

アルゴリズムとデータ構造入門 7 2009年12月1日

## アルゴリズムとデータ構造入門

### 2.データによる抽象の構築

#### 2.1 高階手続きによる抽象化

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/08/IntroAlgDs/>  
[okuno@i.kyoto-u.ac.jp](mailto:okuno@i.kyoto-u.ac.jp)

12月8日3階大会議室で中間試験

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/>

祝 無事終了  
京都大學  
11月祭

友達は増えましたか  
よく遊び、よく学びましたか

12月8日にその成果を

12月1日・本日のメニュー

### データによる抽象化

#### 2 Building Abstractions with Data

##### 2.1 Introduction to Data Abstraction

2.1.1 Example: Arithmetic Operations for Rational Numbers

2.1.2 Abstraction Barriers

2.1.3 What Is Meant by Data?

2.1.4 Interval Arithmetic

2.2. Hierarchical Data and the Closure Property(階層データ構造と閉包性)



---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---



## ニュートン法を不動点手続きから見直す

```
(define (deriv g)
  (lambda (x) (/ (- (g (+ x dx)) (g x)) dx)))
(define dx 0.00001)

(define (cube x) (* x x x))
(define (newton-transform g)
  (lambda (x) (- x (/ (g x) ((deriv g) x)))))
(define (newtons-method g guess)
  (fixed-point (newton-transform g) guess))
(define (sqrt x)
  (newtons-method (lambda (y) (- (square y) x))
    1.0))
```

$$y = x - \frac{g(x)}{g'(x)}$$

ニュートン法



## Average damping (平均緩和法)

One way to control such oscillations:

Redefine a new function

$$y \mapsto \frac{1}{2} \left( y + \frac{x}{y} \right)$$

```
(define (sqrt x)
  (fixed-point
    (lambda (y) (average y (/ x y)))
    1.0))
```

Average damping (平均緩和法)

5



## 更なる抽象化・first-class procedures

```
(define (fixed-point-of-transform g transform
  guess)
  (fixed-point (transform g) guess))

1st method: 平均緩和法
(define (sqrt x)
  (fixed-point-of-transform
    (lambda (y) (/ x y))
    average-damp
    1.0))

2nd method: ニュートン法
(define (sqrt x)
  (fixed-point-of-transform
    (lambda (y) (- (square y) x))
    newton-transform
    1.0))
```

手続きの構築で何ら差別がない



## 手続き(関数)への演算: 導関数

```
(define dx 0.0001)
(define (ddx f x) 数値微分
  (/ (- (f (+ x dx)) (f x)) dx) )
(ddx square 3) => 6.00010000001205

我々はもっとスマート！導関数という考え方を採用
(define (deriv f)
  (lambda (x)
    (/ (- (f (+ x dx)) (f x)) dx) ))
((deriv square) 3) => 6.00010000001205
((deriv (deriv square)) 3) => 1.99999998
(define (new-ddx f x)
  ((deriv f) x))
```

7

---

---

---

---

---

---

---

---



## 手続き(関数)の合成: 高階導関数

この考え方を発展させ、高階導関数が構築できる

```
(define (compose f g)
  (lambda (x)
    (f (g x)) ))
(define 2nd-deriv (compose deriv deriv))
  ((2nd-deriv square) 3) => 1.9999999878

もちろん手続きの合成も
  ((compose square sqrt) 7) => 7.0
  ((2nd-deriv cos) pi) => 0.999999993922529
(define 3rd-deriv (compose deriv 2nd-deriv))
  ((3rd-deriv sin) pi) => 0.999999960615838
  ((4th-deriv cos) pi) => 1.11022302462516
```

8

---

---

---

---

---

---

---

---



## Let's Play JMC with your num.

```
(define (jmc n)
  (if (> n 100)
      (- n 10)
      (jmc (jmc (+ n 11))))
  ))
```

- 各自、次の式を求めよ

(jmc (modulo 学籍番号 100))

