MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring Semester, 1999

Recitation – Wednesday, March 10

1. Generic Operations

Type Abstraction	Rationals and Integers

Type Abstraction

```
(define (attach-type type contents)
  (cons type contents))

(define (type datum)
  (if (pair? datum)
      (car datum)
      (error "Bad Datum")))

(define (contents datum)
  (if (pair? datum)
      (cdr datum)
      (error "Bad Datum’’)))
```

Rationals and Integers

```
(define (make-rat n d)
  (attach-type ’rat (cons n d)))
(define (numer x) (car x))
(define (denom x) (cdr x))

(define (mul-rat x y)
  (make-rat (MUL (numer x) (numer y))
            (MUL (denom x) (denom y))))

(define (make-int n) (attach-type ’int n))
(define (mul-int n1 n2)
  (make-int (* n1 n2)))
```

Dispatch Method

```
(define (MUL x y)
  (cond ((and (rat? x) (rat? y))
         (mul-rat (contents x) (contents y)))
        ((and (int? x) (int? y))
         (mul-int (contents x) (contents y)))
        (else (error "No method avail."))))
```

Table Method

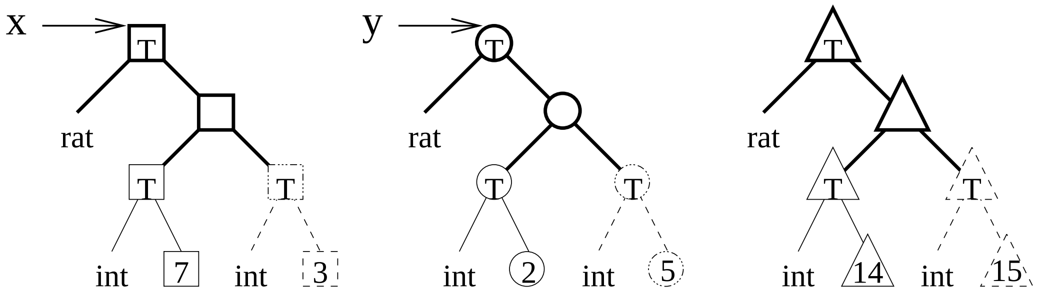
```
(define (apply-generic op . args)
  (let ((type-tags (map type args)))
    (let ((proc (get op type-tags)))
      (if proc
          (apply proc (map contents args))
          (error "No method available")))))

(define (MUL x y) (apply-generic ’mul x y))
(put ’mul ’(rat rat) mul-rat)
(put ’mul ’(int int) mul-int)
```

2. An Example

Let’s walk through the evaluation of `(mul x y)` with the tree structures below. Notice the pattern:

1. Find the correct procedure based on type.
2. Strip off the tag to get to the data.
3. Call the procedure with the data arguments (maybe recursively calling “smart procedures”).
4. Attach the tag to the result.



3. Comments on Generic Operations

1. Adding New Types and Operators

- How would we add a new type (say polynomials)?
  For the dispatch method, each operator must be changed to account for the new type.
  For the table method, we need another put for each operator.
- How would we add a new operator (say divide)?
  For the dispatch method, write a new operator that has the proper dispatch table
  For the table method, we need another put for each type.

2. There are three main ways of dealing with operations on these different types:

- **Dumb:** Implement each box separately, and be sure to use the right procedure.
- **Type Dispatch:** Make "Smart Procedures" from the Columns.
- **Object Oriented:** Make "Smart Data" from the Rows.

	MUL	ADD	SUB
rat	`mul-rat`	`add-rat`	`sub-rat`
int	`mul-int`	`add-int`	`sub-int`
complex	`mul-complex`	`add-complex`	`sub-complex`
poly	`mul-poly`	`add-poly`	`sub-poly`
sets	`set-xsect`	`set-union`	`set-sub`
...	...	...	...

4. Representing Sets

A set is a mathematical object defined as a collection of unique objects (i.e. an element appears at most once in a set). To model sets, we need to build an implementation that supports several operations. Build an implementation for sets of symbols, using unordered lists as the basic representation. Try to use map, filter, and accumulate where appropriate.

- `element?` takes as input an element and a set and returns true if the element is in the set.

```
(define (element? x s)
  (not (null?
    (filter (lambda (elt) (eq? elt x)) s)))
)
```

- `adjoin` takes as input an element and a set and returns a new set with that element added.

```
(define (adjoin x s)
  (if (element? x s)
      s
      (cons x s))
)
```

- `union` takes as input two sets and returns a new set will all elements from both sets.

```
(define (union s t)
  (accumulate adjoin t s)
)
```

- `intersection` takes as input two sets and returns a new set containing any element contained in both of the input sets.

```
(define (intersection s t)
  (filter
     (lambda (x) (element? x t)
     s))
)
```

- `set-` takes as input two sets and returns a new set containing all elements of the first set that are not in the second set

```
(define (set- s t)
  (filter
    (lambda (x) (not (element? x t)
    s)))
)
```

What are the orders of growth of these functions in time and space?

Some of these operations are very expensive? Is there any way that we can do better than this?
Sorted sets using `symbol<`

	Time	Space
`element?`	$\Theta(n)$	$\Theta(n)$
`adjoin`	$\Theta(n)$	$\Theta(n)$
`union`	$\Theta(n^2)$	$\Theta(n)$
`intersection`	$\Theta(n^2)$	$\Theta(n)$
`set-`	$\Theta(n^2)$	$\Theta(n)$