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

January 18, 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 (define square (lambda (x) (* x x)))
 2 (define attach-tag (lambda (type-tag contents)
 3   (if (eq? type-tag 'scheme-number)
 4     contents
 5     (cons type-tag contents))))
 6 (define type-tag (lambda (datum)
 7   (cond
 8     ((pair? datum) (car datum))
 9     ((number? datum) 'scheme-number)
10     (else (error "Bad tagged datum -- TYPE-TAG" datum)))))
11 (define contents (lambda (datum)
12   (cond
13     ((pair? datum) (cdr datum))
14     ((number? datum) datum)
15     (else (error "Bad tagged datum -- CONTENTS" datum)))))
16
17 (define add (lambda (x y) (apply-generic 'add x y)))
18 (define sub (lambda (x y) (apply-generic 'sub x y)))
19 (define mul (lambda (x y) (apply-generic 'mul x y)))
20 (define div (lambda (x y) (apply-generic 'div x y)))
21 (define raise (lambda (x) (apply-generic 'raise x)))
22 (define drop (lambda (x) (apply-generic 'drop x)))
23
24 ;--------------------SCHEME NUMBER PACKAGE----------------------
25 ;;scheme-number
26 (define install-scheme-number-package (lambda ()
```

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

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

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

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

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

```
242   (put 'magnitude '(complex) magnitude)
243   (put 'angle '(complex) angle)
244   (put 'add '(complex complex)
245        (lambda (z1 z2) (simplification (tag (add-complex z1 z2)))))
246   (put 'sub '(complex complex)
247        (lambda (z1 z2) (simplification (tag (sub-complex z1 z2)))))
248   (put 'mul '(complex complex)
249        (lambda (z1 z2) (simplification (tag (mul-complex z1 z2)))))
250   (put 'div '(complex complex)
251        (lambda (z1 z2) (simplification (tag (div-complex z1 z2)))))
252   (put 'make-complex-from-real-imag '(complex)
253        (lambda (x y) (tag (make-from-real-imag x y))))
254   (put 'make-complex-from-mag-ang '(complex)
255        (lambda (r a) (tag (make-from-mag-ang r a))))
256   (put 'equ? '(complex complex)
257     (lambda (x y) (eq? x y)))
258   (put '=zero? '(complex)
259     (lambda (x) (apply-generic '=zero? x)))
260   (put 'drop '(complex)
261     (lambda (x) (drop x)))
262   'done
263 ))
264
265 (define make-complex-from-real-imag (lambda (x y)
266   ((get 'make-complex-from-real-imag '(complex)) x y)))
267 (define make-complex-from-mag-ang (lambda (r a)
268   ((get 'make-complex-from-mag-ang '(complex)) r a)))
269
270 (define rectangular? (lambda (z)
271   (eq? (type-tag (contents z)) 'rectangular)))
272 (define polar? (lambda (z)
273   (eq? (type-tag (contents z)) 'polar)))
274
275 ;-----------------------------------------
276
277 (define apply-generic (lambda (op . args)
278   (let ((type-tags (map type-tag args)))
279     (let ((proc (get op type-tags)))
280       (if proc
281         (apply proc (map contents args))
282         (if (= (length args) 2)
283           (let* ((type1 (car type-tags))
284                  (type2 (cadr type-tags))
```

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

```
328  (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。