MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1996

Lecture Notes, September 24 – Aggregate Data

Common Patterns – List Procedures

Common Pattern #1: cdr'ing down a list

Recursive plan for `list-ref`:

```
; list-ref: List, Int → T
(define (list-ref lst n)
  (if (= n 0)
      (car lst)
      (list-ref (cdr lst) (- n 1))))
```

Common Pattern #2: cons'ing up a list

```
(define (copy lst)
  (if (null? lst)
      nil                           ; base case
      (cons (car lst)               ; recursion
            (copy (cdr lst)))))
```

Recursive plan for `append`:

```
(define (append list1 list2)
  (cond ((null? list1) list2)            ; base case
        (else
          (cons (car list1)              ; recursion
                (append (cdr list1))))))
```

Common Pattern #3: transforming a list

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

`square-em` using `map`:

Common Pattern #4: filtering

```
(define (filter pred lst)
  (cond ((null? lst) nil)
        ((pred (car lst))
         (cons (car lst)
               (filter pred (cdr lst))))
        (else (filter pred (cdr lst)))))
```

Common Pattern #5: enumeration

```
(define (integers-between low high)
  (if (> low high)
      nil
      (cons low (integers-between (+ low 1) high))))
```

Common Pattern #6: accumulation

```
(define (accumulate op init lst)
  (if (null? lst)
      init
      (op (car lst)
          (accumulate op init (cdr lst)))))
```

Write `length` as an accumulation:

Conventional Interfaces

```
(define (easy lo hi)
  (accum * 1
         (map fib
              (filter even?
                      (integers-between lo hi)))))
```

Draw `easy` as a series of black boxes connected by lists:

```
(define (hard lo hi)
  (cond ((> lo hi) 1)
        ((even? lo) (* (fib lo)
                       (hard (+ lo 1) hi)))
        (else (hard (+ lo 1) hi))))
```

Common Patterns - Tree Procedures

```
(define countleaves tree)
  (cond ((null? tree) 0)                  ;base case
        ((atom? tree) 1)                  ;base case
        (else (+ (countleaves (car tree))   ;tree-recursion
                 (countleaves (cdr tree))))))

(define (scale-tree tree factor)
  (cond ((null? tree) nil)
        ((atom? tree) (* tree factor))
        (else
          (cons (scale-tree (car tree) factor)
                (scale-tree (cdr tree) factor)))))

(define (scale-tree tree factor)
  (map (lambda (sub-tree)
         (if (atom? sub-tree)
             (* sub-tree factor)
             (scale-tree sub-tree factor)))
       tree))

(define (enumerate-tree tree)
  (cond ((null? tree) nil)
        ((atom? tree) (list tree))
        (else (append (enumerate-tree (car tree))
                      (enumerate-tree (cdr tree))))))
```

Trees and Conventional Interfaces

```
; Compute the sum of the squares of the odd leaves in a tree.
(define (sum-odd-squares tree)
  (cond ((null? tree) 0)
        ((atom? tree)
         (if (odd? tree) (square tree) 0))
        (else (+ (sum-odd-squares (car tree))
                 (sum-odd-squares (cdr tree))))))

; Construct a list of all the even Fibonacci numbers Fib(k) where k <= n
(define (even-fibs n)
  (define (next k)
    (if (< k n)
        nil
        (let ((f (fib k)))
          (if (even? f)
              (cons f (next (+ k 1)))
              (next (+ k 1))))))
  (next 0))
```

Using Conventional Interfaces:

```
(define (sum-odd-squares tree)
  (accumulate +
              0
              (map square
                   (filter odd?
                           (enumerate-tree tree)))))

(define (even-fibs n)
  (accumulate cons
              nil
              (filter even?
                      (map fib
                           (integers-between 0 n)))))
```

Draw using signal flow and conventional interfaces: