Programming Languages Recitation

Scheme

Arpit Jain

Computer Science Department
Courant Institute of Mathematical Sciences
New York University

arpit.jain@cs.nyu.edu

October 2, 2014

Overview

Scheme

Characteristics

- Statically scoped
- Dynamically typed
- First class functions
- Garbage collection
- Simple syntax !
- Continuation

Scheme Interpreter

Logistics

- Racket Scheme interpreter - Downloaded from racket-lang.org
- Set language to R5RS
- You are good to go !

Sample programs

- $(+\ 1\ 2) \Rightarrow 3$
- $(*\ 3\ 4) \Rightarrow 12$
- '(hello world) $\Rightarrow$ (hello world)

List

- Expressions are either atoms or lists
- Atoms are either constants (Boolean, numeric, string) or symbols
- If list is a computation first element must evaluate to an operation and remaining are actual parameter
- (+ (* 10 10) (* 5 10)) $\Rightarrow$ 150

List Manipulation

- car : Get head of the list. Returns an atom
- cdr: Get rest of the list. Returns a list
- cons: Prepend an element to a list
- '() Null list

Describing Data

Notation

- (quote (1 2 3)) $\Rightarrow$ (1 2 3)
- '(or this way) $\Rightarrow$ (or this way)

List decomposition example

- (car '(list of symbols))
- (cdr '(1 2 3))
- (cdr '(this list))

List decomposition example

- (car '(list of symbols))
  - $\Rightarrow$ this
- (cdr '(1 2 3))

- (cdr '(this list))

List decomposition example

- (car '(list of symbols))
  - $\Rightarrow$ this
- (cdr '(1 2 3))
  - $\Rightarrow$ (2 3)
- (cdr '(this list))

List decomposition example

- (car '(list of symbols))
  - ⇒ this
- (cdr '(1 2 3))
  - ⇒ (2 3)
- (cdr '(this list))
  - ⇒ (list)

List Decomposition shortcut

- (cadr X) $\Leftrightarrow$ (car (cdr X))
- (cdddr X) is $\Leftrightarrow$ (cdr (cdr (cdr X)))
- Up to 4 a's and/ or d's

List building

- (cons 'this '(that list)) ⇒ (this that list)
- (cons 'a '()) ⇒ (a)
- Shortcut: (list 'a 'b 'c 'd) ⇒ (a b c d)

Condition

(if condition expr1 expr 2)

General Form

(cond
(pred1 expr 1)
(pred 2 expr 2)
(pred 3 expr 3)
. . . . . .
(else expr))

- Evaluate preds in order until one is true
- Then evaluate the corresponding expr

Define

- (define (sqr n) (* n n))
- Body is not evaluated
- Binding is created. sqr is bound to the body
- Works for non functions too
- (define x 15)
- (define x '(list of elements))

Predicated

- list?
- number?
- pair?
- null?
- zero?

Usage

- (define x '(2 3 4))
- (list? x)
- $\#t$

Recursion on list

```
(define ( member elem lis )
(cond
((null? lis) #f)
((= elem (car lis )) lis )
(else (member elem (cdr lis )))))
```

- Convention: return rest of the list rather than #t

λ expressions

- (lambda (x) (+ x x)) $\Rightarrow$ returns a procedure
- ((lambda (x) (+ x x)) 5) $\Rightarrow$ 10
- (define doubleit (lambda (x) (+ x x))) $\Rightarrow$ Binding doubleit to the expression