BBM301 Programming Languages
Tail recursion and Iterative functions in Scheme

İbrahim Burak Tanrıkulu, 21827852

January 10, 2021

PART A: Converting recursive functions to tail recursive ones

1- Finding length of a list with tail recursion:

step 0: (length '(1, 2, 3, 4, 5))
step 1: (length_helper '(1, 2, 3, 4, 5) 0)
step 2: (length_helper '(2, 3, 4, 5) (+ 1 0))
step 3: (length_helper '(3, 4, 5) (+ 1 1))
step 4: (length_helper '(4, 5) (+ 1 2)
step 5: (length_helper '(5) (+ 1 3))
step 6: (length_helper '() (+ 1 4))
step 7: lst is null, return 5.

We just store current_length variable and increase it at each stage. We don't store return address and parameters of each recursive function. Thus we gained time and space.

2- sum-of-squares tail recursion:

a)

```
(define (sum-of-squares n)
(letrec (
        (sum-of-squares-helper (lambda (n sum)
                    (if (= n 0)
                      sum
                      (sum-of-squares-helper (- n 1) (+ sum (* n n)))
                    ))
        ))
(sum-of-squares-helper n 0)
))
```

b)

In this part, i will use "SoS" abbreviation for sum-of-squares and SoSh for sum-of-squares-helper.

```
         recursive              tail recursive
step 0:   (SoS 5)               (SoS 5)
step 1:  -(SoS 4)               (SoSh 5 0)
step 2:  --(SoS 3)              (SoSh 4 25)
step 3:  ---(SoS 2)             (SoSh 3 41)
step 4:  ----(SoS 1)            (SoSh 2 50)
step 5:  -----(SoS 0)           (SoSh 1 54)
step 6:  -----0                 (SoSh 0 55)
step 7:  ----1                  55
step 8:  ---5
step 9:  --14
step 10: -30
step 11: 55
```

As you can see, recursive is slower than tail recursive. Also, recursive uses more stack space.

3- sum-of-factorials-of-elements:

a)

```
(define sum-of-factorials-of-elements
    (lambda (lst)
      (if (null? lst)
        0
        (+ (factorial (car lst)) (sum-of-factorials-of-elements (cdr lst))))))
```

b)

```
(define (sum-of-factorials-of-elements lst)
(letrec(
       (sofoeh (lambda (lst sum)
               (if (null? lst)
                   sum
                  (sofoeh (cdr lst) (+ sum (factorial (car lst))))
       ))
  ))
  (sofoeh lst 0)
))
```

c)

In this part, i will use "sofoe" abbreviation for sum-of-factorials-of-elements.

```
          recursive steps
step 0:  (sofoe '(3 2 5 1 4))
step 1:  (+ 3! (sofoe '(2 5 1 4)))
step 2:  (+ 3! (+ 2! (sofoe '(5 1 4))))
step 3:  (+ 3! (+ 2! (+ 5! (sofoe '(1 4)))))
step 4:  (+ 3! (+ 2! (+ 5! (+ 1! (sofoe '(4))))))
step 5:  (+ 3! (+ 2! (+ 5! (+ 1! (+ 4! (sofoe '())))))))
step 6:  (+ 3! (+ 2! (+ 5! (+ 1! (+ 4! 0)))))
step 11:     153
                    tail recursion steps
step 0: (sofoe '(3 2 5 1 4))
step 1: (sofoeh '(3 2 5 1 4) 0)
step 2: (sofoeh '(2 5 1 4) 6)
step 3: (sofoeh '(5 1 4) 8)
step 4: (sofoeh '(1 4) 128)
step 5: (sofoeh '(4) 129)
step 6: (sofoeh '() 153)
step 7: 153
```

PART B: Writing iterative functions

1- sum-of-squares iterative:

```
(define (sum-of-squares n)
(do ( (i 1 (+ i 1)) (sum-of-squares 0) )
    ((> i n)
        sum-of-squares)
        (set! sum-of-squares (+ sum-of-squares (* i i)))))
```

2- sum-of-factorials-of-elements:

```
(define (sofoe lst)
(do ( (mylist lst (cdr mylist)) (sofoe 0))
    ((null? mylist)
        sofoe)
        (set! sofoe (+ sofoe (factorial (car mylist))))))
```

Comparing sum-of-squares

I used (time (sum-of-squares 30000000)) command for getting time.

	Recursive	Tail Recursive	Iterative
Storage	Return addresses, arguments for each call	Extra return variable	Extra loop variable
Time	10.7	3.4	4.5

I didn't used any reference. All these are my own code and comments. I used repl.it and tutorialspoint sites' scheme interpreter.