CSCI-GA-2110– Problem Set 4– Written Part

1 Passing self with macros

Task 1.1

- Consider the following program in racket

```
#lang racket

(define vec (vector 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20))

(define val (box 0))

(define (msg-self o m . a)
  (apply (o m) o a))

(define-syntax (msg/self stx)
  (syntax-case stx ()
    [(msg/self o m a ...)
     #' ((o m) o a ...)]))

(define obj-1
  (lambda (m)
    (case m
      [(up) (lambda (self) (set-box! val (+ (unbox val) 1)))]
      [(down) (lambda (self) (set-box! val (- (unbox val) 1)))]
      [(up1) (lambda (self)
                    (begin
                       (msg-self self 'up)
                       (msg-self self 'up)
                       (msg-self self 'down)
                       (vector-ref vec (unbox val))))])))

(set-box! val 0)
(displayln "using msg-self (which uses apply)")
(displayln (msg-self (begin (set-box! val (+ (unbox val) 4)) (displayln
"val set to 4") obj-1) 'up1))
(display "val: ")
(displayln (unbox val))
(displayln "----------------------------------")

(define obj-2
  (lambda (m)
    (case m
      [(up) (lambda (self) (set-box! val (+ (unbox val) 1)))]
      [(down) (lambda (self) (set-box! val (- (unbox val) 1)))]
      [(up1) (lambda (self)
                    (begin
                       (msg/self self 'up)
```

```
                        (msg/self self 'up)
                        (msg/self self 'down)
                        (vector-ref vec (unbox val))))]))
  )

(set-box! val 0)
(displayln "using msg/self (which uses the macro)")
(displayln (msg/self (begin (set-box! val (+ (unbox val) 4)) (displayln
"val set to 4") obj-2) 'up1))
(display "val: ")
(displayln (unbox val))
```

- The program outputs the following result:

```
Welcome to DrRacket, version 8.14 [cs].
Language: racket, with debugging; memory limit: 256 MB.
using msg-self (which uses apply)
val set to 4
6
val: 5
-----------------------------------
using msg/self (which uses the macro)
val set to 4
val set to 4
10
val: 9
>
```

- The reason why the macro implementation gives an incorrect output is because the macro's expansion duplicates the expression `o`, causing side-effects (like modifying val) to happen more than once. This leads to erroneous program behavior

- Here, in the macro version, val becomes 8 before entering the up1 method due to the double evaluation. We can see that the double evaluation is happening by noting the fact that the statement `"val set to 4"` also occurs 2 times in the output.

2 Continuations

Task 2.1

1. `(+ (+ 3 5) (+ (let/cc k 4) (+ 6 11)))`

- Ans. `(+ (+ 3 5) (+ [...] (+ 6 11)))`

2. `(begin (begin (set-box! b 1) 1) (let/cc k (unbox b)) (set-box! b 2))`

- Ans. `(begin [...] (set-box! b 2))`

3. `(map (begin (begin (set-box! b 1) #f) (lambda (x) (+ (unbox b) x))) (list (let/cc k 1) 2 3 (+ 2 2)))`

- Ans. `(map (begin (begin (set-box! b 1) #f) (lambda (x) (+ (unbox b) x))) (list [...] 2 3 (+ 2 2)))`