6.001 SICP

Environment model

- Models of computation
  - Substitution model
    - A way to figure out what happens during evaluation
      - `(define l '(a b c))`
      - `(car l)` → `a`
      - `(define m '(1 2 3))`
      - `(car l)` → `a`
      - Not really what happens in the computer
        - `(car l)` → `a`
        - `(set-car! l 'z)`
        - `(car l)` → `z`
  - The Environment Model

Can you figure out why this code works?

```
(define make-counter
  (lambda (n)
    (lambda ()
      (set! n (+ n 1))
      n)))
```

```
(define ca (make-counter 0))
(ca) ==> 1
(ca) ==> 2
(define cb (make-counter 0))
(cb) ==> 1
(ca) ==> 3 ; ca and cb are independent
```

What the EM is:

- A precise, completely mechanical description of:
  - name-rule looking up the value of a variable
  - define-rule creating a new definition of a var
  - set!-rule changing the value of a variable
  - lambda-rule creating a procedure
  - application applying a procedure

- Enables analyzing more complex scheme code:
  - Example: `make-counter`

- Basis for implementing a scheme interpreter
  - for now: draw EM state with boxes and pointers
  - later on: implement with code

A shift in viewpoint

- As we introduce the environment model, we are going to shift our viewpoint on computation
  - Variable:
    - OLD – name for value
    - NEW – place into which one can store things
  - Procedure:
    - OLD – functional description
    - NEW – object with inherited context
  - Expressions
    - Now only have meaning with respect to an environment

Frame: a table of bindings

- Binding: a pairing of a name and a value

  Example:
  - `x` is bound to `15` in frame `A`
  - `y` is bound to `(1 2)` in frame `A`
  - the value of the variable `x` in frame `A` is `15`

Environment: a sequence of frames

- Environment `E1` consists of frames `A` and `B`
- Environment `E2` consists of frame `B` only
  - A frame may be shared by multiple environments
Evaluation in the environment model

- All evaluation occurs in an environment
  - The current environment changes when the interpreter applies a procedure
- The top environment is called the global environment (GE)
  - Only the GE has no enclosing environment
- To evaluate a combination
  - Evaluate the subexpressions in the current environment
  - Apply the value of the first to the values of the rest

Name-rule

- A name X evaluated in environment E gives the value of X in the first frame of E where X is bound
- In E1, the binding of x in frame A shadows the binding of x in B

Define-rule

- A define special form evaluated in environment E creates or replaces a binding in the first frame of E

Set!-rule

- A set! of variable X evaluated in environment E changes the binding of X in the first frame of E where X is bound

Your turn: evaluate the following in order

```
(+ z 1) | x1 ==>(set! z (+ z 1)) | x1 (define z (+ z 1)) | x1 (set! y (+ z 1)) | x1
```

Double bubble: how to draw a procedure

```
(lambda (x) (* x x))
```

A compound proc that squares its argument

(parameters: x)

body: (* x x)
Lambda-rule

- A lambda special form evaluated in environment E creates a procedure whose environment pointer is E

\[\text{define square (lambda (x) (* x x))} \mid E_1\]

Environment pointer points to frame A because the lambda was evaluated in E1 and E1 \(\rightarrow\) A

To apply a compound procedure P to arguments:

1. Create a new frame A
2. Make A into an environment E:
   - A's enclosing environment pointer goes to the same frame as the environment pointer of P
3. In A, bind the parameters of P to the argument values
4. Evaluate the body of P with E as the current environment

Example: inc-square

\[\text{define inc-square (lambda (y) (+ 1 (square y)))} \mid GE\]

Example cont'd: (inc-square 4) \(\mid GE\)

Example cont'd: (square y) \(\mid E_1\)
Lessons from the inc-square example

- EM doesn't show the complete state of the interpreter
  - missing the stack of pending operations
  - The GE contains all standard bindings (*, cons, etc)
    - omitted from EM drawings
- Useful to link environment pointer of each frame to the procedure that created it

Example: make-counter

- Counter: something which counts up from a number

\[
\begin{align*}
\text{(define make-counter} & \ (\lambda (n) \\
& \ (\lambda () (\text{set! } n (+ n 1))) \\
& \ n) \\
)\end{align*}
\]

\[
\begin{align*}
\text{(define ca (make-counter 0))} & \ (ca) \Rightarrow 1 \\
\text{(define cb (make-counter 0))} & \ (cb) \Rightarrow 1 \\
\text{(ca) \Rightarrow 2} & \ ; \text{ca and cb are independent}
\end{align*}
\]
Lessons from the `make-counter` example

- Environment diagrams get complicated very quickly
- Rules are meant for the computer to follow, not to help humans
- A lambda inside a procedure body captures the frame that was active when the lambda was evaluated
- This effect can be used to store local state