アルゴリズムとデータ構造入門 2005年11月15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 Building Abstractions with Data

奥 乃 博

「具体から抽象へは行けるが、
抽象から具体へは行けない」

（畑村洋太郎『直観でわかる数学』岩波書店）

1

11月15日・本日のメニュー

- 2 Building Abstractions with Data
- 2.1 Introduction to Data Abstraction
  - 2.1.2 Abstraction Barriers
  - 2.1.3 What Is Meant by Data?
  - 2.1.4 Extended Exercise: Interval Arithmetic
- 2.2. Hierarchical Data and the Closure Property
  - 2.2.1 Representing Sequences
  - 2.2.2 Hierarchical Structures

2

Rational Number Representation

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

n	d

ペア（pair）で表現

```
(define (numer x) (car x)) ; numerator

(define (denom x) (cdr x)) ; denominator

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

7

TA開設質問掲示板から: Read-Eval-Print-Loop (REPL) での出力

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

(print-rat
  (make-rat 1 2))
1/22

(print-rat
 (add-rat
  (print-rat (make-rat 1 2))
  (print-rat (make-rat 2 3))
  ))
1/2
2/3
error
```

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

(print-rat
  (make-rat 1 2))
1/2(1 . 2)

(print-rat
 (add-rat
  (print-rat (make-rat 1 2))
  (print-rat (make-rat 2 3))
  ))
1/2
2/3
7/6(7 . 6)
```

8

2.1.2 Abstraction barrier（抽象化の壁）

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

ペアの実装法

10

抽象化の壁から考察すると

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

と次の定義との違いは？

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

11

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

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

ペアとして構成される有理数

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

cons car cdr

ペアの実装法

ペアの実装法の抽象化がない

13

Rational Number Reduction（既約化）

小学校で分数は**既約化しなさい**と習った。

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

では、どの時点で既約化するか？

1. 構築子(make-rat)で。
2. 選択子(numer, denom)で。
3. 既約化は他のプログラムに影響を与えるか

14

Rational Number Reduction（簡約化）

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

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？

15

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

ペアとして構成される有理数

```
(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) ))
```

cons car cdr

ペアの実装法

18

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

```
(numer x)       n
---------   =  ---
(denom x)       d
```

1. cons, car, cdr を通常のセルで構築

2. 次の手続きで構築

```
(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))
```

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)

22

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)
```

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

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

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))
```

11月15日・本日のメニュー

- 2 Building Abstractions with Data
- 2.1 Introduction to Data Abstraction
- 2.1.2 Abstraction Barriers
- 2.1.3 What Is Meant by Data?
- **Intermission (Church Numerals)**
- 2.1.4 Extended Exercise: Interval Arithmetic
- 2.2. Hierarchical Data and the Closure Property
- 2.2.1 Representing Sequences
- 2.2.2 Hierarchical Structures
- 2.2.3 Sequences as Conventional Interfaces

26

かっこよく自然数も手続きで実現

```
(define c0 (lambda (f) (lambda (x) x)))
(define (%succ c)
  (lambda (f) (lambda (x) (f ((c f) x))))
```

この自然数の表現を Church numerals （チャーチ数）という

```
(define c1 (%succ c0))
   ⇒ (lambda (f) (lambda (x) (f ((zero f) x)))
   ⇒ (lambda (f)
        (lambda (x)
          (f (((lambda (f) (lambda (x) x)) f) x)) ))
   ⇒ (lambda (f)
        (lambda (x)
          (f ((lambda (x) x) x)) ))
   ⇒ (lambda (f) (lambda (x) (f x)))
```

自然数が0とf（後続関数）で定義

- (define c1 (lambda (f) (lambda (x) (f x))))
- (define c2 (lambda (f) (lambda (x) (f (f x)))))
- (define c3 (lambda (f) (lambda (x) (f (f (f x))))))

27

Church NumeralsとTAO

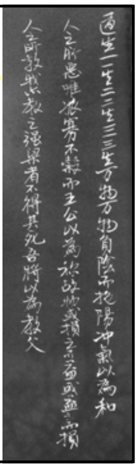
『老子』第42節の

「道 (TAO) から一が生まれ, 一から二が生まれ, 二から三が生まれ, 三から万物が生まれ, 云々」

Tao produced the one.
The one produced the two.
The two produced the three.
And the three produced the ten thousand things.
The ten thousand things carry the yin and embrace the yang, and through the blending of the material force they achieve harmony. *Tao-te Ching, 42, Lao Tzu.*

と符合するものです.
改良型Backus 記法が導入された Revised Report on the Algorithmic Language ALGOL 68 の113ページ にも上記の一節が引用されています.

INTREAL :: SIZETY integral ; SIZETY real.
SIZETY :: long LONGSETY ; short SHORTSETY ; EMPTY.



Operations on Church Numerals

```
(define c0 (lambda (f) (lambda (x) x)))
(define (%succ c)
  (lambda (f) (lambda (x) (f ((c f) x))))
```

- (define c1 (lambda (f) (lambda (x) (f x))))
- (define c2 (lambda (f) (lambda (x) (f (f x)))))
- (define c3 (lambda (f) (lambda (x) (f (f (f x))))))

```
(define (%add n m)
  (lambda (f) (lambda (x) ((m f) ((n f) x) ))))
(define (%multiply n m)
    (lambda (f) (lambda (x) ((n (m f)) x))) )
(define (%power n m)
    (lambda (f) (lambda (x) (((m n) f) x))) )
```

30

Church Numerals の出力

- 実際の動きを見るために，入出力の関数を定義しましょう．