10

---

---

---

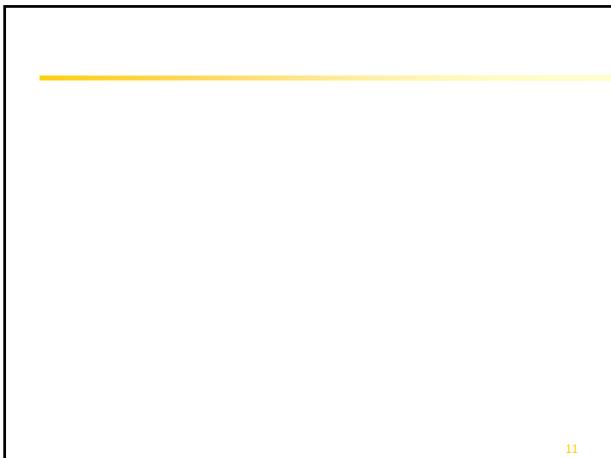
---

---

---

---

---



11

---

---

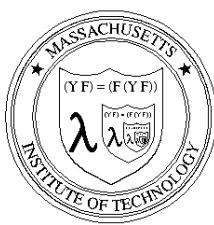
---

---

---

---

### 補足: Fixed Point



```
(define (jmc n)
  (if (> n 100)
      (- n 10)
      (jmc (jmc (+ n 11))))))

(fixed-point jmc 1) => ?
(Y F) = (F (Y F))  Y operator
                (不動点となる手続きを作成)

(Y jmc) = (F (Y jmc))
          = (lambda (n)
              (if (> n 100) (- n 10) ?))
```

12

---

---

---

---

---

---

### Fixed Point Operator F



```
(define (Y F)
  (lambda (s)
    (F (lambda (x) (lambda (x) ((s s) x)))
        (lambda (s) (F (lambda (x) ((s s) x))))))
  )))

再帰呼び出しに無名手続きを使いたい
(Y F) = (F (Y F))

詳しくは、Church numeralの項で
説明。
```

13

---

---

---

---

---

---

「具体から抽象へは行けるが、  
抽象から具体へは行けない」

We can go from Concrete to  
Abstract, while we cannot  
from Abstract to Concrete.

Prof. Yotaro Hatamura

(畠村洋太郎『直観でわかる数学』岩波書店)

---

---

---

---

---

---

## 第2章 データによる抽象の構築

- 第1章は手続き抽象化
  - 基本手続き
  - 合成手続き・手続き抽象化
  - 例:  $\Sigma$ ,  $\Pi$ , accumulate, filtered-accumulate
- 第2章はデータ抽象化
  - 基本データ構造(primitive data structure/object)
  - 合成データオブジェクト(compound data object)
- データ抽象化で手続きの意味(semantics)を拡張
  - 加算(+)でどのようなデータ構造も扱える
  - 基本手続き: 整数+整数、有理数+有理数、実数+実数
  - 合成手続き: 複素数+複素数、行列+行列

15

---

---

---

---

---

---

## 第2章 データ抽象化で学ぶこと

- 抽象化の壁(abstraction barrier)の構築
  - データ構造の実装を外部から隠蔽(blackbox)
- 閉包(closure)
  - 組み合わせを繰り返してもよい
- 従来型インターフェース(conventional interface)
  - Sequence を手続き間インターフェースとして使用
  - ベルトコンベア、生産ライン、UNIXのパイプ
- 記号式(symbolic expression)表現
- 汎用演算(genetic operations)
- データ主導プログラミング(data-directed programming)

16

---

---

---

---

---

---



## 2.1 データ抽象化(data abstraction)

抽象データの4つの基本操作

1. 構成子 (constructor)
2. 選択子 (selector)
3. 述語 (predicate)
4. 入出力 (input/ output)

