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

TAの居室は10号館4階奥乃1研，2研
TAのページがオープン，質問箱もあります

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

10月27日・本日のメニュー

- 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 Orders of Growth
- 1.2.4 Exponentiation
- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality

Ex.1.3 Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

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))
```

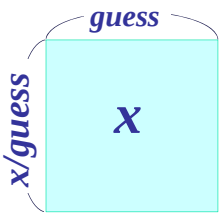
6

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

7

1.1.7 Square Root by Newton's Method

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

と定義すれば,

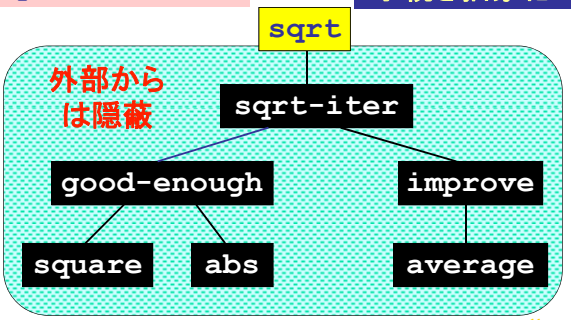
```
(sqrt 9)

(sqrt (+ 100 37))

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

(sqrt (sqrt 1000))
```

9

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



11

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))` x^2
- `(define (square x)`
  `(exp (double (log x))) )`
  `(define (double x) (+ x x))` $e^{2\ln x}$

2.局所名(local names)の隠蔽

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

12

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

```
(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)
```

bound
(束縛)

- **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)**:変数が束縛される式の範囲

13

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

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

静的有効範囲(lexical scoping)

10月27日・本日のメニュー

- 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 Orders of Growth
- 1.2.4 Exponentiation
- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality

16

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

再帰的に考える

- 1個の場合
- *k-1* 個ができたとすると

17

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) )
```

18

Ex.1.10改 Ackermann Function

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

Ackermann関数は線形再帰ではない

```
(define (ackermann m n)
  (if (= m 0)
      (+ n 1)
      (if (= n 0)
          (ackermann (- m 1) 1)
          (ackermann
             (- m 1)
             (ackermann m (- n 1)) ))))
```

22

宿題1:アッカーマン関数の値

提出先は10号館レポート箱，締切9日正午

- (ackermann 0 2)
- (ackermann 1 2)
- (ackermann 2 2)
- (ackermann 3 2)

計算過程を書くこと．以下随意課題

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

23

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

25

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)

26

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) ))
```

28

宿題：11月9日（月）17時　締切

1. Ackermann関数の値を求める
2. Ex.1.5
3. 両替のプログラムを動かす
4. ＄1，＄3の両替は何通り？

DON'T PANIC!



29

10月27日・本日のメニュー

- 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 Orders of Growth**
- 1.2.4 Exponentiation
- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality

31

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

For all $n > n_0$

39

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

40

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
(factorial n)		
(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

43

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
```

44

Fact-iter の置換モデルによる実行

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

- Linear iterative process
  (線形反復プロセス)

46

2種類のFibonacci 関数

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

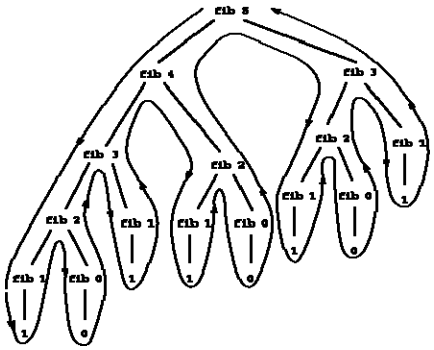
- トップダウン(top-down)式に計算 – 木構造再帰

```
(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

49

fib-iter – Tree Recursion



50

fib-iter の置換モデルによる実行

```
(fib-iter 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
```

- **Linear iterative process**
  (線形反復プロセス)

51

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
(fib n)		
(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

52

Order of Growth の表を埋めよ

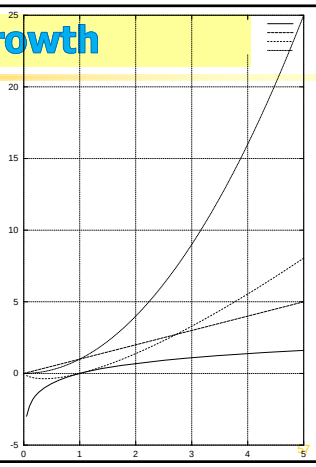
手続き	ステップ数	スペース
factorial		
fact-iter		
テーブル参照型fact		
fib		
fib-iter		
テーブル参照型fib		

54

Order of Growth

次の式はどの曲線・直線に対応するか

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



55

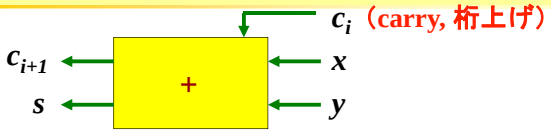
ギリシャ文字

Α	α	alpha
Β	β	beta
Γ	γ	gamma
Δ	δ	delta
Ε	ε	epsilon
Ζ	ζ	zeta
Ψ	η	eta
Θ	θ	theta
Ι	ι	iota
Κ	κ	kappa
Λ	λ	lambda
Μ	μ	mu

Ν	ν	nu
Ξ	ξ	xi
Ο	ο	omicron
Π	π	pi
Ρ	ρ	rho
Σ	σ	sigma
Τ	τ	tau
Υ	υ	upsilon
Φ	φ	phi
Χ	χ	chi
Ψ	ψ	psi
Ω	ω	omega

58

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) ))
```

60

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

63

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

64

Exponentiation(べき乗)

- 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”

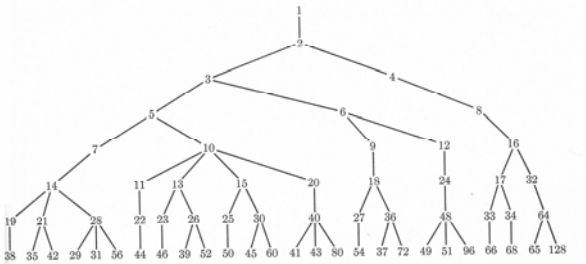


Fig. 13. The “power tree.”