## TAIL RECURSION AND INTERPRETERS

## Computer Science Mentors 61A

April 9 to April 11, 2018

**1** Tail Recursion

- 1. What is a tail context? What is a tail call? What is a tail recursive function?
- 2. Why are tail calls useful for recursive functions?
- 3. Consider the following function:

Why is count-instance not a tail call? Optional: draw out the environment diagram of this sum-list with lst (1 2 1) with x = 1.

4. Rewrite count-instance in a tail recursive context.
 (define (count-tail lst x)

)

)

5. Implement filter, which takes in a one-argument function f and a list lst, and returns a new list containing only the elements in lst for which f returns true. Your function must be tail recursive.

You may wish to use the built-in append function, which takes in two lists and returns a new list containing the elements of the first list followed by the elements of the second.

(**define** (filter f lst)

## 2 Interpreters

1. Circle the number of calls to scheme\_eval and scheme\_apply for the code below.
 (+ 1 2)

scheme\_eval 1 3 4 6 scheme\_apply 1 2 3 4

2. Circle the number of calls to scheme\_eval and scheme\_apply for the code below. (if 1 (+ 2 3) (/ 1 0))

```
(or #f (and (+ 1 2) 'apple) (- 5 2))
scheme_eval 6 8 9 10
scheme_apply 1 2 3 4
```

(**define** (square x) (\* x x))

```
(+ (square 3) (- 3 2))
scheme_eval 2 5 14 24
scheme_apply 1 2 3 4
```

(**define** (add x y) (+ x y)) (add (- 5 3) (**or** 0 2))

```
scheme_eval 12 13 14 15
scheme_apply 1 2 3 4
```