

# アルゴリズムとデータ構造入門

## 1.5 Formulating Abstractions with Higher-Order Procedures

### 2.データによる抽象の構築

#### 2 Building Abstractions with Data

奥 乃 博



- 世の中のシステムは楽観主義 (optimistic) と悲観主義 (pessimistic) の中庸 (trade-off) で設計されている。

1

祝 京大チーム2年  
連続世界大会出場

Team Name (Affiliation)	No. of Completed Problems	Time	Final standing
Neural (Chongqing Jiao Tong University)	8	1070	*(*)1
emerald (Kyoto University)	7	123	1
Powderpuss TRV (Fudan University)	7	1153	2
oXtreme QHC-4 w/ System 360ry (The University of Tokyo)	6	676	3
HHI Masters (The University of Tokyo)	6	752	*(*)4
El Oriental Tamasu University	6	820	4
whiskHusk (The University of Tokyo)	6	888	*(*)5
agp (The University of Tokyo)	6	1000	*(*)9
Enjoy (Peking University)	5	294	5
DuoR3nD.org (Tokyo Institute of Technology)	5	296	6
Plan (The University of Aizu)	3	330	7
LawheadPeace (Tokyo Institute of Technology)	3	345	*(*)8
echelon (Kyoto University)	3	272	*(*)6
Finalizer (City University of Hong Kong)	3	351	8
ZeroF.NET (Osaka University)	3	356	9
Hwa Hwa (Hwa Hwa Institute of Technology)	3	367	10
BEYOND THE EDGE (Kyoto University)	3	403	*(*)11
MOCC (Tokyo University of Agriculture and Technology)	3	604	11

2

<http://www.teu.ac.jp/icpc/regional/results.html>



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))

4



## 11月8日・本日のメニュー

- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values
- 2 Building Abstractions with Data
  - 2.1 Introduction to Data Abstraction
    - 2.1.1 Example: Arithmetic Operations for Rational Numbers

---

---

---

---


---

---

---

---

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



### 1.3.3 Procedures as General Methods

#### Finding roots of equations by the half-interval method (区間二分法)

```

(define (search f neg-point pos-point)
  (let ((midpoint (average neg-point pos-point)))
    (if (close-enough? neg-point pos-point)
        midpoint
        (let ((test-value (f midpoint)))
          (cond ((positive? test-value)
                 (search f neg-point midpoint))
                ((negative? test-value)
                 (search f midpoint pos-point))
                (else midpoint)))))))

```

---

---

---


---

---

---

---

---



### Finding roots of equations by the half-interval method

```

(define (close-enough? x y)
  (< (abs (- x y)) 0.001))

(define (half-interval-method f a b)
  (let ((a-value (f a))
        (b-value (f b)))
    (cond ((and (negative? a-value) (positive? b-value))
           (search f a b))
          ((and (negative? b-value) (positive? a-value))
           (search f b a))
          (else
           (error "Values are not of opposite sign" a b)))
  )))

```

**L:** 開始時の区間長、**T:** 誤差許容度、  
**ステップ数:**  $\Theta(\log(L/T))$

---

---

---


---

---

---

---

---



### Finding fixed points of functions(不動点)

```

(define tolerance 0.00001)
(define (fixed-point f first-guess)
  (define (close-enough? v1 v2)
    (< (abs (- v1 v2)) tolerance))
  (define (try guess)
    (let ((next (f guess)))
      (if (close-enough? guess next)
          next
          (try next))))
  (try first-guess))

```

**xが不動点**  $x = f(x)$   $f(x), f(f(x)), f(f(f(x))),$

---

---

---


---

---

---

---

---

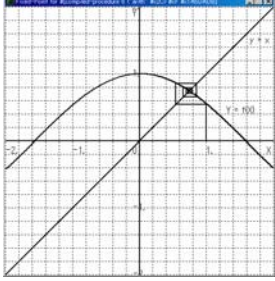
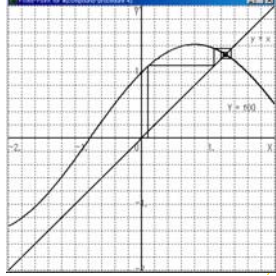


### Finding fixed points of functions(不動点)

```

(fixed-point cos 1.0) (fixed-point
  (lambda (y)
    (+ (sin y) (cos y)))
  0.1 )

