

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



1/31

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
```

2/31

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 var
 - 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

3/31

A shift in viewpoint

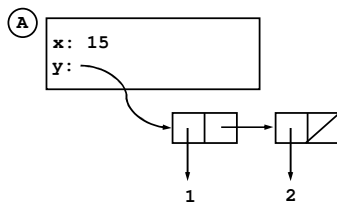
- As we introduce the environment model, we are going to shift our viewpoint on computation
- Variable
 - OLD – name for value
 - NEW – place into which one can store things
- Procedure
 - OLD – functional description
 - NEW – object with **inherited context**
- Expressions
 - Now **only** have meaning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

4/31

Frame: a table of bindings

- Binding:** a pairing of a name and a value

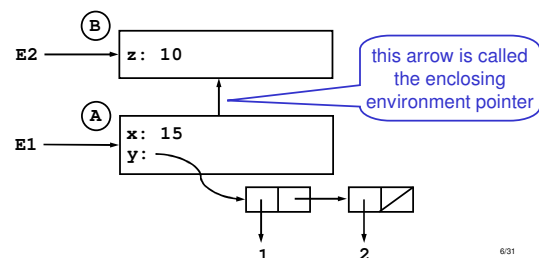
Example: x is bound to 15 in frame A
 y is bound to (1 2) in frame A
 the value of the variable x in frame A is 15



5/31

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**



6/31

Evaluation in the environment model

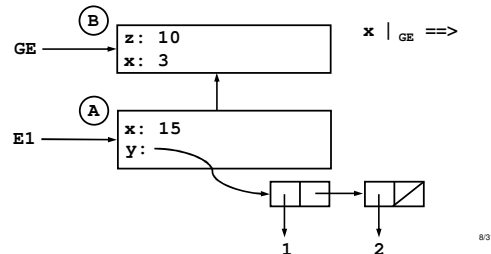
- All evaluation occurs **in an environment**
 - The **current environment** changes when the interpreter applies a procedure
- The top environment is called the **global environment (GE)**
 - Only the GE has no enclosing environment**
- To **evaluate** a combination
 - Evaluate the subexpressions *in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

7/31

Name-rule

- A name X evaluated in environment E gives **the value of X in the first frame of E where X is bound**

$$z \mid_{GE} \Rightarrow \quad z \mid_{E1} \Rightarrow \quad x \mid_{E1} \Rightarrow$$
- In E1, the binding of x in frame A **shadows** the binding of x in B

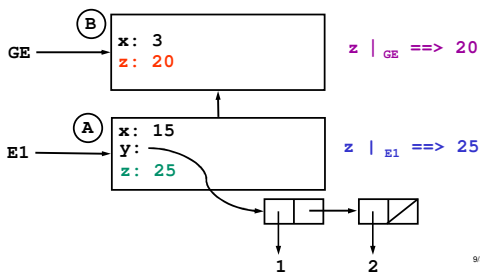


8/31

Define-rule

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

`(define z 20) |GE` `(define z 25) |E1`

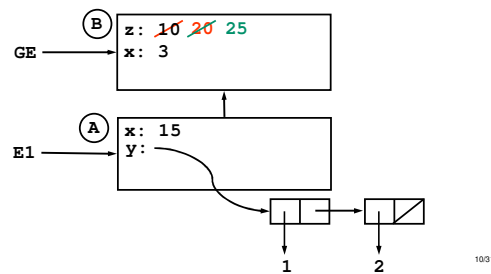


9/31

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**

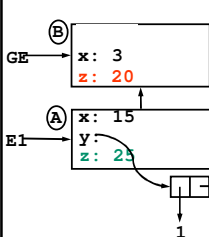
`(set! z 20) |GE` `(set! z 25) |E1`



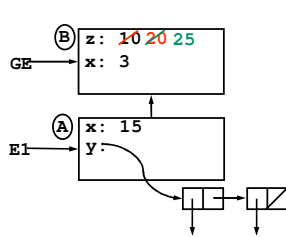
10/31

Define versus Set!

Using define



Using set!

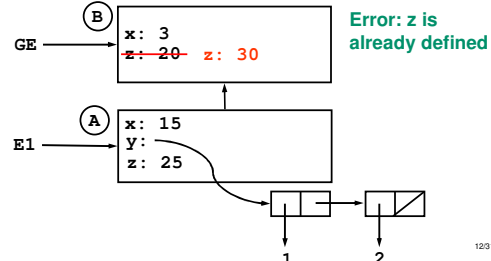


11/31

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**

`(define z 30) |GE` `(define z 35) |E1`



12/31

Your turn: evaluate the following in order

```

(+ z 1) | E1 ==> 11
(set! z (+ z 1)) | E1 (modify EM)
(define z (+ z 1)) | E1 (modify EM)
(set! y (+ z 1)) | GE (modify EM)

```

Error: unbound variable: y

1301

Double bubble: how to draw a procedure

```

(lambda (x) (* x x)) #[compound-...]

```

READ → EVAL → PRINT

A compound proc that squares its argument

Environment pointer

Code pointer

parameters: x
body: (* x x)

1501

Lambda-rule

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

```

(define square (lambda (x) (* x x))) | E1

```

1601

To apply a compound procedure P to arguments:

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

1701

(square 4) | GE

1801

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

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 environment

*Om Mani Padme Hum...

1901

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



Diagram illustrating a hardware accelerator for the Fibonacci sequence. The accelerator consists of a control unit labeled `inc-square: square:` and a register labeled `y: 4`.

The control unit receives a `GE` signal and a `p: x` input. It outputs `b: (* x x)` and `p: y`. The `p: y` output is connected to the `y: 4` register via a blue line labeled `E1`.

The state transition logic is defined as follows:

```

inc-square | GE ==> #[proc]
p: y | E1 ==> #[prim] (+ 1 (square y))
              | E1 ==> #[prim] (square y)
  
```

[illegible]

- 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



```
(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
(cb) ==> 2 ; ca and cb are independent
```

[illegible]

