アルゴリズムとデータ構造入門 第十三回 課題

1029-24-9540 山崎啓太郎

February 7, 2013

1 組み込み数・有理数・複素数システムを統合した汎用算術システム

```
1  ; Thanks to @sielico
2  ; He is the main writer :-)
3
4  (define square (lambda (x) (* x x)))
5  (define attach-tag (lambda (type-tag contents)
6    (if (eq? type-tag 'scheme-number)
7      contents
8      (cons type-tag contents))))
9  (define type-tag (lambda (datum)
10   (cond
11     ((pair? datum) (car datum))
12     ((number? datum) 'scheme-number)
13     (else (error "Bad tagged datum -- TYPE-TAG" datum)))))
14 (define contents (lambda (datum)
15   (cond
16     ((pair? datum) (cdr datum))
17     ((number? datum) datum)
18     (else (error "Bad tagged datum -- CONTENTS" datum)))))
19
20 (define add (lambda (x y) (apply-generic 'add x y)))
21 (define sub (lambda (x y) (apply-generic 'sub x y)))
22 (define mul (lambda (x y) (apply-generic 'mul x y)))
23 (define div (lambda (x y) (apply-generic 'div x y)))
24 (define raise (lambda (x) (apply-generic 'raise x)))
25 (define drop (lambda (x) (apply-generic 'drop x)))
26
```

```
27 ;------------------SCHEME NUMBER PACKAGE-----------------------
28 ;;scheme-number
29 (define install-scheme-number-package (lambda ()
30   (define tag (lambda (x) (attach-tag 'scheme-number x)))
31   (put 'add '(scheme-number scheme-number)
32     (lambda (x y) (tag (+ x y))))
33   (put 'sub '(scheme-number scheme-number)
34     (lambda (x y) (tag (- x y))))
35   (put 'mul '(scheme-number scheme-number)
36     (lambda (x y) (tag (* x y))))
37   (put 'div '(scheme-number scheme-number)
38     (lambda (x y) (tag (/ x y))))
39   (put 'make '(scheme-number)
40     (lambda (x) (tag x)))
41   (put 'equ? '(scheme-number scheme-number)
42     (lambda (x y) (eq? x y)))
43   (put '=zero? '(scheme-number)
44     (lambda (x) (eq? x 0)))
45   (put 'raise '(scheme-number)
46     (lambda (x) (make-rational (contents x) 1)))
47   'done
48 ))
49
50 (define make-scheme-number (lambda (n)
51   ((get 'make '(scheme-number)) n)))
52
53 ;------------------RATIONAL PACKAGE-----------------------
54 ;;rational
55 (define install-rational-package (lambda ()
56   ;private
57   (define numer (lambda (x) (car x)))
58   (define denom (lambda (x) (cdr x)))
59   (define make-rat (lambda (n d)
60     (let ((g (gcd n d)))
61       (cons (/ n g) (/ d g)))))
62   (define numer (lambda (x) (car x)))
63   (define denom (lambda (x) (cdr x)))
64   (define add-rat (lambda (x y)
65     (make-rat (+ (* (numer x) (denom y))
66                  (* (numer y) (denom x)))
67               (* (denom x) (denom y)))))
68   (define sub-rat (lambda (x y)
69     (make-rat (- (* (numer x) (denom y))
```

```
70                    (* (numer y) (denom x)))
71                 (* (denom x) (denom y)))))
72   (define mul-rat (lambda (x y)
73     (make-rat (* (numer x) (numer y))
74                 (* (denom x) (denom y)))))
75   (define div-rat (lambda (x y)
76     (make-rat (* (numer x) (denom y))
77                 (* (denom x) (numer y)))))
78   (define equ? (lambda (x y) (apply-generic 'equ? x y)))
79   (define =zero? (lambda (x) (apply-generic '=zero? x)))
80
81   ;others
82   (define tag (lambda (x) (attach-tag 'rational x)))
83   (put 'add '(rational rational)
84        (lambda (x y) (simplification (tag (add-rat x y)))))
85   (put 'sub '(rational rational)
86        (lambda (x y) (simplification (tag (sub-rat x y)))))
87   (put 'mul '(rational rational)
88        (lambda (x y) (simplification (tag (mel-rat x y)))))
89   (put 'div '(rational rational)
90        (lambda (x y) (simplification (tag (div-rat x y)))))
91   (put 'make '(rational)
92        (lambda (n d) (tag (make-rat n d))))
93   (put 'equ? '(rational rational)
94     (lambda (x y) (eq? x y)))
95   (put '=zero? '(rational)
96     (lambda (x) (eq? (numer x) 0)))
97   (put 'raise '(rational)
98     (lambda (x) (make-real (/ (numer x) (denom x)))))
99   (put 'drop '(rational)
100    (lambda (x)
101      (if (= (denom x) 1)
102        (make-scheme-number (numer x))
103        (tag x))))
104  'done
105 ))
106
107 (define make-rational (lambda (n d)
108   ((get 'make '(rational)) n d)))
109
110 ;----------------REAL PACKAGE----------------------
111 ;;real
112 (define install-real-package (lambda ()
```

