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 Procedures and the Processes They generate
（手続きとそれが生成するプロセス）

奥 乃 博

1. TUT Schemeが公開されました.
   - Windowsは動きます.
   - Linux, Cygwin も動きます.

1

10月25日・本日のメニュー

- 1.2.1 Linear Recursion and Iteration
- 1.2.2 Tree Recursion
- 1.2.3 Orders of Growth
- 1.2.4 Exponentiation
- 1.2.5 Greatest Common Divisors
- 1.2.6 Example: Testing for Primality

2

左上教科書表紙：http://mitpress.mit.edu/images/products/books/0262011530-f30.jpg

1-2-1 Linear Recursion and Iteration

- 階乗の定義

```
(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)!

```
n! = n * (n-1)!
```

どう実行されるか。Substition model で実行

3

factorial の置換モデルによる実行

```
(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
```

4

1-2-1 Linear Recursion and Iteration

- 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。Substition model で実行
- Linear recursive process（線形再帰的プロセス）
（Nに比例して再帰プロセスが生じる）
- 積は deferred operations（遅延演算）

5

factorial の置換モデルによる実行

Deferred operation

```
(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
```

6

1-2-1 Linear Recursion and Iteration

- 階乗の定義（その2）

```
(define (factorial 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) ))
```

To define n!, **n! = 1 * 2 * ⋯ * n**

*product = counter * product*

counter = counter + 1

どう実行されるか。Substition model で実行

factorial の置換モデルによる実行

```
(factorial 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)**

factorial – Block Structure

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

- 手続き iter は、factorial の中で有効。
- 外部からは隠蔽。

Tail recursion の補足説明

```
(define (fact n)
  (if (= n 1)
      1
      (* (fact (- n 1)) n) ))
```

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

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

10

factの置換モデルによる実行

```
(fact 6)
(* (fact 5) 6)
(* (* (fact 4) 5) 6)
(* (* (* (fact 3) 4) 5) 6)
(* (* (* (* (fact 2) 3) 4) 5) 6)
(* (* (* (* (* (fact 1) 2) 3) 4) 5) 6)
(* (* (* (* (* (* (fact 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
```

11

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? (fact 5000)))
total time: 1.34000000000015 secs
user time: 1.321901 secs
system time: 0 secs
#f
SC> (time (null? (fact-iter 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 とfact-iterは同じコードに変換される。

12

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

14

Ex.1.10 Ackermann Function

```
(define (A x y)
  (cond ((= y 0) 0)
        ((= x 0) (* 2 y))
        ((= y 1) 2)
        (else (A (- x 1)
                 (A x (- y 1)) ))))
```

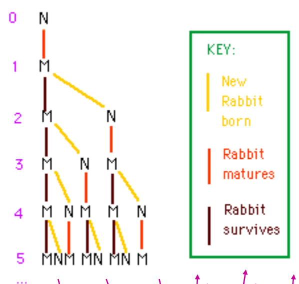
- Ackermann関数は線形再帰ではない！

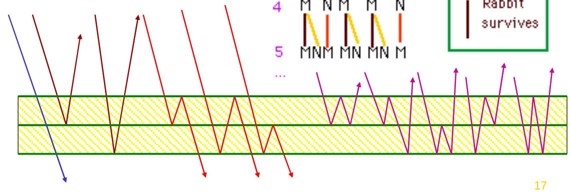
```
(define (Ack m n)
  (cond ((= m 0) (+ n 1))
        ((= n 0) (Ack (- m 1) 1))
        (else (Ack (- m 1)
                   (Ack m (- n 1)) ))))
```

15

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





17

1.2.2 Fibonacci – Tree Recursion（木構造再帰）

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

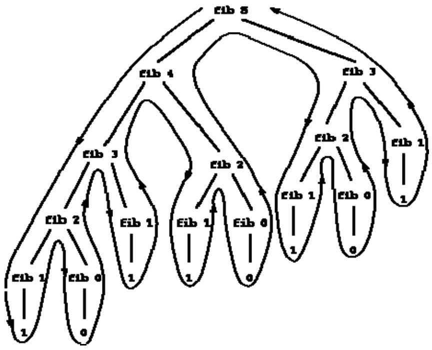
```
(define (fib n)
  (fib-iter 1 0 n))
```

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

18

出所:http://mitpress.mit.edu/sicp/chapter1/fib-tree.gif

1.2.2 Fibonacci – Tree Recursion



19

1.2.2 Fibonacci – Iteration

```
(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

```
(define (fib n)
  (fib-iter 1 0 n))

