アルゴリズムとデータ構造入門 第九回課題

1029-24-9540 山崎啓太郎

December 6, 2012

1 微分システム

```
1  (define variable? (lambda (x) (symbol? x)))
2  (define same-variable? (lambda (x y)
3          (and (variable? x) (eq? x y))
4  ))
5  (define =number? (lambda (exp num)
6          (and (number? exp) (= exp num))
7  ))
8
9  (define sum? (lambda (x)
10         (and (pair? x) (eq? (car x) '+) )
11 ))
12 (define subtract? (lambda (x)
13         (and (pair? x) (eq? (car x) '-))
14 ))
15 (define product? (lambda (x)
16         (and (pair? x) (eq? (car x) '*))
17 ))
18 (define division? (lambda (x)
19         (and (pair? x) (eq? (car x) '/))
20 ))
21 (define exponentiation? (lambda (x)
22         (and (pair? x) (eq? (car x) '**))
23 ))
24
25 (define make-sum (lambda (x y)
26         (cond
27                 ((=number? x 0) y)
28                 ((=number? y 0) x)
29                 ((and (number? x) (number? y)) (+ x y))
```

```
30                  (else (list '+ x y))
31          )
32  ))
33
34  (define make-subtract (lambda (x y)
35          (cond
36                  ((=number? x 0) (- y))
37                  ((=number? y 0) x)
38                  ((and (number? x) (number? y)) (- x y))
39                  (else (list '- x y))
40          )
41  ))
42
43  (define make-product (lambda (x y)
44          (cond
45                  ((or (=number? x 0) (=number? y 0)) 0)
46                  ((=number? x 1) y)
47                  ((=number? y 1) x)
48                  ((and (number? x) (number? y)) (* x y))
49                  (else (list '* x y))
50          )
51  ))
52
53  (define make-division (lambda (x y)
54          (cond
55                  ((=number? x 0) 0)
56                  ((=number? y 1) x)
57                  ((and (number? x) (number? y)) (/ x y))
58                  (else (list '/ x y))
59          )
60  ))
61
62  (define make-exponentiation (lambda (b e)
63          (cond
64                  ((=number? e 0) 1)
65                  ((=number? e 1) b)
66                  ((=number? b 1) 1)
67                  ((and (number? b) (number? e)) (** b e))
68                  (else (list '** b e))
69          )
70  ))
71
72  (define addend (lambda (x) (cadr x)))
```

```
73 (define augend (lambda (x) (caddr x)))
74 (define minuend (lambda (x) (cadr x)))
75 (define subtrahend (lambda (x) (caddr x)))
76 (define dividend (lambda (x) (cadr x)))
77 (define divisor (lambda (x) (caddr x)))
78 (define multiplicant (lambda (x) (cadr x)))
79 (define multiplier (lambda (x) (caddr x)))
80 (define base (lambda (x) (cadr x)))
81 (define exponent (lambda (x) (caddr x)))
82
83 (define deriv (lambda (exp var)
84         (cond
85                 ((number? exp) 0)
86                 ((variable? exp)
87                         (if (same-variable? exp var) 1 0)
88                 )
89                 ((sum? exp)
90                         (make-sum
91                                 (deriv (addend exp) var)
92                                 (deriv (augend exp) var)
93                         )
94                 )
95                 ((subtract? exp)
96                         (make-sum
97                                 (deriv (minuend exp) var)
98                                 (make-product -1 (deriv (subtrahend exp) va
99                         )
100                )
101                ((product? exp)
102                        (make-sum
103                                (make-product
104                                        (multiplier exp)
105                                        (deriv (multiplicant exp) var)
106                                )
107                                (make-product
108                                        (deriv (multiplier exp) var)
109                                        (multiplicant exp)
110                                )
111                        )
112                )
113                ((division? exp)
114                        (make-sum
115                                (make-division
```

```
116                                         (make-product
117                                                 (make-product -1 (dividend
118                                                 (deriv (divisor exp) var)
119                                         )
120                                         (make-product
121                                                 (divisor exp)
122                                                 (divisor exp)
123                                         )
124                                 )
125                                 (make-product
126                                         (make-division 1 (divisor exp))
127                                         (deriv (dividend exp) var)
128                                 )
129                         )
130                 )
131                 ((exponentiation? exp)
132                         (make-product
133                                 (make-product
134                                         (exponent exp)
135                                         (deriv (base exp) var)
136                                 )
137                                 (make-exponentiation
138                                         (base exp)
139                                         (make-sum (exponent exp) -1)
140                                 )
141                         )
142                 )
143                 (else
144                         (error "unknown expression type - DERIV" exp )
145                 )
146         )
147 ))
```

2　微分システムの実行例

累乗は (** b e) で表現するとする

```
>(deriv '(+ x 2) 'x)
1

>(deriv '(- 2 x) 'x)
-1
```

```
>(deriv '(* x 2) 'x)
2

>(deriv '(/ x 2) 'x)
0.5

>(deriv '(/ 2 x) 'x)
(/ -2 (* x x))

>(deriv '(** x 4) 'x)
(* 4 (** x 3))

>(deriv '(- (** x y) (* (+ (/ z x) x) x)) 'x)
(+ (* y (** x (+ y -1))) (* -1 (+ (* x (+ (/ (* -1 z) (* x x)) 1)) (+ (/ z x)
x))))
```