```

---

---

---


---

---

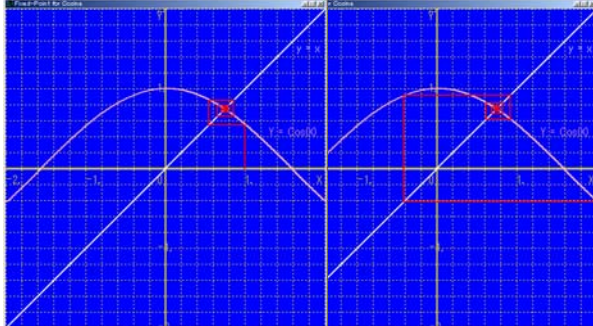
---

---

---



### (fixed-point cos 1.0)& (fixed-point cos 2.0)



---

---

---


---

---

---

---

---



### 不動点が求まらない場合がある $\sqrt{x}$

```

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

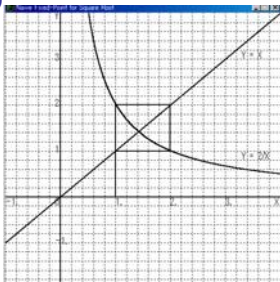
```

$$y \mapsto \frac{x}{y}$$

(sqrt 2) を実行すると

$1 \rightarrow 2 \rightarrow 1$

$(y_1 \rightarrow y_2 \rightarrow y_1)$



---

---

---


---

---

---

---

---



### 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)

15

---

---

---


---

---

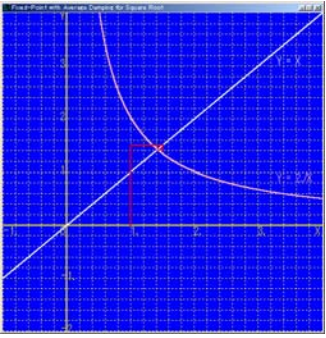
---

---

---



### Fixed Point with Average Damping



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

Average damping  
平均緩和法

16

---

---

---

---

---

---

---

---



## 11月8日・本日のメニュー

- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values
- 2 Building Abstractions with Data
  - 2.1 Introduction to Data Abstraction
    - 2.1.1 Example: Arithmetic Operations for Rational Numbers

17

---

---

---

---

---

---

---

---



## 1.3.4 Procedures as Returned Values

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

平均緩和法を不動点の観点から眺めると

```
(define (average-damp f)
  (lambda (x) (average x (f x))))
```

```
((average-damp square) 10)
```

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

```
(define (cube-root x)
  (fixed-point
    (average-damp (lambda (y) (/ x (square y))))
    1.0))
```

average-damp で  
統一的に  
捉えるこ  
とが可能

18

---

---

---

---

---

---

---

---



## Newton's method & differentiation

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

```
(define (cube x) (* x x x))
((deriv cube) 5)
```

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

ニュートン法

```
(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))
```

20

---

---

---

---

---

---

---

---



## 更なる抽象化・first-class procedures

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

1<sup>st</sup> method

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

2<sup>nd</sup> method

```
(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

21

---

---

---

---

---

---

---

---



## First-class citizen (第1級市民)

第1級市民の“権利と特権”

- 変数で名前をつけることができる。
- 手続きへ引数として渡すことができる。
- 手続きの結果として返すことができる。
- データ構造の中に含めることができる。

*Microsoft Longhorn will make RAW  
'first class citizen.'*

The Inquirer, Wed. Jun-8, 2005

23

---

---

---

---

---

---

---

---



## 手続き(関数)への演算: 導関数

- (define dx 0.0001)
- (define (ddx f x)
 (/ (- (f (+ x dx)) (f x)) dx) )
- (ddx square 3) ⇒ 6.00010000001205
- 我々もっとスマートだった! 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) ⇒ 1.99999999878
- (define (new-ddx f x)
 ((deriv f) x) )

24

---

---

---

---

---

---

---

---

## 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
```