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

- ボトムアップ(bottom-up)式に計算

20

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) ))
```

21

1.2.3 Order of Growth

R(n) は、ステップ数あるいはスペース量

- *R(n)* が $\Theta(n)$ $k_1 f(n) \geq R(n) \geq k_2 f(n)$
- *R(n)* が $O(n)$ 上限 $R(n) \leq k\ f(n)$
- *R(n)* が $\Omega(n)$ 下限 $R(n) \geq k\ f(n)$

For all $n > n_0$

23

Order of Growth: Examples

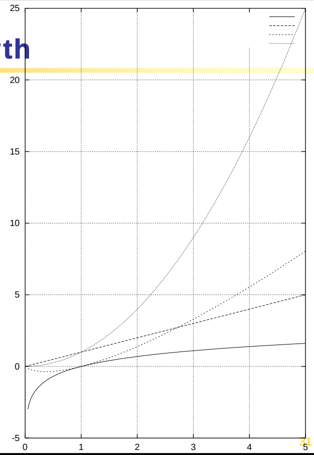
手続き	ステップ数	スペース
factorial	$\Theta(n)$	$\Theta(n)$
fact-iter	$\Theta(n)$	$\Theta(1)$
テーブル参照型fact	$\Theta(1)$	$\Theta(n)$
fib	$\Theta(\phi^n)$	$\Theta(n)$
fib-iter	$\Theta(n)$	$\Theta(1)$
テーブル参照型fib	$\Theta(1)$	$\Theta(n)$

30

Order of Growth

次の式はどの曲線・直線に対応するか

- $y = x$
- $y = x^2$
- $y = \log x$
- $y = x \log x$



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 $\Theta(n)$ steps, $\Theta(n)$ space

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

p =b * p
c = c - 1

- Linear iterative process $\Theta(n)$ steps, $\Theta(1)$ space

36

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 $\Theta(\log n)$ steps, $\Theta(\log n)$ space

37

Exponentiation(べき乗)

- X^{16}
- 16≡10000_2 より2進数を4回左シフト

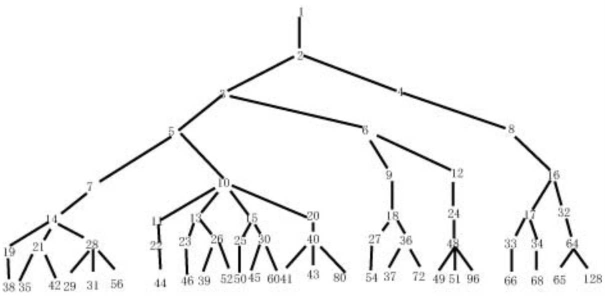
1. まず、1を“SX”、0を“S”で置換し、
2. 次に、先頭の“SX”を除く。
3. 得られたSとXを「Square」「xをかける」と読む。

- 例： X^{23}
- 23≡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}

38

“Power Tree”



39

Greatest Common Divisors（最大公約数）

- ***a*** **mod** ***b*** ＝ ***r*** （modulo 剰余）とすると
- GCD(*a*, *b*) = GCD(*b*, *r*) が成立。
- ユークリッドの互除法

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

41

Modularity Calculus

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

42

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、ただ一つの解がある。

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。

43

Chinese Remainder Theoremの例

$2^{90} \bmod 91$ 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、
  $2^{84} \equiv 1 \pmod{13} \Rightarrow 2^6 \equiv -1 \pmod{13}$
- $13*6 \equiv 1 \pmod 7$
- $7*2 \equiv 1 \pmod{13}$ より、
- $2^{90} \bmod 91 \equiv 1*13*6 - 1*7*2 = 64$

44

Discussion: Fermat's or Wilson's?

1. 単純な素数判定：
2. Fermat's test： p が素数なら
   $\forall a < p,\ a^{(p-1)} \equiv 1 \mod p$
3. Wilson's test: p が素数である必要十分条件は
   $(p-1)! \equiv -1 \mod p$

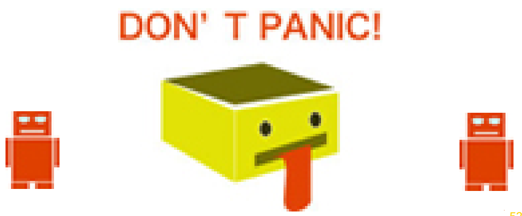
   ちなみに

   $n! \sim (2\pi n)^{1/2}\,(n/e)^n$

51

宿題：10月31日午後5時締切

- Tail recursion は iteration に自動変換
- 宿題は、次の7題：
- Ex.1.9, 1.10, 1.12, 1.14, 1.16, 1.17, 1.19.



52