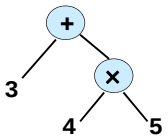
- 式 (演算子 被演算子 …)
  operator operands
- 式の記法
  - 前置記法 (prefix notation , Pollish notation, ポーランド記法)
    + 3 * 4 5
  - 中置記法 (infix notation)
    3 + (4 * 5)
  - 後置記法 (postfix notation, reverse Pollish notation、逆ポーランド記法)
    3 4 5 * +
- 木表現はどれも同じ



69

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/11/IntroAlgDs/

出席表への記入 (5分間)

NoStudent Left Behind

1. 反復型(繰返型)階乗のプログラムを書きなさい.

2. 本日の講義の感想を31文字でまとめなさい.

72

宿題:10月23日正午締切

NoStudent Left Behind

1. 反復型階乗プログラムのファイルを作成せよ. fact-iter.scm
2. (fact-iter 100+ご自分の学籍番号の下1桁) を実行し, 出力結果を求めよ.
3. Fibonacci数の再帰型と反復型手続きのファイルを作成せよ. 1つのファイル中に fib と fib-iter を定義する. fib.scm
4. 2種類のFibonacci数の手続きを実行し, 10, 20, 30 の Fibonacci数を求めよ.
5. 階乗のプログラムの説明と出力結果, および, Fibonacci数のプログラムの説明と出力結果を latexで作成し, pdfを提出.
6. Program ファイルとpdf (学籍番号-名前-回.pdf) を
   SICP-3@zeus.kuis.kyoto-u.ac.jp に送付

- 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)

73