

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
 Department of Electrical Engineering and Computer Science
 6.001—Structure and Interpretation of Computer Programs
 Spring 2006

Recitation 10 Solutions — 3/15/2006
Symbols and Quote

Scheme

1. Special Forms

(a) `quote` - (`quote expr`)
 Returns whatever the reader built for `expr`.

(b) `'thing` - syntactic sugar for (`quote thing`).

2. Procedures

(a) (`eq? v1 v2`) - returns true if `v1` and `v2` are bitwise identical. “Works on” symbols, booleans, and pairs. Doesn’t “work on” numbers and strings.

(b) (`eqv? v1 v2`) - like `eq?` except it “works on” numbers as well.

(c) (`equal? v1 v2`) - return true if `v1` and `v2` print out the same. “Works on” almost everything.

Problems

1. **Evaluation** - give printed value. `x` is 5.

(a) `'3 ==> 3`

(b) `'x ==> x`

(c) `''x ==> (quote x)`

(d) `(quote (3 4)) ==> (3 4)`

(e) `('+ 3 4) ==> error symbol + is not a procedure`

(f) `(if '(= x 0) 7 8) ==> 7`

(g) `(eq? 'x 'X) ==> #f` (Depends on Scheme implementation really)

(h) `(eq? (list 1 2) (list 1 2)) ==> #f`

(i) `(equal? (list 1 2) (list 1 2)) ==> #t`

Sets

A *set* is a collection of unique elements. Attempting to add a second copy of an element to a set will not change the set. We'll be working with sets of symbols.

```
(define (empty-set)
  (cons 'set '()))
```

```
(define (set-elements set)
  (cdr set))
```

2. Write `set-contains?` which returns `#t` if the set contains the element.

```
(define (set-contains? elem set)
  (cond ((null? (cdr set)) #f)
        ((eq? (cadr set) elem) #t)
        (else (set-contains? elem (cons 'set (cddr set))))))
```

3. Write `set-add` which returns a new set which contains includes the new and old elements, but no duplicate elements.

```
(define (set-add elem set)
  (if (not (set-contains? elem set))
      (cons 'set (cons elem (set-elements set)))
      set))
```

Another useful set procedure:

```
(define (set-union set1 set2)
  (fold-right set-add set1 (set-elements set2)))
```

Boolean Formulas

A boolean formula is a formula containing boolean operations and boolean variables. A boolean variable is either `true` or `false`. `and`, `or`, and `not` are all boolean operations. For the purposes of this problem, `and` and `or` will be defined to take exactly two inputs.

Example formulas:

```
a
(not b)
(or b (not c))
(and (not a) (not c))
(not (or (not a) c))
(and (or a (not b)) (or (not a) c))
```

Some useful procedures:

```

(define (variable? exp)
  (symbol? exp))
(define (make-variable var)
  var)
(define (variable-name exp)
  exp)

(define (or? exp)
  (and (pair? exp) (eq? (car exp) 'or)))
(define (make-or exp1 exp2)
  (list 'or exp1 exp2))
(define (or-first exp)
  (cadr exp))
(define (or-second exp)
  (caddr exp))

(define (and? exp)
  (and (pair? exp) (eq? (car exp) 'and)))
(define (make-and exp1 exp2)
  (list 'and exp1 exp2))
(define (and-first exp)
  (cadr exp))
(define (and-second exp)
  (caddr exp))

```

4. Write selectors, constructor, and predicate for not

```

(define (not? exp)
  (and (pair? exp) (eq? (car exp) 'not)))
(define (make-not exp)
  (list 'not exp))
(define (not-operand exp)
  (cadr exp))

```

5. Given a formula, we'd like to be able to tell which variables it involves. `formula-variables` should return the *set* of variables used in the formula.

```

(define (formula-variables exp)
  (cond ((variable? exp)
        (set-add (variable-name exp) (empty-set)))
        ((not? exp)
         (formula-variables (not-operand exp)))
        ((or? exp)
         (set-union (formula-variables (or-first exp))
                    (formula-variables (or-second exp))))
        ((and? exp)
         (set-union (formula-variables (and-first exp))
                    (formula-variables (and-second exp)))))

```

```
(else (error "unknown exp" exp))))
```

6. Given a formula and a list of variable assignments, decide whether the formula is `#t` or `#f`. Assume that you have a procedure `(variable-value bindings vname)`, which takes a list of assignments and a variable name and returns the value assigned to the variable.

```
(define (formula-value bindings exp)
  (cond ((variable? exp)
        (variable-value bindings (variable-name exp)))
        ((not? exp)
         (not (formula-value bindings (not-operand exp))))
        ((or? exp)
         (or (formula-value bindings (or-first exp))
             (formula-value bindings (or-second exp))))
        ((and? exp)
         (and (formula-value bindings (and-first exp))
              (formula-value bindings (and-second exp))))
        (else (error "unknown exp" exp))))
```