Environment model

- Models of computation
- Substitution model
  - A way to figure out what happens during evaluation
    (define l '(a b c))
    (car l) ==> a
    (car l) ==> a
  - Not really what happens in the computer
    (car l) ==> a
    (car l) ==> x
- The Environment Model

Why does this code work?

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

(define ca (make-counter 0))
(ca) ==> 1
(ca) ==> 2 ; not functional programming!
(define cb (make-counter 0))
(cb) ==> 1
(ca) ==> 3 ; ca and cb are independent

The Environment Model

- The Environment Model (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 variable
  - set!-rule changing the value of a variable
  - lambda-rule creating a procedure
  - application applying a procedure
- EM enables analyzing more complex Scheme code
  - Example: make-counter
  - EM will be a basis for implementing a Scheme interpreter
    - for now, we'll just draw EM state with boxes and pointers
    - later on, we'll implement EM 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

This arrow is called the enclosing environment pointer
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

\[ x \mid \text{GE} \implies 3 \quad z \mid \text{E1} \implies 20 \]

- In E1, the binding of x in frame A shadows the binding of x in B

\[ x \mid \text{GE} \implies 3 \quad x \mid \text{E1} \implies 15 \]

Define-rule

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

\[
\begin{align*}
\text{(define z 20)} & \mid \text{GE} \\
\text{(define z 25)} & \mid \text{E1}
\end{align*}
\]

\[ x \mid \text{GE} \implies 3 \\
x \mid \text{E1} \implies 15 \\
z \mid \text{GE} \implies 20 \\
z \mid \text{E1} \implies 25 \]

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

\[
\begin{align*}
\text{(set! z 20)} & \mid \text{GE} \\
\text{(set! z 25)} & \mid \text{E1}
\end{align*}
\]

\[ z \mid \text{GE} \implies 25 \\
z \mid \text{E1} \implies 20 \]

Define versus Set!

Using define

\[ x \mid \text{GE} \implies 3 \\
z \mid \text{E1} \implies 20 \\
x \mid \text{E1} \implies 15 \\
z \mid \text{E1} \implies 25 \]

Using set!

\[ x \mid \text{GE} \implies 3 \\
z \mid \text{E1} \implies 20 \\
x \mid \text{E1} \implies 15 \\
z \mid \text{E1} \implies 25 \]

Define-rule for existing variables (DrScheme only)

- If X is already bound in the first frame of E, and E is:
  - the global environment: define replaces the binding
  - any other environment: it is an error

\[
\begin{align*}
\text{(define z 30)} & \mid \text{GE} \\
\text{(define z 35)} & \mid \text{E1}
\end{align*}
\]

Error: z is already defined
Your turn: evaluate the following in order

\[
(\text{+ } z \text{ 1}) \mid E_1 \\
\text{(set! } z \text{ (+ } z \text{ 1}) \mid E_1 \\
\text{(define } z \text{ (+ } z \text{ 1}) \mid E_1 \\
\text{(set! } y \text{ (+ } z \text{ 1}) \mid EM) \\
\text{Error: unbound variable: } y
\]

Double bubble: how to draw a procedure

\[
\text{(lambda (x) (* x x))} \mid [\text{compound} \ldots] \\
\text{READ} \\
\text{EVAL} \\
\text{A compound proc that squares its argument} \\
\text{PRINT}
\]

Lambda-rule

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

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

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

\[
(\text{square 4}) \mid E_1 \\
\text{GE} \\
\text{x: 10} \\
\text{square:} \\
\text{parameters: x} \\
\text{body: (* x x)} \\
\text{square} \mid E_1 \Rightarrow \text{#proc} \\
\text{(* x x)} \mid E_1 \Rightarrow 16 \\
\text{*} \mid E_1 \Rightarrow \text{#prim} \\
\text{x} \mid E_1 \Rightarrow 4
\]

Achieving Inner Peace (and A Good Grade), Part II

\[
\text{*Om Mani Padme Hum...}
\]

1. Make a new frame
2. Link frame to the environment pointer of procedure
3. Bind parameters in the new frame
4. Evaluate procedure body in the new frame
Example: inc-square

```
(define square (lambda (x) (* x x))) | =
(define inc-square (lambda (y) (+ 1 (square y)))) | =
```

Example cont'd: (inc-square 4) | =

```
GE
inc-square: square:

GE
inc-square:

GE
square: inc-square:
```

Example cont'd: (square y) | =

```
GE
inc-square: square:

GE
inc-square:

GE
square: inc-square:
```

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)
- usually omitted from EM drawings
- Useful to link the environment pointer of each new frame to the procedure that created it

Example: make-counter

- Counter: counts upwards from a starting number

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

```
(define ca (make-counter 0))
```

```
(define cb (make-counter 0))
```
Capturing state in local frames & procedures

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
Environments are important in other languages

USA
- Macintosh | USA
- Milkshake | USA
- Canadian bacon | New England

Britain

New England
- Canadian bacon | New England

Canada
- Unbound variable!!
- Canadian bacon | Canada