Lecture 19: Programming with State

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

Notices
- Today: normal lab hours (4-5:30pm)
- Thursday’s lab hours will be 5-7pm (instead of normally scheduled times)
- Spring Break – there will not be normally scheduled lab hours or office hours between March 3 and March 10. Normal lab hours resume Sunday, March 11.
- Exam 1 will be returned at end of today’s class
- Schedule for rest of semester is updated on the course web (www.cs.virginia.edu/schedule/)

From Lecture 3:

Evaluation Rule 2: Names
If the expression is a name, it evaluates to the value associated with that name.

> (define two 2)
> two
2

Names and Places
- A name is not just a value, it is a place for storing a value.
- define creates a new place, associates a name with that place, and stores a value in that place

(define x 3)

set! should make you nervous

set! (“set bang”) changes the value associated with a place

> (define x 3)
> (nextx)
3
> (set! x 7)
> x
7

Before set! all procedures were functions (except for some with side-effects).
The value of (f) was the same every time you evaluate it. Now it might be different!
Lecture 19: Mutation

Defining nextx

(define (nextx)
  (set! x (+ x 1))
  x)

Syntactic sugar for

(define nextx
  (lambda ()
    (begin
      (set! x (+ x 1))
      x))))

Evaluation Rules

> (define x 3)
> (+ (nextx) x)
7
or 8
> (+ x (nextx))
9
or 10

DrScheme evaluates application subexpressions left to right, but Scheme evaluation rules allow any order.

set-car! and set-cdr!

(set-car! p v)
Replaces the car of the cons p with v.

(set-cdr! p v)
Replaces the cdr of the cons p with v.

These should scare you even more then set!!

> (define pair (cons 1 2))
> pair
(1 . 2)
> (set-car! pair 0)
> (car pair)
0
> (cdr pair)
2
> (set-cdr! pair 1)
> pair
(0 . 1)

map

Functional Solution: A procedure that takes a procedure of one argument and a list, and returns a list of the results produced by applying the procedure to each element in the list.

(define (map proc lst)
  (if (null? lst) null
      (cons (proc (car lst))
            (map proc (cdr lst)))))

Imperative Solution

A procedure that takes a procedure and list as arguments, and replaces each element in the list with the value of the procedure applied to that element.

(define (map! f lst)
  (if (null? lst) (void)
      (begin
        (set-car! lst (f (car lst)))
        (map! f (cdr lst)))))

Programming with Mutation

> (map! square (intsto 4))
> (define i4 (intsto 4))
> (map! square i4)
> i4
(1 4 9 16)

> (define i4 (intsto 4))
> (map square i4)
(1 4 9 16)
> i4
(1 2 3 4)

Mutation Changes Everything!

• We can no longer talk about the “value of an expression”
  – The value of a given expression can change!
  – We need to talk about “the value of an expression in an execution environment”
• The order in which expressions are evaluated now matters

Why Substitution Fails?

> (define (nextx) (set! x (+ x 1)) x)
> (define x 0)
> ((lambda (x) (+ x x)) (nextx))
2

Substitution model:
(+ (nextx) (nextx))
(+ (begin (set! x (+ x 1)) x) (begin (set! x (+ x 1)) x))
(+ (begin (set! 0 (+ 0 1)) 0) (begin (set! 0 (+ 0 1)) 0))
(+ 0 0)
0

Names and Places

• 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

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)
> x
3

How are these places different?
Location, Location, Location

- Places live in frames
- An environment is a frame and a pointer to a parent environment
- All environments except the global environment have exactly one parent environment; global environment has no parent
- Application creates a new environment

Environments

$\begin{align*}
\text{global environment} & : + : \# \langle \text{primitive: +} \rangle \\
& \quad \text{null? : } \# \langle \text{primitive: null?} \rangle \\
& \quad x : 3 \\
\end{align*}$

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

> (define x 3)

Evaluation Rule 2: Names

A name expression evaluates to the value associated with that name.

To find the value associated with a name, look for the name in the frame associated with the evaluation environment. If it contains a place with that name, the value of the name expression is the value in that place. If it doesn’t, the value of the name expression is the value of the name expression evaluated in the parent environment if the current environment has a parent. Otherwise, the name expression evaluates to an error (the name is not defined).

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

Procedures

$\begin{align*}
\text{global environment} & : + : \# \langle \text{primitive: +} \rangle \\
& \quad \text{null? : } \# \langle \text{primitive: null?} \rangle \\
& \quad x : 3 \\
& \quad \text{double: } ?? \\
\end{align*}$

> (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)`

**Application**

- Old rule: (Substitution model)

**Apply Rule 2: Constructed Procedures.** To apply a constructed procedure, **evaluate** the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.

**New Application Rule 2:**

1. Construct a new environment, whose parent is the environment to which the environment pointer of the applied procedure points.
2. Create a place in that frame for each parameter containing the value of the corresponding operand expression.
3. Evaluate the body in the new environment. Result is the value of the application.

**Charge**

- **WHAOO! Auctions**
  - Now: Return Exam 1
    - My stack of exams is sorted by first name (as you wrote it on your exam)
    - What is a good algorithm for sorting you all in order to hand back the exams?
  - PS5: You know everything you need to do it after today, so start early!
  - Friday: Error Messages, Golden Ages, Sex, Politics, and Religion