アルゴリズムとデータ構造入門

1.5 Formulating Abstractions with Higher-Order Procedures

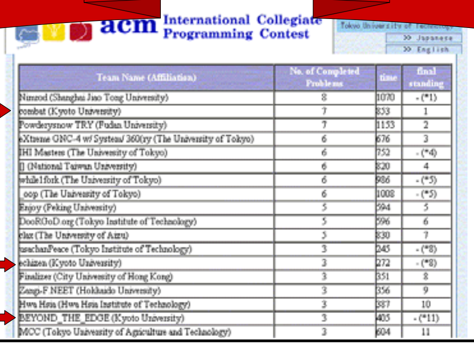
2.データによる抽象の構築

2 Building Abstractions with Data

奥 乃 博

1. 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。

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
Nimrod (Shanghai Jiao Tong University)	8	1070	- (*1)
combat (Kyoto University)	7	853	1
Powderysnow TRY (Fudan University)	7	1153	2
eXtreme GNC-4 w/ System/ 360(ry (The University of Tokyo)	6	676	3
IHI Masters (The University of Tokyo)	6	752	- (*4)
[] (National Taiwan University)	6	820	4
while1fork (The University of Tokyo)	6	986	- (*5)
_oop (The University of Tokyo)	6	1008	- (*5)
Enjoy (Peking University)	5	594	5
DooRGoD.org (Tokyo Institute of Technology)	5	596	6
clax (The University of Aizu)	5	830	7
usachanPeace (Tokyo Institute of Technology)	3	245	- (*8)
echizen (Kyoto University)	3	272	- (*8)
Finalizer (City University of Hong Kong)	3	351	8
Zangi-F NEET (Hokkaido University)	3	356	9
Hwa Hsia (Hwa Hsia Institute of Technology)	3	387	10
BEYOND_THE_EDGE (Kyoto University)	3	405	- (*11)
MCC (Tokyo University of Agriculture and Technology)	3	604	11

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))
```

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

5

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))))))
```

6

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: $\Theta(\log(L/T))$

7

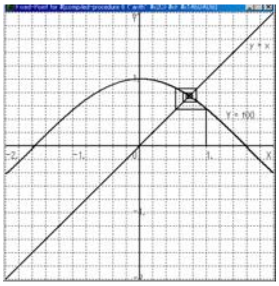
Finding fixed points of functions（不動点）

```
(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))
```

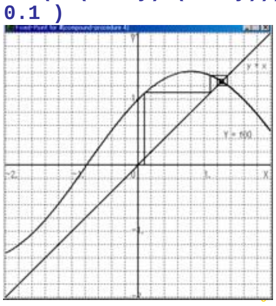
Xが不動点 x = f(x) $f(x), f(f(x)), f(f(f(x))),$

Finding fixed points of functions（不動点）

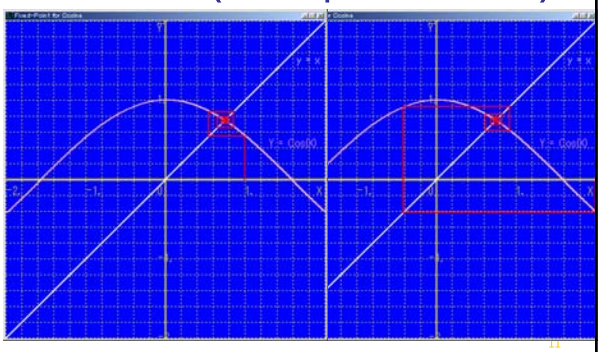
```
(fixed-point cos 1.0)
```



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



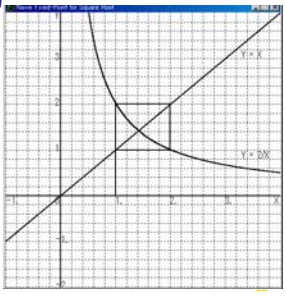
(fixed-point cos 1.0)& (fixed-point cos 2.0)



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 → 2 → 1

(y_1 → y_2 → y_1)

Average damping（平均緩和法）

One way to control such ocillations:

Redefine a new function

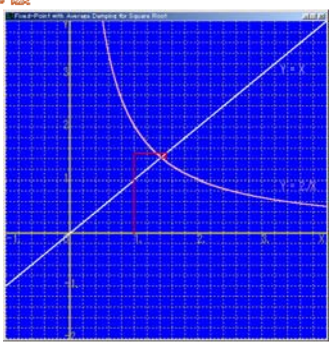
$$y \mapsto \frac{1}{2}\left(y + \frac{x}{y}\right)$$

```
(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)

(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

20

更なる抽象化· first-class procedures

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

1st method

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

手続きの構築で何ら差別がない

2nd method

```
(define (sqrt x)
  (fixed-point-of-transform
     (lambda (y) (- (square y) x))
     newton-transform
     1.0 ))
```

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
- 我々は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)` ⇒ 1.9999999878
- ```
  (define (new-ddx f x)
    ((deriv f) x) )
  ```

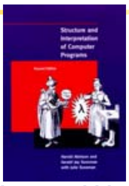
24
  ```

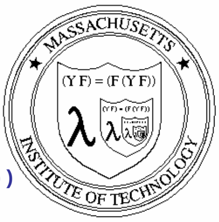
手続き(関数)の合成: 高階導関数

- 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

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} →対数→10log2
- $2^{10} \fallingdotseq 10^3$ →1K
- $2^{30} \fallingdotseq 10^9$ →1G

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	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	$\times 10^{-24}$

30

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

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	繊 (纎)
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

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	弾指ダンシ
6minex	微 ビ	18minex	刹那
7minex	繊 (纎)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		

34

ギリシャ文字

A	α	alpha	N	ν	nu
B	β	beta	Ξ	ξ	xi
Γ	γ	gamma	O	o	omicron
Δ	δ	delta	Π	π	pi
E	ϵ	epsilon	R	ρ	rho
Z	ζ	zeta	Σ	σ	sigma
Y	η	eta	T	τ	tau
Θ	θ	theta	Υ	υ	upsilon
I	ι	iota	Φ	ϕ	phi
K	κ	kappa	X	χ	chi
Λ	λ	lambda	Ψ	ψ	psi
M	μ	mu	Ω	ω	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： Σ, Π, 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
- 慣用インタフェース(conventional interface)
  - Sequence を手続き間インタ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(genetic 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

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

40

2.1.1 Rational Numbers(有理数)

- 構成子(constructor)
  (make-rat <*n*> <*d*>)
  <*n*> numenator(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) $$\frac{n_1}{d_1}+\frac{n_2}{d_2}=\frac{n_1d_2+n_2d_1}{d_1d_2}$$
- 減算 (subtraction) $$\frac{n_1}{d_1}-\frac{n_2}{d_2}=\frac{n_1d_2-n_2d_1}{d_1d_2}$$
- 乗算 (multiplication) $$\frac{n_1}{d_1}\times\frac{n_2}{d_2}=\frac{n_1n_2}{d_1d_2}$$
- 除算 (division) $$\frac{n_1}{d_1}\div\frac{n_2}{d_2}=\frac{n_1d_2}{d_1n_2}$$
- 述語 $$n_1d_2=n_2d_1 \Rightarrow \frac{n_1}{d_1}=\frac{n_2}{d_2}$$

42

Rational Number Operations

$$\frac{n_1}{d_1}+\frac{n_2}{d_2}=\frac{n_1d_2+n_2d_1}{d_1d_2}$$

$$\frac{n_1}{d_1}-\frac{n_2}{d_2}=\frac{n_1d_2-n_2d_1}{d_1d_2}$$

```
(define (add-rat x y)
  (make-rat (+ (* (numer x) (denom y))
               (* (numer y) (denom x)) )
            (* (denom x) (denom y)) ))

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

43

Rational Number Operations

$$\frac{n_1}{d_1}\times\frac{n_2}{d_2}=\frac{n_1n_2}{d_1d_2}$$

$$\frac{n_1}{d_1}\div\frac{n_2}{d_2}=\frac{n_1d_2}{d_1n_2}$$

$$n_1d_2=n_2d_1 \Rightarrow \frac{n_1}{d_1}=\frac{n_2}{d_2}$$

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

(define (div-rat x y)
  (make-rat (* (numer x) (denom y))
            (* (denom x) (numer y) )))

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

44

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 )
```

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 gcd) (/ d gcd)) ))
```

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

53