An empirical perspective on why we perform a formal analysis of algorithms

By leveraging the property of code as data in the meta-circular evaluator, as shown in Appendix A, it is not difficult to compare the speed of original meta-circular evaluator with the syntax-analyzing version, as Exercise 4.24 asks.

By counting the calls to each conditional case within the evaluators, we can compare the speed of two evaluators by running fibonacci function with an argument value of 15:

—Number of calls required by each conditional case within the original meta-circular evaluator when running expression "(fib 15)":

```
(mc-eval . 26638)
(application? . 6905)
(definition? . 2)
(self-evaluating? . 3946)
(variable? . 11837)
(if? . 1973)
(lambda? . 2)
(define-variable! . 5)
(cond? . 1973)
(compound-application/eval-sequence . 1974)
(lookup-variable-value . 11837)
(extend-environment . 1975)
(primitive-application . 4931)
```

—Number of calls required by each conditional case within the analyzing syntax version of evaluator when running expression "(fib 15)":

```
(analyze-total . 28)
(analyze-application? . 8)
(analyze-definition? . 1)
(analyze-self-evaluating? . 3)
(analyze-variable? . 12)
(analyze-if? . 1)
(analyze-lambda? . 2)
(define-variable! . 5)
(analyze-cond? . 1)
(execute-application-compound . 1974)
(lookup-variable-value . 11837)
(extend-environment . 1975)
(execute-application-primitive . 4931)
```

We see that we can save a lot of calls to the evaluator by analyzing the syntax (purple). On the other hand, the number of calls for manipulating the environment data structure remains the same (green).

We can predict the number of calls to the environment data structure with respect to the argument of the Fibonacci function based on the formula:
predicted value = previous × 1.618^(difference in n).

Fibonacci function	Lookup-variable-value-operation-count	Estimated-operation-number	Primitive-application-operation-count	Estimated-operation-number
n = 2	17		6	
n = 3	29	17*1.618^1=27.5	11	6*1.618^1=9.708
n = 4	53	29*1.618^1=46.9	21	11*1.618^1=17.798
n = 8	401	53*1.618^4=363	166	21*1.618^4=144
n = 16	19157	401*1.618^8=18835	7981	166*1.618^8=7797

We see that the growth in the number of operations on the environment data structure, with respect to the argument of the Fibonacci function, is related to the golden ratio of 1.618.

It would be great if we could formally discuss these predictions without having to repeatedly construct tables for each function.

Appendix A

```
;;Note: This analyze is modified from original version.
;;      mc-eval have changed name to a-eval in order to
;;      distinguish from original meta-circular mc-eval.
(load  "~/cs61a/work/4.1/analyze.scm")


(define test-environment #f)

(define (initialization param value)
  (set! mc-eval-table (make-hash-table eq?))
  (set! test-environment (setup-environment))
  (mc-eval (list 'define param value) test-environment)
  )

(define (speed-experiment exp param value)
  (begin
    (initialization param value)
    (mc-eval exp test-environment)
    (hash-table-for-each mc-eval-table   ;;hash-table-for-each is a system function
                          (lambda (key value)
                            (newline)
                            (display (cons key value))))
    (newline)
    (display "-----dividing--line-------")
    (newline)
    (initialization param value)
    ((analyze exp) test-environment)
    (hash-table-for-each mc-eval-table
                          (lambda (key value)
                            (newline)
                            (display (cons key value))))
    'done))
(define (test-fib value)
  (speed-experiment '((lambda ()
                         (define (fib n)
                           (cond ((< n 2) n)
                                 (else (+ (fib (- n 1))
                                          (fib (- n 2))))))
                         (fib x)))
                     'x
                     value)
  )
```