18

---

---

---

---

---

---



### 2.1.0 Integers(整数)

- 構成子 (constructor)  
 $<n>$  ;  $<n>$  integer
- 選択子 (selector)  
 $<n>$  ;  $<n>$  integer
- 述語 (predicate)  
`(integerp? <x>)`  
`(= <x> <y>)`
- 入出力 (input/output)  
 $<n>$  ;  $<n>$  integer

19

---

---

---

---

---

---



### 2.1.1 Rational Numbers(有理数)

- 構成子 (constructor)  
`(make-rat <n> <d>)`  
 $<n>$  numerator(分子),  
 $<d>$  denominator (分母)
- 選択子 (selector)  
`(numer <x>)`  
`(denom <x>)`  
 $<x>$  rational number
- 述語 (predicate)  
`(rational? <x>)`  
`(equal-rat? <x> <y>)`
- 入出力 (input/output)  
 $<n>/<d>$

21

---

---

---

---

---

---



## 2.1.1 Rational Numbers(有理数)

■ 加算  
(addition)

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2}$$

■ 減算  
(subtraction)

$$\frac{n_1}{d_1} - \frac{n_2}{d_2} = \frac{n_1 d_2 - n_2 d_1}{d_1 d_2}$$

■ 乘算  
(multiplication)

$$\frac{n_1}{d_1} \times \frac{n_2}{d_2} = \frac{n_1 n_2}{d_1 d_2}$$

■ 除算 (division)

$$\frac{n_1}{d_1} \div \frac{n_2}{d_2} = \frac{n_1 d_2}{d_1 n_2}$$

■ 述語

$$n_1 d_2 = n_2 d_1 \quad \Rightarrow \quad \frac{n_1}{d_1} = \frac{n_2}{d_2}$$

22

---

---

---

---

---

---



## Rational Number Operations

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2}$$

$$\frac{n_1}{d_1} - \frac{n_2}{d_2} = \frac{n_1 d_2 - n_2 d_1}{d_1 d_2}$$

```
(define (add-rat x y)
  (make-rat
```

```
(define (sub-rat x y)
  (make-rat
```

23

---

---

---

---

---

---



## Rational Number Operations

$$\frac{n_1}{d_1} \times \frac{n_2}{d_2} = \frac{n_1 n_2}{d_1 d_2}$$

$$\frac{n_1}{d_1} \div \frac{n_2}{d_2} = \frac{n_1 d_2}{d_1 n_2}$$

$$n_1 d_2 = n_2 d_1 \quad \Rightarrow \quad \frac{n_1}{d_1} = \frac{n_2}{d_2}$$

```
(define (mul-rat x y)
  (make-rat
```

```
(define (div-rat x y)
  (make-rat
```

```
(define (equal-rat? x y)
```

25

---

---

---

---

---

---



## Rational Number Representation

```
(define (make-rat n d) (cons n d))  
    n   d      ペア(pair)で表現  
(define (numer x) (car x))  
(define (denom x) (cdr x))  
(define (print-rat x)  
  (newline)  
  (display (numer x))  
  (display "/")  
  (display (denom x))  
  x )
```

27

---

---

---

---

---

---



## Rational Number Reduction(既約化)

```
(define (make-rat n d) (cons n d))  
この表現は曖昧: e.g., 2/3, 4/6, 6/9  
(define (make-rat n d)  
  (let ((g (gcd n d)))  
    (cons (/ n g) (/ d g)) ))
```

既約化: *reducing rational numbers to the lowest terms*

28

---

---

---

---

---

---



## いつの時点で簡略化すべきか?

```
(define (make-rat n d)  
  (let ((g (gcd n d)))  
    (cons (/ n g) (/ d g)) ))
```

両者の長所・短所は?

```
(define (make-rat n d) (cond (n d))  
(define (numer x)  
  (let ((g (gcd (car x) (cdr x))))  
    (/ (car x) g) ))  
(define (denom x)  
  (let ((g (gcd (car x) (cdr x))))  
    (/ (cdr x) g) ))
```

この違いは他のプログラムに影響を与えるか?

