LAZY EVALUATOR AND ANALYZING EVALUATOR
12

COMPUTER SCIENCE 61AS

Warm-up

1. What are applicative order and normal order? Which one is used in Lazy Evaluator?

2. What is a thunk? What are the two procedures that are used to create and evaluate a thunk?

Lazy Evaluator

1. There are two versions of the lazy evaluator – one in which thunks are memoized, and the other without memoization (memoization of thunks is analogous to the memoization of streams in lab 11). Which procedures need to be different in the memoizing and the non-memoizing versions of the lazy evaluator?

2. When should an expression be delayed in the lazy evaluator? (In other words, when should delay-it be called?) Hint: What's the difference between applicative and normal order?)

3. What should an expression be forced in the lazy evaluator, and why?

Lazy Evaluator VS Metacircular Evaluator

1. For each of the following, say whether lazy evaluator would evaluate and print an answer faster than the regular metacircular evaluator.

   (a)
   ```
   (define x 4)
   (define (foo a b)
           (set! x (+ a 2))
           b)
   (foo (* 6 9) (* 8 3))
   ```

   Faster Not faster

   (b)
   ```
   ((lambda (x y) (+ x y)) (* 4 4) (* 9 9))
   ```

   Faster Not faster

   (c)
   ```
   ((lambda (x y) (* 2 x)) (* 4 4) (* 9 9))
   ```

   Faster Not faster

2. Here's the way we represent (non-memoized) thunks:

```
(define (delay-it exp env)
        (list 'thunk exp env))

(define (actual-value exp env)
        (force-it (mc-eval exp env)))

(define (force-it obj)
        (if (thunk? obj)
            (actual-value (thunk-exp obj) (thunk-env obj))
            obj))

(define (thunk? obj)
        (tagged-list? obj 'thunk))
```

What would be the result of the following pieces of code in the metacircular evaluator, and in the lazy evaluator? Assume second is a primitive procedure that returns its second argument.

(a)
```
(define (f x) 42)
(f (/ 4 0))
```

MCE:_____ Lazy evaluator:_____

(b)
```
(list 'thunk 1 2)
```

MCE:_____ Lazy evaluator:_____

(c)
```
(list 'thunk '(+ 2 3) ())
```

MCE:_____ Lazy evaluator:_____

(d)
```
(define x 10)
(begin (second (set! x (- x 1)) (set! x (- x 1)))
       x)
```

MCE:_____ Lazy evaluator:_____

(e)
```
(define (my-second x y) y)
(define x 10)
(begin (my-second (set! x (- x 1)) (set! x (- x 1)))
       x)
```

MCE:_____ Lazy evaluator:_____

(f)
```
(define x 10)
(begin (second x
               (my-second (set! x (- x 1))
                          (set! x (- x 1))))
       x)

MCE:_____ Lazy evaluator:_____
```

Analyzing Evaluator

1. What is the benefit of analyzing evaluator and how is it achieved?

2. What does the procedure analyze take as an argument? What does it return?

3. Which expressions will be evaluated faster in Analyzing Evaluator?

```
a. 5
b. (* 2 3)
c. (map (lambda (x) (* x 3)) (list 1))
d. (define (factorial n)
     (if (= n 1)
         1
         (* (factorial (- n 1)) n)))
   (factorial 4)
e. (map (lambda (x) (+ x 100)) (list 1 2 3 4 5 6 7 8 9 10 11))
f. (accumulate cons nil (list 'hi 'nice 'to 'meet 'you))
```