Quiz 11b

**Note: The code for the Meta-Circular Evaluator is at the end of this quiz.**

1. (2+1 points) Louis Reasoner has become fed up of the Metacircular Evaluator exiting every time he types an invalid statement. To fix this problem, he decides to not run the MCE but instead to just call mc-eval directly! He tries out the following lines of code:

> (mc-eval 3 the-global-environment)

3

> (mc-eval (+ 2 3) the-global-environment)

5

> (mc-eval ‘foo the-global-environment)

ERROR: Unbound variable foo

a. (2 points) Why does the last call cause an error even though the first two work?

Because Scheme is applicative order, we must first evaluate all arguments to mc-eval before calling mc-eval. In the last call, this means the arguments to mc-eval are foo and a long list that represents the-global-environment. At this point, foo is not quoted, which means mc-eval treats it as a variable, but it does not have a value in the-global-environment, causing the error.

Rubric:

2 points - completely correct

1 point - on the right track but vague, or slightly incorrect

0 point - completely incorrect, shows little understanding

b. (1 point) Make a change to the the last call so that it works as expected (returns foo). Make as small of a change as possible to fix the problem.

To solve this problem, we should instead call:

(mc-eval “foo the-global-environment)

Additionally, if we actually want mc-eval to do the work we expect it to, we should change the previous two calls to also be ‘3 and ‘(+ 2 3).

Rubric:

1 Point - Above solution

0.5 points - Properly extending the-global-environment with foo

0 points - Too complicated or wrong

1. (4 points) Write the new special form test-multiple. It takes 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 is true, then it outputs false. Show all changes you will have to make to the MCE. **We may take points off for bad style.**

;;; 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

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))

Rubric:

4 points - completely correct

1 - 3 points - up to reader’s discretion.

Take 1 point off for lack of data abstraction.

0 points - completely wrong

1. (3 points) In Scheme, we create a new frame whenever we invoke a compound procedure. This means variables created while evaluating the body of the procedure are in this new frame. Some languages make if-statements create a new frame. Make changes to the Metacircular Evaluator such that if-statements have their own frame. Show all necessary changes below. Here is an example of how your new code should work:

;;; M-Eval input:

(define x 0)

;;; M-Eval output:

okay

;;; M-Eval input:

(if #t (define x 3) ‘do-nothing)

;;; M-Eval output:

okay

;;; M-Eval input:

x

;;; M-Eval output:

0 ;;; The original version would have returned 3

Changes are in CAPS:

(define (eval-if exp env)

(LET ((NEW-ENV (EXTEND-ENVIRONMENT ‘() ‘() ENV)))

(if (true? (mc-eval (if-predicate exp) NEW-ENV))

(mc-eval (if-consequent exp) NEW-ENV)

(mc-eval (if-alternative exp) NEW-ENV))))

Rubric:

Overly complicated solutions can get at most 2 points.

Rest of points up to reader’s discretion.

(define (mc-eval exp env)

(cond ((self-evaluating? exp) exp)

((variable? exp) (lookup-variable-value exp env))

((quoted? exp) (text-of-quotation exp))

((assignment? exp) (eval-assignment exp env))

((definition? exp) (eval-definition exp env))

((if? exp) (eval-if exp env))

((lambda? exp)

(make-procedure (lambda-parameters exp)

(lambda-body exp)

env))

((begin? exp)

(eval-sequence (begin-actions exp) env))

((cond? exp) (mc-eval (cond->if exp) env))

((application? exp)

(mc-apply (mc-eval (operator exp) env)

(list-of-values (operands exp) env)))

(else (error "Unknown expression type -- EVAL" exp))))

(define (mc-apply procedure arguments)

(cond ((primitive-procedure? procedure)

(apply-primitive-procedure procedure arguments))

((compound-procedure? procedure)

(eval-sequence

(procedure-body procedure)

(extend-environment

(procedure-parameters procedure)

arguments

(procedure-environment procedure))))

(else

(error "Unknown procedure type -- APPLY" procedure))))

(define (make-procedure parameters body env)

(list 'procedure parameters body env))

(define (compound-procedure? p) (tagged-list? p 'procedure))

(define (primitive-procedure? proc) (tagged-list? proc 'primitive))

(define (primitive-implementation proc) (cadr proc))

(define (primitive-procedure-objects)

(map (lambda (proc) (list 'primitive (cadr proc)))

primitive-procedures))

(define (make-frame variables values) (cons variables values))

(define (add-binding-to-frame! var val frame)

(set-car! frame (cons var (car frame)))

(set-cdr! frame (cons val (cdr frame))))

(define (lookup-variable-value var env)

(define (env-loop env)

(define (scan vars vals)

(cond ((null? vars)

(env-loop (enclosing-environment env)))

((eq? var (car vars))

(car vals))

(else (scan (cdr vars) (cdr vals)))))

(if (eq? env the-empty-environment)

(error "Unbound variable" var)

(let ((frame (first-frame env)))

(scan (frame-variables frame)

(frame-values frame)))))

(env-loop env))

(define (set-variable-value! var val env)

(define (env-loop env)

(define (scan vars vals)

(cond ((null? vars)

(env-loop (enclosing-environment env)))

((eq? var (car vars))

(set-car! vals val))

(else (scan (cdr vars) (cdr vals)))))

(if (eq? env the-empty-environment)

(error "Unbound variable -- SET!" var)

(let ((frame (first-frame env)))

(scan (frame-variables frame)

(frame-values frame)))))

(env-loop env))

(define (define-variable! var val env)

(let ((frame (first-frame env)))

(define (scan vars vals)

(cond ((null? vars)

(add-binding-to-frame! var val frame))

((eq? var (car vars))

(set-car! vals val))

(else (scan (cdr vars) (cdr vals)))))

(scan (frame-variables frame)

(frame-values frame))))