METACIRCULAR EVALUATOR 11

COMPUTER SCIENCE 61AS

Metacircular Evaluator

1. How are expressions treated as in Metacircular Evaluator?

   Solution: They are treated as lists.

2. There are many types of procedures, such as special forms. How does MCE figure out the type of the expression?

   Solution: Since the expression is treated as a list, MCE uses tagged-list? procedure to check if the expression is a list with specific tag, such as ('let ...)

3. How are primitive procedures represented? What about compound procedures?

   Solution: A primitive procedure is represented as a tagged list whose cdr is an actual procedure in the underlying Lisp that implements that primitive. For example, procedure + is represented as (primitive #[closure arglist=args ...]). Primitive procedures are applied by the procedure apply-primitive-procedure, which employs underlying lisp. Compound procedures are represented as a list that contains 4 elements: 'compound-procedure, list of procedure parameters, list of procedure body, and the environment where the procedure was created.

Operations on Environment

1. MCE is an extended version of Scheme-1. What is the most significant improvement from Scheme-1?

Solution: In MCE, you can use not only substitution model of evaluation, but also environment model of evaluation. New operations for manipulating environments were added to support the feature.

2. What is a frame? How is it represented?

Solution: A collection of name-value associations or bindings. It is represented as a pair of lists: a list of the variables bound in that frame and a list of the associated values.

3. How does MCE look up a variable?

Solution: In look-up-variable-value, it scans the list of variables in the first frame. If it finds the desired variable, it returns the corresponding element in the list of values. If it does not find the variable in the current frame, it searches the enclosing environment, and so on. If it reaches the empty environment, it signals an ”unbound variable” error.

Practice with MCE

1. Recall that mceval.scm tests true or false using the true? and false? procedure:

```
(define (true? x) (not (eq? x false)))
(define (false? x) (eq? x false))
```

Suppose we type the following definition into MCE:

```
MCE> (define true false)
```

What would be returned by the following expression:

```
MCE> (if (= 2 2) 3 4)
```

____

Solution: 3

2. Suppose we type the following into mc-eval:

```
MCE> (define 'x (* x x))
```

This expression evaluates without error. What would be returned by the following expressions?

```
MCE> quote
```

____

```
MCE> (quote 10)
```

____

Solution: (compound-procedure x ((* x x)) `<procedure-env>`), 10

3. Suppose we just loaded mceval.scm into STk, and erroneously started MCE using (driver-loop) instead of (mce). What is the return value of the following sequence of expressions typed into MCE? If there will be an error, just write ERROR:

```
MCE> 2
```

____

```
MCE> (+ 2 3)
```

____

```
MCE> (define x 10)
```

____

Solution: 2, ERROR, ERROR

4. Rewrite one procedure in the metacircular evaluator so that it will understand infix arithmetic operators. That is, if a compound expression has three subexpressions, of which the second is a procedure but the first isn't, then the procedure should be called with the first and third subexpressions as arguments:

```
> (2 + 3)
5
> (+ 2 3)
5
```

Solution: In the mce-eval add an if clause to the application? clause.

```
(if (procedure? (eval (operator exp))
    ;;Normal eval application
    (mc-apply (mc-eval (car exp) env)
      (list-of-values (cdr exp) env))
    (mc-apply (mc-eval (cadr exp) env)
      (list-of-values (cons (car exp) (cddr exp)) env)))
```

5. Write the new special from `text-multiple`. It takes in as arguments a predicate of one argument and an arbitrary number of arguments. It tests the predicate on each argument in

turn. As soon as one of them returns `#t`, it outputs true. If it has tested all of the arguments and none are true, it outputs false.

```
;;; M-Eval input:
(test-multiple (lambda (x) (equal? 'b x)) 'a 'e 'i 'o 'u)

;;; M-Eval output:
#f

;;; M-Eval input:
(test-multiple (lambda (x) (= x 0)) 3 (/ 1 0) 0)

;;; M-Eval output:
Error

;;; M-Eval input:
(test-multiple (lambda (x) (= x 0)) 0 1 (/ 1 0) 2)


;;; M-Eval output:
#t

;;; M-Eval input
(test-multiple (lambda (x) (= x 0)) 2 3 4)

;;; M-Eval output:
#f
```

Solution: Add to big cond in mc-eval anywhere before `application?` clause:

```
        ((test-multiple? exp) (eval-test-multiple exp env))

    (define (test-multiple? exp)
      (tagged-list? exp 'test-multiple))

    (define (test-multiple-proc exp)
      (cadr exp))

    (define (test-multiple-args exp)
      (cddr exp))
```

```
(define (eval-test-multiple exp env)
  (define (helper proc args env)
    (if (null? args)
    #f
    (if (mc-apply (mc-eval proc env)
                  (list (mc-eval (car args) env)))
        #t
        (helper proc (cdr args) env))))
  (helper (test-multiple-proc exp) (test-multiple-args exp) env))
```