```
113   (define tag (lambda (x) (attach-tag 'real x)))
114   (put 'add '(real real)
115     (lambda (x y) (simplification (tag (+ x y)))))
116   (put 'sub '(real real)
117     (lambda (x y) (simplification (tag (- x y)))))
118   (put 'mul '(real real)
119     (lambda (x y) (simplification (tag (* x y)))))
120   (put 'div '(real real)
121     (lambda (x y) (simplification (tag (/ x y)))))
122   (put 'make '(real)
123     (lambda (x) (tag x)))
124   (put 'equ? '(real real)
125     (lambda (x y) (eq? x y)))
126   (put '=zero? '(real)
127     (lambda (x) (eq? x 0)))
128   (put 'raise '(real)
129     (lambda (x) (make-complex-from-real-imag (contents x) 0)))
130   (put 'drop '(real)
131     (lambda (x)
132       (if (integer? (contents x))
133         (make-rational (contents x) 1)
134         (tag x))))
135   'done
136 ))
137
138 (define make-real (lambda (n)
139   ((get 'make '(real)) n)))
140
141 ;------------------COMPLEX PACKAGE----------------------
142 ;rectangular
143 (define install-rectangular-package (lambda ()
144   ;private
145   (define real-part (lambda (z) (car z)))
146   (define imag-part (lambda (z) (cdr z)))
147   (define magnitude (lambda (z)
148    (sqrt (+ (square (real-part z))
149             (square (imag-part z))))))
150   (define angle (lambda (z)
151    (atan (imag-part z) (real-part z))))
152   (define make-from-real-imag (lambda (x y) (cons x y)))
153   (define make-from-mag-ang (lambda (r a)
154     (cons (* r (cos a)) (* r (sin a)))))
155   (define equ? (lambda (x y) (apply-generic 'equ? x y)))
```

```
156   (define =zero? (lambda (x) (apply-generic '=zero? x)))
157
158   ;others
159   (define tag (lambda (x) (attach-tag 'rectangular x)))
160   (put 'real-part '(rectangular) real-part)
161   (put 'imag-part '(rectangular) imag-part)
162   (put 'magnitude '(rectangular) magnitude)
163   (put 'angle '(rectangular) angle)
164   (put 'make-from-real-imag '(rectangular)
165     (lambda (x y) (tag (make-from-real-imag x y))))
166   (put 'make-from-mag-ang '(rectangular)
167     (lambda (r a) (tag (make-from-mag-ang r a))))
168   (put 'equ? '(rectangular rectangular)
169     (lambda (x y) (eq? x y)))
170   (put '=zero? '(rectangular)
171     (lambda (x) (eq? 0)))
172   'done
173 ))
174
175 ;polar
176 (define install-polar-package (lambda ()
177   ;private
178   (define magnitude (lambda (z) (car z)))
179   (define angle (lambda (z) (cdr z)))
180   (define make-from-mag-ang (lambda (r a) (cons r a)))
181   (define real-part (lambda (z)
182     (* (magnitude z) (cos (angle z)))))
183   (define imag-part (lambda (z)
184     (* (magnitude z) (sin (angle z)))))
185   (define make-from-real-imag (lambda (x y)
186     (cons (sqrt (+ (square x) (square y)))
187           (atan y x))  ))
188   (define equ? (lambda (x y) (apply-generic 'equ? x y)))
189   (define =zero? (lambda (x) (apply-generic '=zero? x)))
190
191   ;others
192   (define tag (lambda (x) (attach-tag 'polar x)))
193   (put 'real-part '(polar) real-part)
194   (put 'imag-part '(polar) imag-part)
195   (put 'magnitude '(polar) magnitude)
196   (put 'angle '(polar) angle)
197   (put 'make-from-real-imag '(polar)
198     (lambda (x y) (tag (make-from-real-imag x y))))
```

```
199     (put 'make-from-mag-ang '(polar)
200       (lambda (r a) (tag (make-from-mag-ang r a))))
201     (put 'equ? '(polar polar)
202       (lambda (x y) (eq? x y)))
203     (put '=zero? '(polar)
204       (lambda (x) (eq? (magnitude z) 0)))
205     'done
206   ))
207
208   (define real-part (lambda (z) (apply-generic 'real-part z)))
209   (define imag-part (lambda (z) (apply-generic 'imag-part z)))
210   (define magnitude (lambda (z) (apply-generic 'magnitude z)))
211   (define angle (lambda (z) (apply-generic 'angle z)))
212
213   ;complex
214   (define install-complex-package (lambda ()
215     ;from rectangular and polar
216     (define make-from-real-imag (lambda (x y)
217       ((get 'make-from-real-imag '(rectangular)) x y)))
218     (define make-from-mag-ang (lambda (r a)
219       ((get 'make-from-mag-ang '(polar)) r a)))
220
221     ;private
222     (define add-complex (lambda (z1 z2)
223       (make-from-real-imag (+ (real-part z1) (real-part z2))