25

## 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) ?))
```

26

<http://libraries.mit.edu/archives/mithistory/seal/>

## 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。

27



## What is this instrument?

- 計算尺
- 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
- $2^{10} \rightarrow \text{対数} \rightarrow 10\log 2 \rightarrow 3$
- $2^{10} \div 10^3 \rightarrow 1\text{K}$
- $2^{30} \div 10^9 \rightarrow 1\text{G}$



29

---

---

---

---

---

---

---

---



## 大きな数・小さな数

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	$\mu$	$\times 10^{-6}$
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	$\times 10^{-24}$

30

---

---

---

---

---

---

---

---



$10^1$	ten <i>or</i> decad
$10^2$	hundred <i>or</i> hecatontad
$10^3$	thousand <i>or</i> chiliad
$10^4$	myriad
$10^5$	lac <i>or</i> lakh
$10^6$	million
$10^7$	crore
$10^8$	myriamyrriad
$10^9$	milliard <i>or</i> 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
$10^{\text{googol}}$	googolplex
$10^N$	$N$ plex
$10^{-N}$	$N$ minex

32

---

---

---

---

---

---

---

---





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 (織)
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	穰	2minex	厘	14minex	逡巡
24plex	杼 (禾偏)	3minex	毫(毛)	15minex	須臾

33

---

---

---

---

---

---

---

---

---

---



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	彈指ダンシ
6minex	微 ビ	18minex	殺那
7minex	纖 (織) 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	淨
12minex	漠 バク		

34

---

---

---

---

---

---

---

---

---

---



## ギリシヤ文字

A	$\alpha$	alpha	N	$\nu$	nu
B	$\beta$	beta	Ξ	$\xi$	xi
Γ	$\gamma$	gamma	O	$o$	omicron
Δ	$\delta$	delta	Π	$\pi$	pi
E	$\epsilon$	epsilon	R	$\rho$	rho
Z	$\zeta$	zeta	Σ	$\sigma$	sigma
Y	$\eta$	eta	T	$\tau$	tau
Θ	$\theta$	theta	Υ	$\upsilon$	upsilon
I	$\iota$	iota	Φ	$\phi$	phi
K	$\kappa$	kappa	X	$\chi$	chi
Λ	$\lambda$	lambda	Ψ	$\psi$	psi
M	$\mu$	mu	Ω	$\omega$	omega

35

---

---

---

---

---

---

---

---

---

---



## 11月8日・本日のメニュー

- 1.3.3 Procedures as General Methods
- 1.3.4 Procedures as Returned Values
- 2 Building Abstractions with Data
- 2.1 Introduction to Data Abstraction
- 2.1.1 Example: Arithmetic Operations for Rational Numbers

36

---

---

---

---

---

---

---



## 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) が拡大
  - 加算 (+)
  - 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

37

---

---

---

---

---

---

---



## 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
- 慣用インターフェース (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、トヨタの生産ライン、UNIXのパイプ
- 記号式 (symbolic expression) による表現
- 汎用演算 (generic operations)
- データ主導プログラミング (data-directed programming)

38

---

---


---

---

---

---

---



## 2.1 データ抽象化(data abstraction)

- 抽象データの4つの基本操作

- 構成子(constructor)
- 選択子(selector)
- 述語(predicate)
- 入出力(input/ output )

40

---

---

---


---

---

---

---

---



### 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>`

41

---

---

---


---

---

---

---

---



### 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)**  

$$\frac{n_1}{d_1} - \frac{n_2}{d_2} = \frac{n_1 d_2 - n_2 d_1}{d_1 d_2}$$
- 乗算 (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**  

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

42

---

---

---

---

---

---

---

---



## 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 (+ (* (number x) (denom y))
                (* (number y) (denom x)) )
            (* (denom x) (denom y)) ))
```

```
(define (sub-rat x y)
  (make-rat (- (* (number x) (denom y))
                (* (number y) (denom x)) )
            (* (denom x) (denom y)) ))
```

43

---

---

---

---

---

---

---

---



## Rational Number Operations

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

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

$$n_1 d_2 = n_2 d_1$$



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

```
(define (mul-rat x y)
  (make-rat (* (number x) (number y))
            (* (denom x) (denom y) )))
```

```
(define (div-rat x y)
  (make-rat (* (number x) (denom y))
            (* (denom x) (number y) )))
```

```
(define (equal-rat? x y)
  (= (* (number x) (denom y))
     (* (number y) (denom x))))
```

44

---

---

---

---

---

---

---

---



## Rational Number Representation

```
(define (make-rat n d) (cons n d))
```

n	d
---	---

ペア(pair)で表現

```
(define (number x) (car x))
```

```
(define (denom x) (cdr x))
```

```
(define (print-rat x)
  (newline)
  (display (number x))
  (display "/" )
  (display (denom x))
  x )
```

45

---

---

---

---

---

---

---

---



## Rational Number Reduction(既約化)

```
(define (make-rat n d) (cond n d))
```

これでは、表現が曖昧になる

```
(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*

46

---

---

---

---

---

---

---



## 宿題: 11月14日午後5時締切

- 宿題は、次の9問:
- Ex.1.35, 1.36, 1.37, 1.40, 1.41, 1.42, 1.43, 1.44, 2.1

DON' T PANIC!



53

---

---

---

---

---

---

---