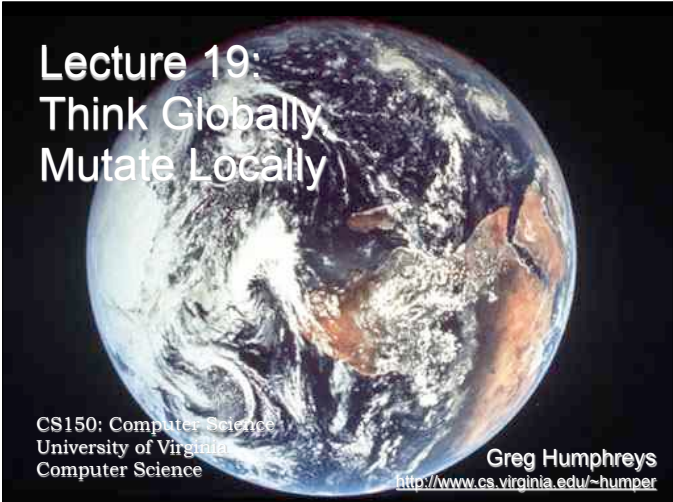


Lecture 19: Think Globally Mutate Locally



CS150: Computer Science
University of Virginia
Computer Science

Greg Humphreys
<http://www.cs.virginia.edu/~humper>

Menu

- Environments
- Evaluation Rules

```
(define nest  
  (lambda (x)  
    (lambda (x)  
      (+ x x))))  
> ((nest 3) 4)
```

8

Does the substitution model of evaluation tell us how to evaluate this?

Names, Places, Mutation

- A name is a **place** for storing a value.
- **define** creates a new place
- **cons** creates two new places, the **car** and the **cdr**
- **(set! name expr)** changes the value in the place *name* to the value of *expr*
- **(set-car! pair expr)** changes the value in the **car** place of *pair* to the value of *expr*
- **(set-cdr! pair expr)** changes the value in the **cdr** place of *pair* to the value of *expr*

Lambda and Places

- **(lambda (x) ...)** also creates a new place named *x*
- The passed argument is put in that place

```
> (define x 3)  
> ((lambda (x) x) 4)
```

4

```
> x  
3
```

x: 3

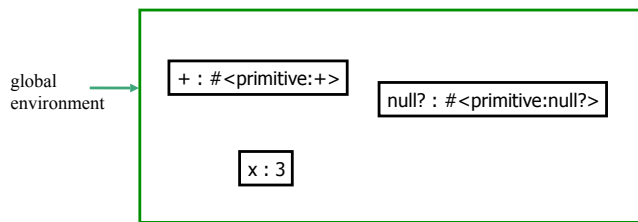
x: 4

How are these places different?

Location, Location, Location

- Places live in **frames**
- An **environment** is a pointer to a frame
- We start in the **global environment**
- Application creates a new frame
- All frames except the global frame have exactly one parent frame

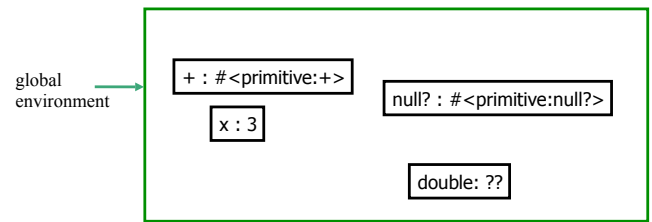
Environments



The global environment points to the outermost frame. It starts with all Scheme primitives.

```
> (define x 3)
```

Procedures



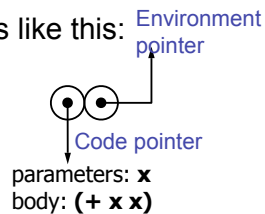
```
> (define double (lambda (x) (+ x x)))
```

How to Draw a Procedure

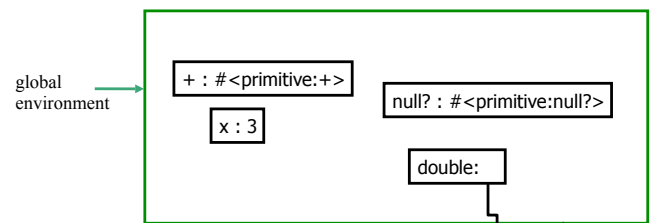
- A procedure needs **both code and an environment**

– We'll see why soon

- We draw procedures like this:



Procedures



```
> (define double
  (lambda (x) (+ x x)))
```

parameters: **x**
body: **(+ x x)**

Application

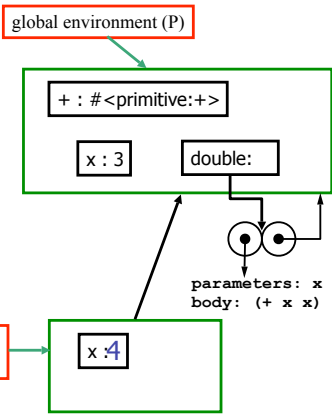
- Old rule: (Substitution model)

Apply Rule 2: Compounds. If the procedure is a *compound procedure*, **evaluate** the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.

New Rule: Application

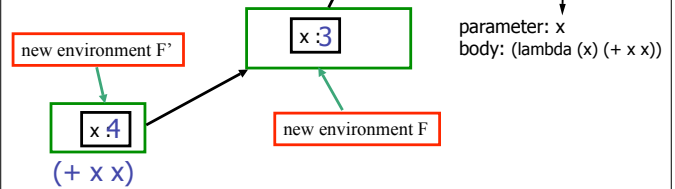
- Construct a new frame "F", parented by the current environment "P"
- Create places in F with the names of each parameter
- Put the values of the parameters in those places
- Set the current frame to F
- Evaluate the body
- Set the current frame to P
- Discard F

1. Construct a new frame "F", parented by the current environment "P"
2. Create places in F with the names of each parameter
3. Put the values of the parameters in those places
4. Set the current frame to F
5. Evaluate the body
6. Set the current frame to P
7. Discard F



> (double 4)
8

```
(define nest
  (lambda (x)
    (lambda (x)
      (+ x x))))
> ((nest 3) 4)
((lambda (x) (+ x x)) 4)
```



Evaluation Rule 2 (Names)

If the expression is a *name*, it evaluates to the value associated with that name.

To find the value associated with a name, look for the name in the frame pointed to by the current environment. If it contains a place with that name, use the value in that place. If it doesn't, evaluate the name using the parent environment as the new evaluation environment. If the frame has no parent, error.

evaluate-name

```
(define (evaluate-name name env)
  (if (null? env) (error "Undefined name: ...")
      (if (frame-contains name (get-frame env))
          (lookup name (get-frame env))
          (evaluate-name name
                        (parent-environment
                         (get-frame env))))))
```

Hmm...maybe we can define a Scheme interpreter in Scheme!

Charge

- Remember, mutation makes evaluation rules more complicated
- Now, you *really* know everything you need to finish PS5.
 - So you should do so
- Make Spring Break plans
- Very cool talk tomorrow (skip class for this)
 - “Dual Photography”
 - Thursday, March 2, 3:30
 - Olsson 009