224                            (+ (imag-part z1) (imag-part z2)))))
225     (define sub-complex (lambda (z1 z2)
226       (make-from-real-imag (- (real-part z1) (real-part z2))
227                            (- (imag-part z1) (imag-part z2)))))
228     (define mul-complex (lambda (z1 z2)
229       (make-from-mag-ang (* (magnitude z1) (magnitude z2))
230                          (+ (angle z1) (angle z2)))))
231     (define div-complex (lambda (z1 z2)
232       (make-from-mag-ang (/ (magnitude z1) (magnitude z2))
233                          (- (angle z1) (angle z2)))))
234     (define equ? (lambda (x y) (apply-generic 'equ? x y)))
235     (define =zero? (lambda (x) (apply-generic '=zero? x)))
236     (define (drop z)
237       (if (= (imag-part z) 0)
238         (make-real (real-part z))
239         (tag z) ))
240
241     ;others
```

```
242   (define tag (lambda (z) (attach-tag 'complex z)))
243   (put 'real-part '(complex) real-part)
244   (put 'imag-part '(complex) imag-part)
245   (put 'magnitude '(complex) magnitude)
246   (put 'angle '(complex) angle)
247   (put 'add '(complex complex)
248        (lambda (z1 z2) (simplification (tag (add-complex z1 z2)))))
249   (put 'sub '(complex complex)
250        (lambda (z1 z2) (simplification (tag (sub-complex z1 z2)))))
251   (put 'mul '(complex complex)
252        (lambda (z1 z2) (simplification (tag (mul-complex z1 z2)))))
253   (put 'div '(complex complex)
254        (lambda (z1 z2) (simplification (tag (div-complex z1 z2)))))
255   (put 'make-complex-from-real-imag '(complex)
256        (lambda (x y) (tag (make-from-real-imag x y))))
257   (put 'make-complex-from-mag-ang '(complex)
258        (lambda (r a) (tag (make-from-mag-ang r a))))
259   (put 'equ? '(complex complex)
260     (lambda (x y) (eq? x y)))
261   (put '=zero? '(complex)
262     (lambda (x) (apply-generic '=zero? x)))
263   (put 'drop '(complex)
264     (lambda (x) (drop x)))
265   'done
266 ))
267
268 (define make-complex-from-real-imag (lambda (x y)
269   ((get 'make-complex-from-real-imag '(complex)) x y)))
270 (define make-complex-from-mag-ang (lambda (r a)
271   ((get 'make-complex-from-mag-ang '(complex)) r a)))
272
273 (define rectangular? (lambda (z)
274   (eq? (type-tag (contents z)) 'rectangular)))
275 (define polar? (lambda (z)
276   (eq? (type-tag (contents z)) 'polar)))
277
278 ;----------------------------------------
279
280 (define apply-generic (lambda (op . args)
281   (let ((type-tags (map type-tag args)))
282     (let ((proc (get op type-tags)))
283       (if proc
284         (apply proc (map contents args))
```

```
285         (if (= (length args) 2)
286           (let* ((type1 (car type-tags))
287                  (type2 (cadr type-tags))
288                  (a1 (car args))
289                  (a2 (cadr args))
290                  (higher-tag (higher type1 type2)))
291             (cond ((eq? type1 type2)
292                         (apply-generic op a1 a2))
293                   ((eq? higher-tag type1)
294                         (apply-generic op a1 (raise a2)))
295                   ((eq? higher-tag type2)
296                         (apply-generic op (raise a1) a2))
297                   (else
298                         (error "No method for these types -- APPLY-GENERIC"
299                                (list op type-tags)))))
300           (display args)))))))
301
302 (define higher (lambda (tag1 tag2)
303   (define tags '(complex real rational scheme-number))
304   (define number (lambda (tag)
305     (let ((order (member tag tags)))
306       (if (eq? order #f)
307         (error "Such type do not exist" tag)
308         (length order)))))
309
310   (let ((order1 (number tag1))
311         (order2 (number tag2)))
312     (if (< order1 order2)
313       tag2
314       tag1))))
315
316 (define simplification (lambda (x)
317     (if (or
318             (eq? (type-tag x) 'scheme-number)
319             (eq? (type-tag x) (type-tag (drop x)))
320         )
321         x
322         (simplification (drop x)))
323 ))
324
325 ;install
326 (install-scheme-number-package)
327 (install-rational-package)
```

```
328 (install-real-package)
329 (install-complex-package)
330 (install-polar-package)
331 (install-rectangular-package)
```