---

---

---

---

---

---

## 12月1日・本日のメニュー



### データによる抽象化

2 Building Abstractions with Data

2.1 Introduction to Data Abstraction

2.1.1 Example: Arithmetic Operations for Rational Numbers

#### 2.1.2 Abstraction Barriers

2.1.3 What Is Meant by Data?

2.1.4 Interval Arithmetic

2.2. Hierarchical Data and the Closure Property(階層データ構造と閉包性)

---

---

---

---

---

---

## 既約化を抽象化の壁から見ると

### 有理数を使ったプログラム

プログラム領域での有理数

### add-rat sub-rat mul-等

分子と分母から構成される有理数

### make-rat numer denom

```
(define (make-rat n d)
  (let ((g (gcd n d)))
    (cons (/ n g) (/ d g))))
```

ペアとして構成される有理数

### cons car cdr

ペアの実装法

```
(define (make-rat n d)
  (cons (gcd n d)
        (let ((x (cons (car n) (cdr n))))
          (let ((g (gcd (car x) (cdr x))))
            (/ (car x) g)))
          (cons (/ (car x) g)
                (let ((g (gcd (car x) (cdr x))))
                  (/ (cdr x) g)))))))
```

---

---

---

---

---

---



## Interesting books

1. Douglas R. Hofstadter
2. Reymond Sumulyan
3. Doug Adams
4. Don E. Knuth
5. John H. Conway and Richard K. Guy

---

---

---

---

---

---



## ラッセルのパラドックス

- $A$  のすべての部分集合:  $2^A$

例:  $A = \{a, b, c\}$

$$2^A = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{b,c\}, \{c,a\}, \{a,b,c\}\}$$

- すべての部分集合を含む集合  $S$  を考える

- いずれが成立するか

- $S \in S$

- $S \notin S$

37

---

---

---

---

---

---

## 12月1日・本日のメニュー



### データによる抽象化

2 Building Abstractions with Data

2.1 Introduction to Data Abstraction

2.1.1 Example: Arithmetic Operations for Rational Numbers

2.1.2 Abstraction Barriers

### 2.1.3 What Is Meant by Data?

2.1.4 Interval Arithmetic

2.2. Hierarchical Data and the Closure Property(階層データ構造と閉包性)

---

---

---

---

---

---



## 2.1.3 データって何？ ペア(対、pair)再考

(make-rat n d) の満足すべき条件は

$$\frac{(\text{numer } x)}{(\text{denom } x)} = \frac{n}{d}$$

1. cons, car, cdr を通常のセルで構築

2. 次の手続きで構築

```
(define (cons x y)
  (define (dispatch m)
    (cond ((= m 0) x)
          ((= m 1) y)
          (else (error "Argument not 0 or 1
                        -- CONS" m ))))
  dispatch)
(define (car z) (z 0))           (z 0) => car
(define (cdr z) (z 1))           (z 1) => cdr
```

---

---

---

---

---

---



## ペア(対、pair)を手続きで実現

```
(define (cons x y)
  (define (dispatch m)
    (cond ((= m 0) x)
          ((= m 1) y)
          (else (error "Argument not 0
                        or 1 -- CONS" m ))))
  dispatch)
(define (car z) (z 0))
(define (cdr z) (z 1))
```

- (define foo (cons 10 25))
- (car foo) :
- (cdr foo)

44

---



---



---



---



---



---



---



---



---



---



---



## もっとかっこよくペア(pair)を手続きで実現

```
(define (cons x y)
  (lambda (m) (m x y)))
(define (car z)
  (z (lambda (p q) p)))
(define (cdr z)
  (z (lambda (p q) q)))
■ (define foo (cons 10 25))
■ (car foo) =>
  ((lambda (m) (m 10 25)) (lambda (p q) p))
  => ((lambda (p q) p) 10 25)
  => 10
■ (cdr foo) =>
  ((lambda (m) (m 10 25)) (lambda (p q) q))
  => ((lambda (p q) q) 10 25)
  => 25
```

