Metacircular Evaluator (MCE)

(read-eval-print loop)

(define (driver-loop)
  (prompt-for-input (input-prompt)
  (let ((input (read)))
    (let ((output
      (mc-eval input
      the-global-environment)))
      (announce-output output-prompt)
      (user-print output)))
  (driver-loop)))

Today’s Plan

• Is mc-eval basically the same as eval-1?
  – Yes
• Is mc-apply basically the same as apply-1?
  – Yes
• How is this different than scheme-1?
  – Everything has its own ADT!
  – We have environments and can define things!

CS61A Lecture 23

2011-07-28
Colleen Lewis

scheme-1 Review

(define (scheme-1)
  (display "Scheme-1: ")
  (flush)
  (print (eval-1 (read)))
  (scheme-1))

Whatever you typed in is treated as a list

Infinite loop
eval-1 evaluated these lists

The big idea!

(define (driver-loop)
  (prompt-for-input (input-prompt)
  (let ((input (read)))
    (let ((output
      (mc-eval input
      the-global-environment)))
      (announce-output output-prompt)
      (user-print output)))
  (driver-loop)))

The big idea!

(define (mc-eval exp env)
  (cond
    ((self-evaluating? exp) ...)
    ((variable? exp) ...)
    ((quoted? exp) ...)
    ((assignment? exp) ...)
    ((definition? exp) ...)
    ((if? exp) ...)
    ((lambda? exp) ...)
    ((begin? exp) ...)
    ((cond? exp) ...)
    ((application? exp) ...)
    (else (error "what?")}))

Is caught by:
A. self-evaluating?
B. variable?
C. quoted?

(define (mc-eval exp env)
  (cond
    ((self-evaluating? exp) ...)
    ((variable? exp) ...)
    ((quoted? exp) ...)
    ((assignment? exp) ...)
    ((definition? exp) ...)
    ((if? exp) ...)
    ((lambda? exp) ...)
    ((begin? exp) ...)
    ((cond? exp) ...)
    ((application? exp) ...)
    (else (error "what?")]))

Is caught by:
A. self-evaluating?
B. variable?
C. quoted?
More things create/use ADTs
(makes not-new stuff different)

STk> (eval-1 '(lambda (x) (* x x)))
(l lambda (x) (* x x))

STk> (mc-eval '(lambda (x) (* x x)) '())
(procedure (x) (** x x) '())

ADT overkill?
This is tagged with procedure, but we already had it tagged with lambda.

What do environments look like?

Frames in MCE
(below the line)

Global
x: 2
y: 4

((x y) . (2 4)) ((a b c) . (5 7 3))
or
((x y) 2 4) ((a b c) 5 7 3)

(define (frame-variables frame)
  (car frame))
(define (frame-values frame)
  (cdr frame))

Environments
(below the line)

List of frames!
(define the-empty-environment '())
(define (extend-environment vars vals base-env)
  (make-frame vars vals base-env))

Error checking omitted

Global
x: 2
y: 4

(a: 5
b: 7
c: 3)

Frame

Environment

((x y) . (1 2))

(((a b c) . (5 7 3)) (((x y) . (1 2)))

Environments
(Below the line)
E3
a: 5
b: 7
c: 3

E1
x: 2
y: 4

E2

How do we look-up values from environments?

(define (scan vars vals)
  (cond ((null? vars)
          ...) ;; look in enclosing env.
       ((eq? var (car vars))
        (car vals))
       (else
        (scan (cdr vars) (cdr vals)))))

How many times is scan called?

A. Once for each frame
B. Once for each variable in the environment
C. Once for each variable you are looking up

What does this environment look like?

STk>(define a 3)
STk>(define sq (lambda (x) (* x x)))

How do we look-up values from environments? (continued)

(define (lookup-variable-value var env)
  (define (env-loop env)
    (if (eq? env the-empty-environment)
        (error "Unbound variable" var)
        (let ((frame (first-frame env)))
          (scan (frame-variables frame) (frame-values frame)))))
  (env-loop env))
What is a procedure?

STk> (mc-eval '(lambda (x) (* x x)) '(((a) 3)))
(procedure (x) ((x x)) (((a) 3)))

The environment

What does this environment look like?

STk> (define a 3)
STk> (define sq (lambda (x) (* x x)))

What would scheme print (wwsp)?

(define (my-scope x)
  (lambda () x))
(define (current-scope x thunk)
  (thunk))

STk> (define my-thunk (my-scope 3))
my-thunk
STk> (current-scope 4 my-thunk)
Prints:
A. 3  B. 4  C. error  D. ???

Lexical vs. Dynamic Scope

- Scheme – Lexical Scope
  - Extend the frame that the procedure was created in
- Logo – Dynamic Scope
  - Extend the frame that the procedure was called from

Printing Environments is...

A. going to be really helpful to see what is going on in mc-eval
B. not going to be possible because they are really big
C. not going to be possible because they contain infinite structures

LOGO

Demo
Commands versus Operations

- In LOGO procedures are divided into
  - Operations – return values
  - Commands – don’t return values
- You have to start each instruction with a command

```logo
print sum 2 3
```

Parentheses can be used

```logo
print (sum 2 3 4 5)
print 3*(4+5)
```

Variables vs. Procedures

- We can have a function and a variable with the same name in LOGO.
- How to make a variable:
  ```logo
  make "x 10
  print :x
  make "sum 15
  print sum :x :sum
  ```

Quoting things in LOGO

- We use " instead of single quotes.
  ```logo
  make "name "colleen
  print :name
  make "my-sent [a b c]
  print :my-sent
  ```

There are no special forms!

- We can just quote things by putting them in [] and then they won’t be evaluated –WOW!

```logo
ifelse 2=3 [print "hi] [print "bye]
```

Defining a function

- We use the word “to” - “to teach logo a new word”.
  ```logo
  ? to add-up :x :y :z
  > sum :x :y :z
  > end
  ? print add-up 1 2 3
  ```
Scope - We have frames

- We have frames so calling a function creates a new bind – it doesn't change the global frame.

```lisp
(make "x 10
? to add-up :x :y :z
> sum :x :y :z
> end
? print add-up 1 2 3
? print :x
```

New frames extend the CURRENT environment (not the environment in which they were created)

```lisp
(make "pi 3.14
? to area
> :radius * :pi
> end
? to mess-up :pi
> area 5
> end
? mess-up 4
```

THE BIG IDEA!

Will LOGO return:
A. 20  
B. 15.70  
C. ??

Solutions

E3 is the current frame. Draw the environment. How many elements are in the list you made?
A. 1  
B. 2  
C. 3  
D. 4  
E. 5

```
(car (cadr (caddr ((a b c) (5 7 3)))))
(car (cadr ((x y) (1 2))))
```

Global
x: 2  
y: 4