2 実行例

(make-scheme-number 5) => 5
(make-rational 5 6) => (rational 5 . 6)
(make-real 5.6) => (real . 5.6)
(make-complex-from-real-imag 33 5) => (complex rectangular 33 . 5)
(make-complex-from-mag-ang 5 2) => (complex polar 5 . 2)
(raise (make-scheme-number 5)) => (rational 5 . 1)
(raise (make-rational 5 3)) => (real . 1.6666666666666667)
(raise (make-real 5.3)) => (complex rectangular 5.3 . 0)
(drop (make-complex-from-real-imag 33 0)) => (real . 33)
(drop (make-real 32)) => (rational 32 . 1)
(drop (make-rational 32 1)) => 32
(add (make-complex-from-real-imag 33 5) (make-rational 4 5)) => (complex rectangular 33.8 . 5)
(sub (make-complex-from-real-imag 33 5) (make-real 4.5)) => (complex rectangular 28.5 . 5)
(mul (make-scheme-number 5) (make-complex-from-mag-ang 5 2)) => (complex polar 25.0 . 2.0)
(div (make-rational 33 5) (make-real 4.5)) => (real . 1.4666666666666666)
(div (make-rational 33 1) (make-real 3)) => 11
(add (make-complex-from-real-imag 5 3) (make-complex-from-real-imag 5 -3)) => 10
(add (make-complex-from-real-imag 2 1) (make-complex-from-real-imag 2.5 -1)) => (real . 4.5)

3 説明

make-scheme-number:整数を生成する関数
make-rational:分母と分子から有理数を生成する関数
make-real:実数を生成する関数
make-complex-from-real-imag:実部と虚部から複素数を生成する関数
make-complex-from-mag-ang:半径と角度から複素数を生成する関数
また、型階層は scheme-number->rational->real->complex であり、raise 関

数によって型階層を上げることができる。
逆に、型階層を下げる場合は drop 関数を使う。
四則演算用の関数が用意されておりそれぞれ、add(和)、sub(差)、mul(積)、div(商) である。
また、内部で simplification 関数が呼ばれることにより、演算結果の簡略化が行われている。