Topic 13
Multiple Representations of Abstract Data – Tagged Data

Section 2.4.2

Complex Numbers: two representations

- Previously we have implemented two different representations for complex numbers – rectangular form and polar form.
- Principle of least commitment – abstraction barriers allow us to decide which concrete representation we want to use up until last minute.
- Take it a step further – want to have both representations at once in a program.

Tagged data

- Allow both representations to be used in the same program
- Issue: how do we interpret a piece of data?
- Given (3 . 4), how do we know which selectors to use in order to interpret this? If it is rectangular representation it means one thing, if it is polar it means another.
- Use a tag to distinguish –

Tags will be "rectangular" and "polar"

Tagged Data

- Need two procedures to select out pieces from tagged data:
- Type-tag – extracts the tag associated with the data
- Contents – extracts the actual data object
- New Procedure: attach-tag takes a tag and contents and produces a tagged data object.

Tagging as a data abstraction

```
; takes a tag and some contents and creates a tagged
; data element (consisting of the tag and the data contents)
(define (attach-tag type-tag contents)
  (cons type-tag contents))

; takes a data item and returns the tag associated with
; the data item.  Assume the data item is tagged as long
; as it is a pair.
(define (type-tag datum)
  (if (pair? datum)
      (car datum)
      (error "TYPE-TAG finds bad datum"
          datum)))
```


(continued)

```
; takes a data item and returns the content part
; of that data item.  Assume the data item is
; tagged as long as it is a pair.
(define (contents datum)
  (if (pair? datum)
      (cdr datum)
      (error "CONTENTS finds bad datum"
          datum)))
```

Predicates for deciding which representation is used:

```
; takes a tagged data item and returns
; #t if the datum is tagged rectangular
(define (rectangular? z)
  (eq? (type-tag z) 'rectangular))

; takes a tagged data item and returns
; #t if the datum is tagged polar
(define (polar? z)
  (eq? (type-tag z) 'polar))
```


Using Tags with Multiple Representations

- Now the two different representations can co-exist in same system
- Need to tag each piece of data as it is made – with rectangular or polar as is specified.
- BE CAREFUL: need to use different constructor and selector names with the two different representations
- Then, implement generic selector which calls the right one on the basis of the tag given
- Note: with these, our original procedures for adding, subtracting etc… still work!

Rectangular Representation Tagged

```
;; lower level implementation of complex numbers
; RECTANGULAR FORM REPRESENTATION

; takes a real and imaginary part and
; creates a complex number represented
; in rectangular form
(define (make-from-real-imag-rectangular x y)
 (attach-tag 'rectangular (cons x y)))

; given an imaginary number in
; rectangular form
; returns the real part
(define (real-part-rectangular z) (car z))

; given an imaginary number in
; rectangular form
; returns the imaginary part
(define (imag-part-rectangular z) (cdr z))
```


Rectangular (cont)

```
; given an imaginary number in
; rectangular form
; return the magnitude
; (using trigonomic rels)
(define (magnitude-rectangular z)
 (sqrt (+ (square (real-part-rectangular z))
          (square (imag-part-rectangular z)))))

; given an imaginary number in
; rectangular form
; return the angle
; (using trigonomic rels)
(define (angle-rectangular z)
 (atan (imag-part-rectangular z) (real-part-rectangular z)))

; takes a magnigude and an angle and
; creates a complex number represented
; in rectangular form
(define (make-from-mag-ang-rectangular r a)
 (attach-tag 'rectangular
        (cons
         (* r (cos a))
         (* r (sin a)))))
```


Polar Representation

```
;; lower level implementation
; POLAR FORM REPRESENTATION

; takes a magnigude and an angle and
; creates a complex number represented
; in polar form
(define (make-from-mag-ang-polar r a)
 (attach-tag 'polar (cons r a)))

; given an imaginary number in
; polar form
; return the magnitude
(define (magnitude-polar z) (car z))

; given an imaginary number in
; rectangular form
; return the angle
(define (angle-polar z) (cdr z))
```


Polar Representation (cont)

```
; given an imaginary number in
; polar form
; returns the real part
; (using trignomic rels)
(define (real-part-polar z)
 (* (magnitude-polar z) (cos (angle-polar z))))

; given an imaginary number in
; polar form
; returns the imaginary part
; (using trigonomic rels)
(define (imag-part-polar z)
 (* (magnitude-polar z) (sin (angle-polar z))))

; takes a real and imaginary part and
; creates a complex number represented
; in polar form (harder)
(define (make-from-real-imag-polar x y)
 (attach-tag 'polar
        (cons
         (sqrt (+ (square x) (square y)))
         (atan y x))))
```

Generic Selectors: real-part

```
; takes a tagged complex number and returns
; the real part
(define (real-part z)
  (cond ((rectangular? z)
          (real-part-rectangular (contents z)))
         ((polar? z)
          (real-part-polar (contents z)))
         (else
           (error
            "data type unknown to REAL-PART“
            z))))
```


Generic Selector: imag-part

```
; takes a tagged complex number and returns
; the imaginary part
(define (imag-part z)
  (cond ((rectangular? z)
          (imag-part-rectangular (contents z)))
         ((polar? z)
          (imag-part-polar (contents z)))
         (else
           (error
            "data-type unknown to IMAG-PART“
            z))))
```


Generic Selector: magnitude

```
; takes a tagged complex number and returns
; the magnitude
(define (magnitude z)
  (cond ((rectangular? z)
          (magnitude-rectangular (contents z)))
         ((polar? z)
          (magnitude-polar (contents z)))
         (else
          (error
             "data-type unknown to MAGNITUDE"
             z))))
```


Generic-Selector: angle

```
; takes a tagged complex number and returns
; the angle
(define (angle z)
  (cond ((rectangular? z)
          (angle-rectangular (contents z)))
         ((polar? z)
          (angle-polar (contents z)))
         (else
           (error
             "data-type unknown to ANGLE"
              z))))
```


Generic Constructors that Tag

```
; takes a real part and an imaginary part and
; creates a complex number -- tags it rectangular
(define (make-from-real-imag x y)
  (make-from-real-imag-rectangular x y))

; takes a magnitude and an angle and creates
; a complex number -- tags it polar
(define (make-from-mag-ang r a)
  (make-from-mag-ang-polar r a))
```


The drama continues . . .

- What will happen if we allowed more representations of the same data type?
- How can we add more representations without rewriting all the implementation procedures all over again?
- How can we make the alternate representations more modular?
- (to be continued)