CS 61A
Discussion 9

Tail Calls and Interpreters

Raymond Chan
Discussion 121
UC Berkeley

Agenda

- Announcements
- Tail Calls
- Interpreters

Announcements

- Homework 6 due Friday (4/8) (Homework party tonight)
- Quiz 2 due today (check course website)
- Lab 10 due Friday
- Scheme Project due 4/23 (Start early)
- Submit Midterm 2 Regrade Requests on Gradescope
- Maps Composition Revision due 4/15

Midterm 2

- Video Walkthrough
- https://youtu.be/0TkYl4jnFlk

Tail Calls

- Tail-optimization in scheme allow recursive functions that take constant space.
- A **Tail Call** occurs if the function call is the **last operation** of the **current frame**.
- With no operations after the function call, we don't need to lookup variables anymore in the current frame.
- Can use the current frame as the function's new call frame.
- Can be a recursive call or a call to another function.

Tail Calls

- Factorial example; Not tail recursive

```
(define (fact n)
    (if (= n 0) 1
        (* n  (fact (- n 1)))))
```

Tail Calls

- After **(fact (- n 1))** returns, we multiply the return value by **n**.
- We need to remember **n** in each frame.

```
(define (fact n)
    (if (= n 0) 1
        (* n  (fact (- n 1)))))
```

Tail Calls

- Tail Recursive .

```
(define (fact n)
    (define (fact-tail n result)
        (if (= n 0) result
            (fact-tail (- n 1) (* n result))))
    (fact-tail n 1))
```

Tail Calls

- After each call to fact-tail, there are no more operations.
- We do not need the current frame’s variables anymore.

```
(define (fact n)
    (define (fact-tail n result)
        (if (= n 0) result
            (fact-tail (- n 1) (* n result))))
    (fact-tail n 1))
```

Tail Calls

- **result** is the list that we are building in each frame.
- At the end, we can just return **result**.

```
(define (fact n)
    (define (fact-tail n result)
        (if (= n 0) result
            (fact-tail (- n 1) (* n result))))
    (fact-tail n 1))
```

Tail Calls

- We keep track of **n** and **result** by passing them on as arguments to the recursive call.
- Use helper functions!

```
(define (fact n)
    (define (fact-tail n result)
        (if (= n 0) result
            (fact-tail (- n 1) (* n result))))
    (fact-tail n 1))
```

Tail Calls

- Closest thing to iteration in Scheme.

```
(define (fact n)
    (define (fact-tail n result)
        (if (= n 0) result
            (fact-tail (- n 1) (* n result))))
    (fact-tail n 1))
```

Tail Calls

- A function call is a tail call if it is in a **tail context**. This function may or may not be tail-recursive.

- A tail-recursive function requires the recursive call to be the last action, which implies it is in a tail context.

Tail Context

- Last sub-expr in the body of a **lambda**.
- Second or Third sub-expr in an **if** form (values that return).
- Any non-predicate sub-expr in a **cond** form.
- Last sub-expr in an **and** or an **or** form.
- Last sub-expr in a **begin**'s body.

Interpreters

- Programs that understand other programs.

- Use an **underlying language** to implement an interpreter that can understand the **implemented language**.

- Read-Eval-Print-Loop (REPL).

Interpreters - REPL

- Read
  - Lever turns input into “tokens.”
  - Parser organizes “tokens” into data structures of the underlying language.
    - We use Pairs, which is a form of Linked List.

Interpreters - REPL

- Eval
    - Mutual recursion between eval and apply.
    - Eval: evaluates an expression according to the rules of the language.
        - Deals with **expressions**.
- Apply: applies the function to the argument values.
    - May call eval to evaluate sub-expressions.
    - Deals with Values

Interpreters - REPL

- Evaluation
  - Primitive expressions evaluated directly.
  - Call expressions
    - **Evaluate** operator.
    - **Evaluate** operands from left to right.
    - **Apply** operator to the operands.

Interpreters - REPL

- Print displays result.

Recap

- Tail calls allow us to use constant space in the number of frames.
- Tail calls require the last action to be a function call.
- Interpreters
- Read-Eval-Print-Loop