

Computer Science I

Professor Tom Ellman
Lecture 17

Comprehension

```
(define comprehension (lambda (lst pred?) ...?...))
```

Welcome to Dr. Scheme.

```
> (comprehension '(5 -2 9 -4) positive?)  
(5 9)  
> (comprehension '(5 2 9 4) negative?)  
( )  
> (comprehension '(5 2 9 4) (lambda (x) (< x 5)))  
(2 4)  
> (comprehension '(5 2 9 4)  
      (lambda (x) (and (< 2 x) (< x 9))))  
(5 4)
```

The COMPREHENSION Procedure

- COMPREHENSION takes a list LST and a predicate PRED? as inputs.
- The predicate PRED? should accept one argument.
- COMPREHENSION returns a list containing the members of LST that satisfy PRED.
- List order and multiple occurrences are preserved.

COMPREHENSION

```
(define comprehension  
  (lambda (lst pred?)  
    (cond ((null? lst) '())  
          ((pred? (car lst))  
           (cons (car lst) (comprehension (cdr lst) pred?)))  
          (else (comprehension (cdr lst) pred?))))
```

The PARTITION Procedure

```
(define partition (lambda (lst n) ...?...))
```

```
Welcome to Dr. Scheme.  
> (define nums '(5 2 9 4))  
> (partition nums 4)  
((2 4)(5 9))  
> (partition nums 9)  
((5 2 9 4)())  
> (partition nums 0)  
(()(5 2 9 4))
```

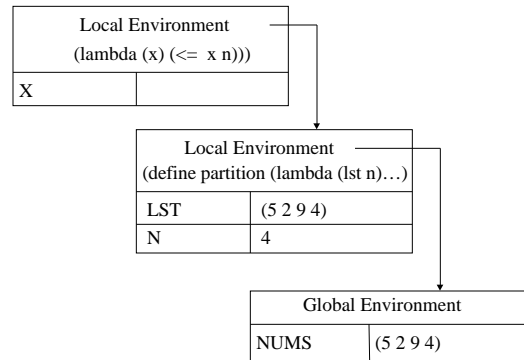
The PARTITION Procedure

- PARTITION takes a list LST of numbers and a single number N as inputs.
- PARTITION returns a list containing two other lists LOW and HIGH:
 - LOW is a list containing all members of LST that are less than or equal to N.
 - High is a list containing all members of LST that are greater than N.

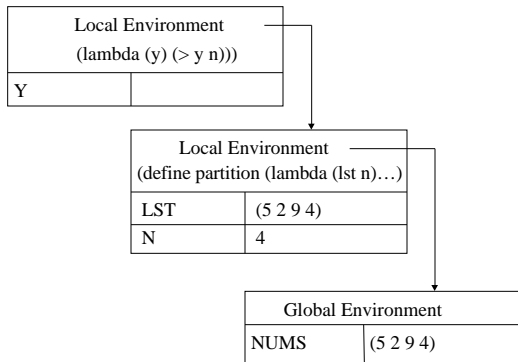
Definition of PARTITION

```
(define partition
  (lambda (lst n)
    (list (comprehension lst (lambda (x) (<= x n)))
          (comprehension lst (lambda (y) (> y n))))))
```

Environment for evaluating ($\leq X N$) during evaluation of (PARTITION NUMS 4):



Environment for evaluating ($> Y N$) during evaluation of (PARTITION NUMS 4):



Reduction

- A pattern of computation.
- Useful in a wide variety of situations.
- Comes in two varieties:
 - Flat reduction: (“List morphism”).
 - Deep reduction: (“Tree morphism”).
- Each is implemented by a higher-order procedure.

Flat Reduction

```
(define reduce-flat
  (lambda (lst fun base)
    (if (null? lst)
        base
        (fun (car lst) (reduce-flat (cdr lst) fun base)))))
```

Flat Reduction

```
(define reduce-flat
  (lambda (lst fun base)
    (letrec ((reduce-flat-helper
              (lambda (lst)
                (if (null? lst)
                    base
                    (fun (car lst) (reduce-flat-helper (cdr lst)))))))
      (reduce-flat-helper lst))))
```

Flat Reduction Examples

```
(define sum (lambda (lst) (reduce-flat lst + 0)))

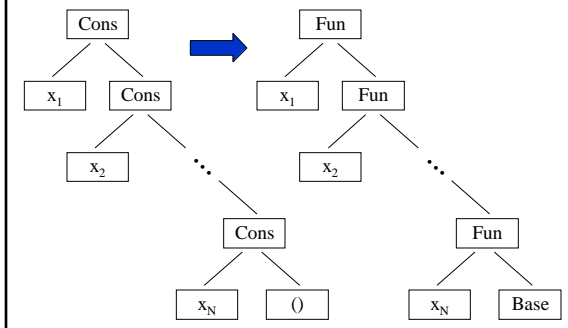
(define length (lambda (lst)
  (reduce-flat lst
    (lambda (x y) (+ 1 y))
    0)))
```

Flat Reduction Examples

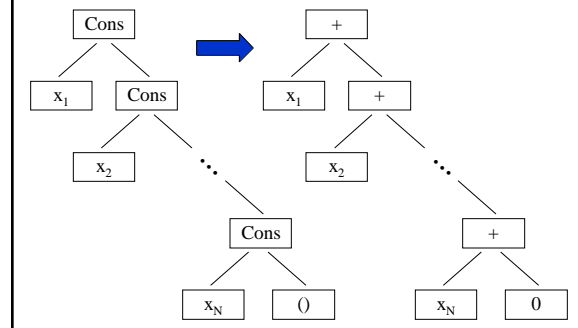
```
(define for-all?
  (lambda (lst p?)
    (reduce-flat (map p? lst)
      (lambda (x y) (and x y))
      #t)))

(define there-exists?
  (lambda (lst p?)
    (reduce-flat (map p? lst)
      (lambda (x y) (or x y))
      #f)))
```

