MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall 2007

Recitation 11 Solutions
Tagged Data: Symbolic Manipulation

Tagging procedure:

```
(define (tagged-list? x tag)
  (and (pair? x) (eq? (car x) tag)))
```

A tagged abstraction for variables:

```
(define *variable-tag* 'variable)

(define (make-variable vname)
  (list *variable-tag* vname))

(define (variable? x)
  (tagged-list? x *variable-tag*))

(define (varname var)
  (if (variable? var)
      (cadr var)
      (error "not a variable: " var)))

(define (variable=? v1 v2)
  (eq? (varname v1) (varname v2)))
```

Tagged abstraction for constants:

```
(define *constant-tag* 'constant)

(define (make-constant c)
  (list *constant-tag* c))

(define (constant? c)
  (tagged-list? c *constant-tag*))

(define (constval c)
  (if (constant? c)
      (cadr c)
      (error "not a constant: " c)))
```

Tagged abstraction for polynomials:

```
(define *poly-tag* 'poly)

(define (make-poly var terms)
  (list *poly-tag* var terms))

(define (poly? x)
  (tagged-list? x *poly-tag*))

(define (poly-get-var poly)
  (if (poly? poly)
      (cadr poly)
      (error "not a polynomial:" poly)))

(define (poly-get-terms poly)
  (caddr poly))
```

Problems

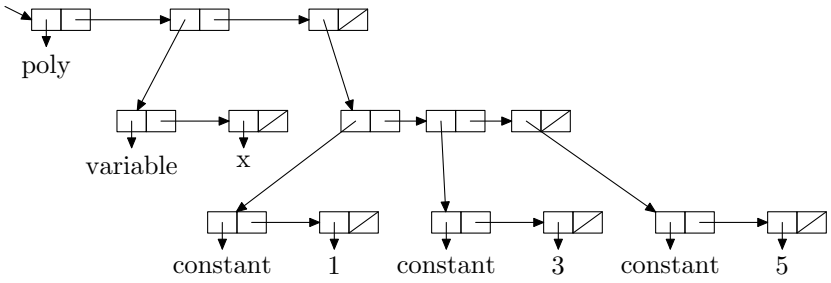
2. Write `constant-add`:

   ```
   (define (constant-add c1 c2)
     (make-constant (+ (constval c1) (constval v2))))
   ```

3. Write a basic `add`, which works only on two constants or two polynomials, assuming you have a procedure `poly-add` which adds two polynomials:

   ```
   (define (add e1 e2)
     (cond ((and (constant? e1)
                 (constant? e2))
            (constant-add e1 e2))
           ((and (poly? e1)
                 (poly? e2))
            (poly-add e1 e2))
           (else (error "not both constants or polys" e1 e2))))
   ```

4. Draw a box-and-pointer diagram of the representation of $5x^2 + 3x + 1$.



5. To actually build `poly-add`, which adds two polynomials:

(a) First write `add-terms`, which takes two lists of terms and returns a new list of sum terms:

```
(define (add-terms t1 t2)
  (cond ((null? t1)
         t2)
        ((null? t2)
         t1)
        (else
         (cons (add (car t1)
                    (car t2))
               (add-terms (cdr t1) (cdr t2))))))
```

(b) Then write `poly-add` using `add-terms`:

```
(define (poly-add p1 p2)
  (if (and (poly? p1) (poly? p2))
      (if (variable=? (poly-get-var p1)
                      (poly-get-var p2))
          (make-poly
           (poly-get-var p1)
           (add-terms (poly-get-terms p1)
                      (poly-get-terms p2)))
          (make-poly
           (poly-get-var p1)
           (cons (add (car (poly-get-terms p1))
                      p2)
                 (cdr (poly-get-terms p1)))))
      (error "not given two polys")))
```

6. What happens (with `add` defined as above), if you try to evaluate the following sequence of expressions:

```
(define x (make-variable 'x))
(define 5x+1 (make-poly x (list (make-constant 1) (make-constant 5))))
(define five (make-constant 5))
(add 5x+1 5x+1)
(add five five)

(add 5x+1 five)
(add x 5x+1)
```

What goes wrong?

All of the add operations only deal with pairs of identical types: two constants or two polynomials. Expressions of mixed types aren't handled

7. Give the following procedures, `var->poly` and `const->poly`, which *promote* variables and constants to polynomials, write a general `->poly` which promotes any of the three types to a polynomial.

```
(define (var->poly var)
  (make-poly var
             (list (make-constant 0)
                   (make-constant 1))))

(define (const->poly var const)
  (make-poly var (list const)))

(define (->poly var exp)
  (cond ((constant? exp)
         (const->poly var exp))
        ((variable? exp)
         (var->poly exp))
        ((poly? exp)
         exp)
        (else
         (error "unknown exp" exp))))
```

8. Write a new version of `add` which uses promotion. Use the following procedure to guess what variable to use when promoting:

```
(define (find-var e1 e2)
  (cond ((poly? e1)
         (poly-get-var e1))
        ((poly? e2)
         (poly-get-var e2))
        ((variable? e1)
         e1)
        ((variable? e2)
         e2)
        (else
         (make-variable 'x))))

(define (add e1 e2)
  (if (and (constant? e1)
           (constant? e2))
      (constant-add e1 e2)
      (let ((var (find-var e1 e2)))
        (poly-add (->poly var e1)
                  (->poly var e2)))))
```