6.001 SICP – October 27

6001-Introduction

Trevor Darrell  trevor@csail.mit.edu
32-D512

Office Hour: W 11

6.001 web page: http://sicp.csail.mit.edu/
section web page: http://www.csail.mit.edu/~trevor/6001/

• trees, revisited

---

Trees

• Type Definition:
  • Tree = List | Leaf
  • Leaf = C

• Operations:
  • leaf?
  • countleaves, scaletree, enum-leaves,
    map-tree & tree-ref

---

Example: deep-reverse

Reverse the order of a tree:
(deep-reverse (list 1 (list 2 3) (list 4 5 6))) ==> ((6 5 4)
  (3 2) 1)

(define (reverse lst)
  (if (null? lst)
      nil
      (append (reverse (cdr lst)) (list (car lst)))))

(define (deep-reverse tree)
  (if (not (pair? tree))
      tree
      (map deep-reverse (reverse tree)))
)

---

Example: map-tree

Write a procedure, (map-tree op tree),
So if tree has value

(((1 2) 3) (4 (5 6)) 7 (8 9 10))

then

(map-tree inc tree)

has value (((2 3) 4) (5 (6 7)) 8 (9 10 11))

You can use leaf? and map.

(map-tree
  (lambda (op tree)
    (if (leaf? tree)
        (op tree)
        (map (lambda (subtree) (map-tree op subtree))
             tree))))

---

General tree-manip

(define (tree-manip tree init leaf first rest accum)
  (cond ((null? tree) init)
        ((not (pair? tree)) (leaf tree))
        (else ( accum
                 (tree-manip (first tree)
                            init leaf first rest accum)
                 (tree-manip (rest tree)
                            init leaf first rest accum))))

Given: (define test-tree (list 1 (list 2 (list 3 (list 4) 5) 6) 7))

You can write expressions using tree-manip on test-tree will subsume many of the specific function we’ve just written, e.g.:
• Flatten a tree
• Deep-reverse a tree
• Sum up the values of the leaves of the tree
• Take the product of the even-valued leaves of the tree
• Create a new tree, which keeps the odd-valued leaves of the original tree within the same tree structure, but completely removes even-valued leaves.
General tree-manip

(define (tree-manip tree init leaf first rest accum)
  (cond ((null? tree) init)
        ((not (pair? tree)) (leaf tree))
        (else (accum
               (tree-manip (first tree) init leaf first rest accum)
               (tree-manip (rest tree) init leaf first rest accum))))

(test-tree
  (tree-manip test-tree 0 (lambda (x) 1) car cdr +)
  (tree-manip test-tree '() (lambda (x) (* x x)) car cdr cons)
  (tree-manip test-tree '() list car cdr append)
  (tree-manip test-tree '() (lambda (x) x) car cdr (lambda (a b) (append b (list a))))
  (tree-manip test-tree '() (lambda (x) x) car cdr (lambda (a b) (append b (list a))))
)