HACETTEPE UNIVERSITY

DEPARTMENT OF
COMPUTER ENGINEERING

BBM301 Programming Languages
Assignment 2
Tail Recursion in Scheme

Author
Kayla AKYÜZ
21726914
b21726914@cs.hacettepe.edu.tr

Advisors
Prof. Pinar DUYGULU SAHIN
pinar@cs.hacettepe.edu.tr
A. Prof. Nazli IKIZLER-CINBIS
nazli@cs.hacettepe.edu.tr

January 10,2021

Part A

Task 1

At the beginning the code will launch with input parameters (list 1 2 3 4 5). After evaluating letrec function, length_helper name will be assigned to it's init. This is kinda letting us define a function within another function. After that (length_helper lst 0) will be called which will insert us into the tail-recursive function.

```
(+ 1 0) => 1 ; At the first step of tail-recursive function length_
   helper will check if the list is null and since it is not null one
   will be added to current_length and one element will be removed
   form list and they will be passed down to recursion. We can observe
   everything already gets calculated and current recursion step is
   over and done with.
(+ 1 1) => 2 ; Recursion will continue until the list is null.
(+ 1 2) => 3 ; Every step we increase length.
(+ 1 3) => 4 ; So calculations are done as we move down the recursion.
(+ 1 4) => 5 ; At this step a null list will be passed down the
   recursion.
(5) => 5 ; Finally list will be checked null so only the
   current_length will be executed which will return it.
```

Task 2

a)

```scheme
(define (sum-of-squares n)
(letrec (
        (oto-summer (lambda (current_sum n)
                (if (= n 0)
                        current_sum
                        (oto-summer (+ current_sum (* n n)) (- n 1))
                ))
        ))
(oto-summer 0 n)
))
```

b)

Original:

```scheme
(+ (+ (+ (+ (+ 0 (* 1 1)) (* 2 2)) (* 3 3))  (* 4 4)) (* 5 5)) => 55 ;
   We can observe even tho first element can be calculated the
   function awaits all the steps to be calculated.
```

Tail-recursive:

```scheme
(+ 0 (* 5 5)) => 25 ; At the beginning recursion will start with sum
   of 0.
(+ 25 (* 4 4)) => 41 ; After each step sum will be increased.
(+ 41 (* 3 3)) => 50 ; I think with this way of calculating even if we
   abort program at some point,
(+ 50 (* 2 2)) => 54 ; since sum is incremented already we will get
   partial result.
(+ 54 (* 1 1)) => 55 ; At this point n will be passed as 0.
(55) => 55 ; So the function will return current_sum.
```

Task 3

a)

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

b)

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

c)

Original:

```
(+ 6 (+ 2 (+ 120 (+ 1 24)))) => 153
```

Tail-recursive:

```
(+ 0 6) => 6
(+ 6 2) => 8
(+ 8 120) => 128
(+ 128 1) => 129
(+ 129 24) => 153
(153) => 153
```

Part B

Task 4

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

Task 5

```
(define (sum-of-factorials-of-elements lst)
(do (( (set! lst (cdr lst)))
(sum 0))
((null? lst) sum)
(set! sum (+ sum (factorial (car lst))))
))
```

Task 6

As we understand from this assignment compilers compile tail-recursive functions as if they are iterative so they have similar memory management. However even iterative functions are usually faster tail-recursive functions are more readable and immutable. Normal recursive functions on the other hand require huge amounts of memory allocation and because of all the memory management they are way more slower. It is clear that recursive functions are the worst and tail-recursive functions are the best, but not the fastest.

REFERENCES

LaTex Tutorials
Scheme listings in LaTeX
Defining Scheme Functions
Scheme's Built-in Procedures
Equality Checking in Scheme
Iterative vs Recursive vs Tail-Recursive in Golang
Tail Recursive vs Iterative Algorithms
Memory Usage Compare Between a Tail Recursive and Iterative Implementation
Tail Recursion and Loops
Scheme Expression
Binding Constructs in Scheme