MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring Semester, 1998

Recitation – Friday, April 10

1. Student Presentation

Now that we can look at the implementation of the metacircular evaluator we might expect that finally we can figure out in what order it evaluates the sub-expressions in a combination. The procedure that evaluates those expressions is named list-of-values and is found on page 367 of Section 4.1 in the textbook. Unfortunately, it turns out that our metacircular evaluator inherits the order of evaluation that the underlying Scheme interpreter happens to use. Explain this observation to the class, and then show how to modify list-of-values so that it always evaluates expressions left to right.

2. The Meta-Circular Evaluator

The heart of the Scheme language is an interpreter, that is, a process that evaluates legal expressions of the language. In lecture, we saw a Scheme interpreter built around the use of two basic procedures, which we will call `mc-eval` and `mc-apply`. Examples of these, without data abstractions, are given below: These are given below:

```
(define mc-eval
  (lambda (exp env)
    (cond ((number? exp) exp)                   ;base case for numbers
          ((symbol? exp) (lookup exp env))      ;base case for variables
          ((eq? (car exp) 'quote) (car (cdr exp)))      ;special forms
          ((eq? (car exp) 'lambda)
            (list 'procedure (cadr exp) (caddr exp) env))
          ((eq? (car exp) 'define) (evdefine (cdr exp) env))
          (else (mc-apply (mc-eval (car exp) env)       ;procedure application
                          (list-of-values (cdr exp) env))))))

(define mc-apply
  (lambda (proc args)
     (cond ((atom? proc) (apply proc args))       ;primitive case
           ((eq? (car proc) 'procedure)           ;compound procedure
            (mc-eval (caddr proc)                         ;procedure body
                     (extend-environment (cadr proc)      ;formal parameters
                                         args             ;supplied arguments
                                         (cadddr proc)))) ;saved environment
           (else (error "Unknown procedure")))))

(define list-of-values
```

3. An Example using the Meta-Circular Evaluator

```
(mc-eval '(define z (+ 1 3)) the-global-environment)
(mc-eval '(define mult (lambda (x y) (* x y))) the-global-environment)
(mc-eval '(mult z 3) the-global-environment)
```

MC-Eval		MC-Apply	
Expression	Env	Procedure	Args
(define z (+ 1 3))	GE		
(+ 1 3)	GE		
+ ⇒ #[Prim +]	GE		
1 ⇒ 1	GE		
3 ⇒ 3	GE		
		#[Prim +] ⇒ 4	(1 3)
(define mult (lambda (x y) (* x y)))	GE		
(lambda (x y) (* x y)) ⇒ (procedure (x y) (* x y) GE)	GE		
(mult z 3)	GE		
mult ⇒ (procedure (x y) (* x y) GE)	GE		
z ⇒ 4	GE		
3 ⇒ 3	GE		
		(procedure (x y) (* x y) GE)	(4 3)
(* x y)	E1		
⇒ #[Prim *]	E1		
x ⇒ 4	E1		
y ⇒ 3	E1		
		#[Prim *] ⇒ 12	(4 3)

4. Adding let to the Meta-Circular Evaluator