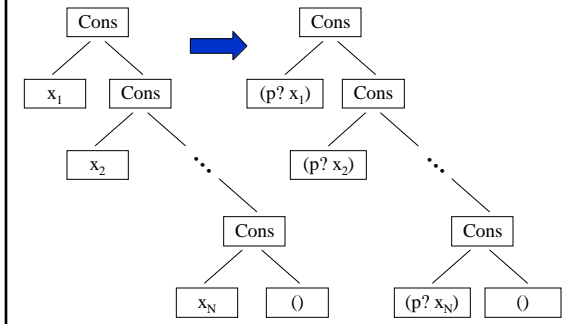
List Morphism



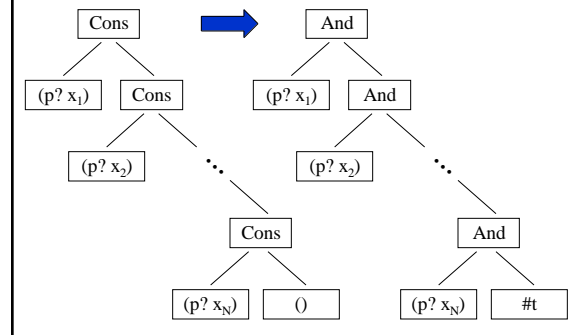
SUM as List Morphism



FOR-ALL? as List Morphism (MAP P? LST)



FOR-ALL? as List Morphism (REDUCE-FLAT ...)



Deep Reduction

```
(define reduce-deep
  (lambda (lst ind-fun base-fun)
    (if (not (pair? lst))
        (base-fun lst)
        (ind-fun (reduce-deep (car lst) ind-fun base-fun)
                  (reduce-deep (cdr lst) ind-fun base-fun))))))
```

Deep Reduction

```
(define reduce-deep
  (lambda (tree ind-fun base-fun)
    (letrec ((reduce-deep-helper
              (lambda (tree)
                (if (not (pair? tree))
                    (base-fun tree)
                    (ind-fun (reduce-deep-helper (car tree))
                            (reduce-deep-helper (cdr tree)))))))
      (reduce-deep-helper tree))))
```

Deep Reduction Examples

```
(define item-count
  (lambda (lst)
    (reduce-deep lst
                  +
                  (lambda (x) (if (null? x) 0 1)))))

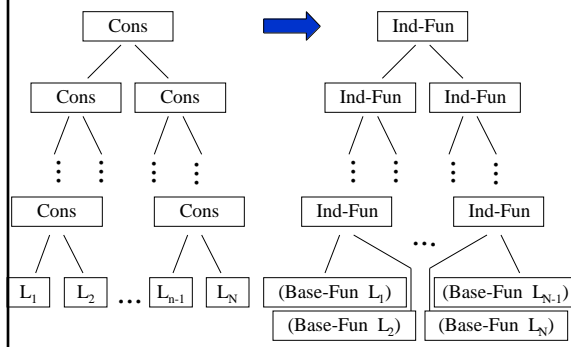
(define item-sum
  (lambda (lst)
    (reduce-deep lst
                  +
                  (lambda (x) (if (null? x) 0 x)))))
```

Deep Reduction Example

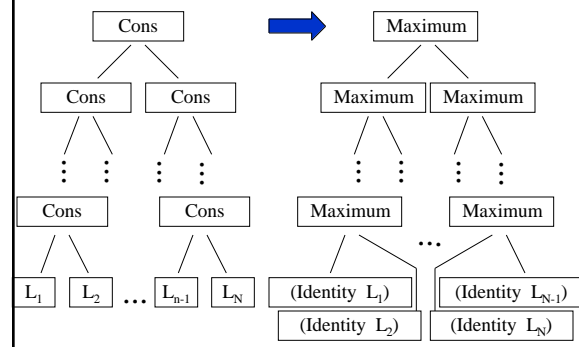
```
(define maximum
  (lambda (x y)
    (cond ((null? x) y)
          ((null? y) x)
          (else (max x y))))

(define max*
  (lambda (lst)
    (reduce-deep lst maximum identity)))
```

Tree Morphism



MAX* as Tree Morphism



Defining Procedures with an Arbitrary Number of Parameters

```
(define max (lambda (values ...?... )
```

```
  Welcome to Dr. Scheme.
```

```
  > (max 3 6 5)
```

```
  6
```

```
  > (max 5 3)
```

```
  5
```

```
  > (max 7)
```

```
  7
```

```
  > (max)
```

```
  'error
```

Defining Procedures with an Arbitrary Number of Parameters

```
(define max  
  (lambda (values  
    (cond ((null? values) 'error)  
          ((null? (cdr values)) (car values))  
          (else (let ((temp (apply max (cdr values))))  
                  (if (>= (car values) temp)  
                      (car values)  
                      temp)))))))
```

Defining Procedures with an Arbitrary Number of Parameters

```
(define and-proc
```

```
  (lambda (values (reduce-flat values
```

```
    (lambda (x y) (and x y))
```

```
    #t)))
```

```
(define or-proc
```

```
  (lambda (values (reduce-flat values
```

```
    (lambda (x y) (or x y))
```

```
    #f)))
```

FOR-ALL? and THERE-EXISTS? (Again)

```
(define for-all?
```

```
  (lambda (lst p?) (apply and-proc (map p? lst))))
```

```
(define there-exists?
```

```
  (lambda (lst p?) (apply or-proc (map p? lst))))
```