Class 19: Think Globally, Mutate Locally

Menu
- Environments
- Evaluation Rules
- Exam 1

---

CS150 Fall 2005: Lecture 19: Environments

Review: 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

```scheme
(define nest
  (lambda (x)
    (lambda (x)
      (+ x x))))

> ((nest 3) 4) 8
```

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

---

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, global frame has no parent

---

Review: 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

```scheme
(define nest
  (lambda (x)
    (lambda (x)
      (+ x x))))

> ((nest 3) 4) 8
```

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

  ![Diagram of a procedure](image)

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

1. Construct a new frame, enclosed in the environment of this procedure
2. Create places in that frame with the names of each parameter
3. Put the values of the parameters in those places
4. Evaluate the body in the new environment

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

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 evaluation environment. If it contains a place with that name, use the value in that place. If it doesn’t, evaluate the name using the frame’s parent environment as the new evaluation environment. If the frame has no parent, error (name is not a place).

Charge

- Mutation makes evaluation rules more complicated
- Environment diagrams quickly get complicated – but like substitution evaluations, just follow rules mechanically
- Monday, we’ll finish Colossus (don’t worry, the Allies still won)