CS61A Notes – Week 08a Solutions: Analyzing Evaluator

Analyzing Evaluator – "This is where the magic happens"

Which of the following would have speed up in `analyzing-evaluator`?

Remember, the non-primitive procedure must be **called more than once** for there to be speedup!

1. **`(+ 1 2)`**
   no, primitive

2. **`(((lambda (x) (lambda (y) (+ x y))) 5) 6)`**
   no, both lambdas only called once

3. **`(map (lambda (x) (* x x)) '(1 2 3 4 5 6 7 8 9 10))`**
   yes, since we're using the same lambda function for all 10 numbers

4. **`(define fib`**
   `  (lambda (n)`
   `    (if (or (= n 0) (= n 1)) 1`
   `        (+ (fib (- n 1)) (fib (- n 2)))))`
   `(fib 5)`
   yes, fib is called several times recursively

5. **`(define fact`**
   `  (lambda (x) (if (= x 0) 1 (* x (fact (- x 1))))))`
   no, since we never call fact! (we only wrote the define :])

6. **`(accumulate cons nil '(1 2 3 4 5 6 7 8 9 10))`**
   no, since cons is primitive, and is therefore already a Scheme procedure

---

FINAL REVIEW QUESTIONS

QUESTION 1.

```
(assert! (rule (less () (a . ?y))))
(assert! (rule (less (a . ?x) (a . ?y))
               (less ?x ?y)))
```

QUESTION 2.

Hello will be printed 9 times. The return values are 2 and then 3.

QUESTION 3.

(a) Incorrect result.
(b) Neither.
(c) Neither.
(d) Neither.
(e) Deadlock.

QUESTION 4.

```
    (define (subseq l1 l2)
       (cond ((null? l1) #t)
             ((null? l2) #f)
             ((equal? (car l1) (car l2)
              (subseq (cdr l1) (cdr l2))))
             ((not (equal? (car l1) (car l2)))
              (subseq l1 (cdr l2)))))
```

```
(assert! (rule (same ?x ?x)))

(assert! (rule (subseq () ?l2)))

(assert! (rule (subseq (?carl1 . ?cdrl1) (?carl1 . ?cdrl2))
               (subseq ?cdrl1 ?cdrl2 ?cdrl2)))

(assert! (rule (subseq (?carl1 . ?cdrl1) (?carl2 . ?cdrl2))
               (and (subseq (?carl1 . ?cdrl1) ?cdrl2)
                    (not (same ?carl1 ?carl2)))))
```

QUESTION 5.
Lem E. Tweakit makes the following change to MC-EVAL:

```
        ((application? exp)
         (mc-apply (mc-eval (operator exp) env)
                   (list (eval-sequence (operands exp) env))))
```

This clause in MC-EVAL is responsible for evaluating the subexpressions in a combination and then calling MC-APPLY. Lem has replaced LIST-OF-VALUES, which takes a list of expressions and produces a list of expression values, with a call to EVAL-SEQUENCE. EVAL-SEQUENCE is ordinarily used only to evaluate the bodies of procedures and BEGINs; it evaluates each expression in sequence, then returns the value of the *last* one. Lem takes that single result and puts it in a list of one element. This approach works fine if the procedure is being called with only one argument, but for procedures of multiple arguments, all but the last argument value will be lost.

```
        (square 1)
```

The result is correct, but for an accidental reason. Clearly the call to SQUARE works fine, since there is only one argument. But in the call to *, the second argument is ignored. However, this doesn't pose a problem, because the value of (* 1 1) is the same as the value of (* 1).

```
        (square 2)
```

This one produces an incorrect result. (* 2 2) does not produce the same value as (* 2).

```
        (+ 3 4)
```

This is incorrect, too. The modified evaluator tries to evaluate (+ 4), which gives 4, not 7.

```
        (lambda (x y) y)
```

This works fine. LAMBDA is a special form that creates procedures. The APPLICATION? clause in MC-EVAL is only used for ordinary procedures, not special forms, so the arguments to the LAMBDA aren't mangled.

```
        ((lambda (x y) y) 6 7)
```

This produces an error. The argument 6 is dropped by Lem's evaluator, which then attempts to call a procedure of two arguments with only one argument: ((lambda (x y) y) 7).

```
        ((lambda (x) (+ x 5)) 9)
```

The result of this expression is incorrect in Lem's evaluator. The call to the LAMBDA-created procedure works fine, since there is only one argument, but the first argument to + is dropped. The result is 5, even though we expect it to be 9.