```
(define (c->n c)  ; 出力
    ((c (lambda (x) (+ 1 x))) 0) )
(define (n->c n)  ; 入力
   (if (> n 0)
       (%succ (n->c (- n 1)))
        c0 ))
```

- 上記の c->n はメモリを大量に消費し遅い。高速版は:

```
(define (c->n c) ((c 1+) 0))
```

- では実験．

```
1.(c->n (%add (n->c 5) (n->c 3)))
2.(c->n (%multiply (n->c 5) (n->c 3)))
3.(c->n (%power (n->c 5) (n->c 3)))
4.(c->n (%add (%power c2 c3)
              (%multiply c3 (n->c 4)) ))
```

31

減算･比較･Fixed Point Operator F

- 減算は難しい。まず、大小比較を定義する。
- 再帰呼び出しに無名手続きを使う必要がある。
- Y オペレータを使う。
  (Y F) = (F (Y F))

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

詳細は Web ページにあります。
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/04/IntroAlgDs/

32

Fermat's Last Theorem

$$x^n + y^n = z^n$$

I have discovered a truly remarkable proof which this margin is too small to contain.

n>2 で x, y, z を満たす正整数

Euler's Conjecture

$$a^4 + b^4 + c^4 \neq d^4$$

が成立するだろう。

$$95800^4 + 217519^4 + 414560^4 = 422481^4$$

1987年発見

34

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)) )))
```

2.1.4 Interval Arithmetic（宿題）

- **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。**
  - 1*in*≒2.54*cm*, 1*ft*≒30.48*cm*, 1*yd*≒0.914*m*, 1*mile*≒1.609*km*, 1*nautical mile*≒1.852*km*, 1*acre*≒4047m^2, 1 *UKgal*≒4.54*l*, 1*USgal*≒3.79*l*, 1*bbl*≒159*l*, 1 π *sec*≒1 *nano-century* (10^{-7} *year*) , 1 *light year*≒9.461×10^{12}km（10^{13}*km*=10T*km*）
  - 1*oz*≒28.3*g*, 1*lb*≒0.454*Kg*, 1*ct*≒0.2*g*

2.2 Hierachical Data and the Closure Property

- **Pair (cell)**

```
(cons a b)
```

- **Box-and-pointer notation**

a	b

- **List structure**

:= は定義

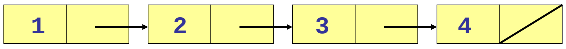
| は代替

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

- **Closure property of cons**

2.2.1 Representing Sequences

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



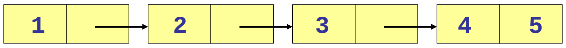

```
(cons 1
  (cons 2
    (cons 3
      (cons 4 nil)
  )))
```

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

41

Sequences表現の簡略化

- Sequence（列・並び） の表現の簡略化
  (1 2 3 4)



1. (xxx . nil) ⇒ (xxx)
2. (xxx . (yyy ···))⇒(xxx yyy ···)

```
  (1 . (2 . (3 . (4 . 5))))
⇒ (1 2 . (3 . (4 . 5)))
⇒ (1 2 3 . (4 . 5))
⇒ (1 2 3 4 . 5)
```

42

2.2.1 List operations

- `(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)) ))
  ```
- ```
  (define (length items)
    (if (null? items)
        0
        (+ 1 (length (cdr items))) ))
  ```
- cdring down the list (cdr down)
- Tail recursion に注意

43

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) )
```

44

cons up while cdring down

```
■(define (append list1 list2)
   (if (null? list1)
       list2
       (cons (car list1)
             (append (cdr list1) list2)
       )))

■(define (reverse items)
   (if (null? items)
       nil
       (append (reverse (cdr items))
               (list (car items)) )))
```

45

Formal parameterの指定

- (define (f x y . Z) <*body*>)
- 例 (f 1 2 3 4 5 6)
- x ⇐ 1, y ⇐ 2, z ⇐ (3 4 5 6)
- (define (g . w) <*body*>)
- 例 (g 1 2 3 4 5 6)
- w ⇐ (1 2 3 4 5 6)

- (define f (lambda (x y . z)
  <*body*>))
- (define g (lambda w <*body*>))

46

Arguments with dotted-tail notation

- `(define (f x y . z)`
  `<body> )`

- ```
  (define (sum . items)
    (define (iter items result)
       (if (null? items)
           result
           (iter (cdr items)
                 (+ result (car items))
           )))
    (iter items 0) )
  ```

- ```
  (define (sum . items)
    (define (recur items)
       (if (null? items)
           0
           (+ (car items) (recur (cdr items))) ))
    (recur items) )
  ```

47

Apply transformation to each element

- ```
  (define (scale-list items factor)
     (if (null? items)
         nil
         (cons (* (car items) factor)
               (scale-list (cdr items) factor) )))
  ```
  ⇓
- ```
  (define (map proc items)
     (if (null? items)
         result
         (cons (proc (car items))
               (map proc (cdr items)) )))
  ```

- `(map abs (list -10 2.5 -11.6 17))`
  ⇒ `(10 2.5 11.6 17)`
- ```
  (define (scale-list items factor)
    (map (lambda (x) (* x factor)) items) ))
  ```
- ```
  (map (lambda (x y) (+ x (* 2 y)))
       (list 1 2 3)
       (list 4 5 6) )
  ```
  ⇒ `(9 12 15)`

48

宿題：11月21日午後5時締切

- 宿題は、次の10問：
- **Ex.2.2, 2.3**, 2.4, 2.5, 2.6, **2.7, 2.8, 2.9, 2.17,** 2.20
- 下線は本日の講義の単なる復習

55