観測したらデータが得られる⇒  
量子コンピュータ風の計算

45

---



---



---



---



---



---



---



---



---



---



---



## ペアの実装法を抽象化の壁から見ると

—— 有理数を使ったプログラム ——

プログラム領域での有理数

—— add-rat sub-rat mul-等 ——

分子と分母から構成される有理数

—— make-rat numer denom ——

ペアとして構成される有理数

cons car cdr

```
(define (make-rat n d) (cons n d))
(define (numer x) (car x))
(define (denom x) (cdr x))
```

ペアの実装法

```
(define (cons x y)
  (define (dispatch m)
    (cond ((= m 0) x)
          ((= m 1) y)
          (else (error "Argument not 0
                        or 1 -- CONS" m ))))
  dispatch)
(define (car z) (z 0))
(define (cdr z) (z 1))
```

---



---



---



---



---



---



---



---



---



---



---

## 12月1日・本日のメニュー

### データによる抽象化



2 Building Abstractions with Data

2.1 Introduction to Data Abstraction

2.1.1 Example: Arithmetic Operations for Rational Numbers

2.1.2 Abstraction Barriers

2.1.3 What Is Meant by Data?

### 2.1.4 Interval Arithmetic

2.2. Hierarchical Data and the Closure Property(階層データ構造と閉包性)

## 2.1.4 Interval Arithmetic

### Constructor

```
(define (make-interval a b) (cons a b))
```

### Selectors 等

```
(define (upper-bound x)
  (define (lower-bound x)
    (define (equal-interval? x y)
      (define (sub-interval x y)
```

### Interval arithmetic は単位系変換で重要。

- 1in=2.54cm, 1ft=30.48cm, 1yd=0.914m,  
1mile=1.609km, 1nautical mile=1.852km,  
1acre=4047m<sup>2</sup>, 1 UKgal=4.54l, 1 USgal=3.79l,  
1bbl=159l, 1sec=1 nano-century ( $10^{-7}$  year), 1 light  
year=9.461 × 10<sup>12</sup>km (9.461Tkm)

- 1oz=28.3g, 1lb=0.454Kg, 1ct=0.2g

49

## 2.1.4 Interval Arithmetic

```
(define (add-interval x y)
  (make-interval
    (+ (lower-bound x) (lower-bound y))
    (+ (upper-bound x) (upper-bound y)) ))
(define (mul-interval x y)
  (let ((p1 (* (lower-bound x) (lower-bound y)))
        (p2 (* (lower-bound x) (upper-bound y)))
        (p3 (* (upper-bound x) (lower-bound y)))
        (p4 (* (upper-bound x) (upper-bound y))))
    (make-interval (min p1 p2 p3 p4)
                  (max p1 p2 p3 p4))))
(define (div-interval x y)
  (mul-interval x
                (make-interval (/ 1.0 (upper-bound y))
                              (/ 1.0 (lower-bound y))))))
```

## 12月1日・本日のメニュー



### データによる抽象化

2 Building Abstractions with Data

2.1 Introduction to Data Abstraction

2.1.1 Example: Arithmetic Operations for Rational Numbers

2.1.2 Abstraction Barriers

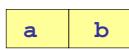
2.1.3 What Is Meant by Data?

2.1.4 Interval Arithmetic

### 2.2. Hierarchical Data and the Closure Property(階層データ構造と閉包性)

## 2.2 Hierarchical Data and the Closure Property(閉包性)

- Pair (cell) 対(セル)  
(cons a b)
- Box-and-pointer notation



- List structure(Backus-Naur Form, BNF記法)

::= は定義 | は代替

- > <list> ::= <null> | (<element> . <element>)
- > <element> ::= <name> | <number> | <list>

- Closure property(閉包性) of cons

52

## 2.2.1 Representing Sequences(シーケンス表現)

- Sequence(列・並び) 1, 2, 3, 4  
(1 2 3 4)



- (cons 1  
      (cons 2  
         (cons 3  
          (cons 4 nil)  
        ))))

- (1 . (2 . (3 . (4 . nil))))

- (list 1 2 3 4)

53



## Sequences表現の簡略化

### ■ Sequence (列・並び) の表現の簡略化

(1 2 3 4 . 5)



$$1. (\text{xxx} . \text{nil}) \Rightarrow (\text{xxx})$$

$$2. (\text{xxx} . (\text{yyy} \dots)) \Rightarrow (\text{xxx} \text{ yyy} \dots)$$

$$(1 . (2 . (3 . (4 . 5))))$$

$$\Rightarrow (1 2 . (3 . (4 . 5)))$$

$$\Rightarrow (1 2 3 . (4 . 5))$$

$$\Rightarrow (1 2 3 4 . 5)$$

54

---

---

---

---

---

---



## 2.2.1 List operations(リスト演算)

```
(define (list-ref items n) ... )
  (list-ref 0 (list 0 1 2))
  0
  (list-ref 2 (list 0 1 2))
  2
  (list-ref 5 (list 0 1 2))
  ()
(define (length items) ... )
  (length ())
  0
  (length (list 1 2 3))
  3
```

55

---

---

---

---

---

---



## Lisp/Scheme Programming十戒

### 1. The First Commandment

Always ask `null?` as the first question in expressing any function.

### 2. The Second Commandment

Use `cons` to build lists.

### 3. The Third Commandment

When building a list, describe the first typical element, and then `cons` it onto the natural recursion.

(Friedman, et al. "The Little Schemer", MIT Press)

56

---

---

---

---

---

---



## 2.2.1 List operations(リスト演算)

```
(list-ref items n)
  if n=0, list-ref is car
  otherwise, (n-1)st item of (cdr items)

(define (list-ref items n)
  (if (= n 0)
      (car items)
      (list-ref (cdr items) (- n 1)) ))
(define (length items)
  (if (null? items)
      0
      (+ 1 (length (cdr items)) ))
```

- cdring down the list (cdr down)
- Tail recursion に注意

57

---

---

---

---

---

---

---

---



## List-ref by cdring down

```
■ (list-ref items n)
  if n=0, list-ref is car
  otherwise, (n-1)st item

■ (define (list-ref items n)
  (if (= n 0)
      (car items)
      (list-ref (cdr items) (- n 1) )))

■ cdring down the list (cdr down)
■ Tail recursion に注意(自動的に iteration に変換)
```

58

---

---

---

---

---

---

---

---



## length by cdring down

```
■ (length items)
  if items is null?, length is 0
  otherwise, 1 + length of the rest

■ (define (length items)
  (if (null? items)
      0
      (+ 1 (length (cdr items)) )))

■ (+ (length (cdr items)) 1) との違い!
```

59

---

---

---

---

---

---

---

---



## length : recursion and iteration versions

```
(define (length items)
  (if (null? items)
      0
      (+ 1 (length (cdr items)) )))

(define (length items)
  (define (iter a count)
    (if (null? a)
        count
        (iter (cdr a) (+ 1 count)) ))
  (iter items 0) )
```

60

---

---

---

---

---

---

---



## Ex2.19 cc change of coins

```
(define us-coins (list 50 25 10 5 1))
(define uk-coins (list 100 50 20 10 5 2 1 0.5))
(define (cc amount coin-values)
  (cond ((= amount 0) 1)
        ((or (< amount 0) (no-more? coin-values))
         0)
        (else
         (+ (cc amount
                  (except-first-denomination coin-values))
            (cc (- amount
                     (first-denomination coin-values) )
                  coin-values )))))
(define (except-first-denomination coins)
  (cdr coins))
(define (first-denomination coins)
  (car coins))
(define (no-more? coins)
  (null? coins))
```

62

---

---

---

---

---

---

---



## 宿題はありません

来週は中間試験です。  
範囲は、2.2章(Ex.2.19)までの教科書  
と講義内容。  
持ち込み一切なし。

DON'T PANIC!



---

---

---

---

---

---

---