

Recursion

Specifying Incremental Work

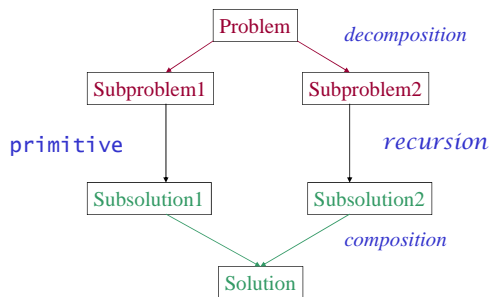
0	->	f(0)	
1	->	f(1)	$\Delta 1$
...		...	
n	->	f(n)	Δn
n+1	->	f(n+1)	

cs480(Prasad)

L14Recur

2

Divide and Conquer



cs480(Prasad)

L14Recur

3

Cartesian Product

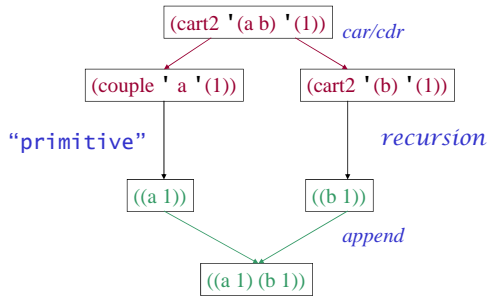
```
> (cart2 '(1 2) '(a b))
  ((1 a) (1 b) (2 a) (2 b))
> (cart2 '(0 1 2) '(a b))
  ((0 a) (0 b) (1 a) (1 b) (2 a) (2 b))
> (couple '1 '(b c))
  ((1 b) (1 c))
> (couple '1 '(a b c))
  ((1 a) (1 b) (1 c))
```

cs480(Prasad)

L14Recur

4

Divide and Conquer



cs480(Prasad)

L14Recur

5

Different Problems; Same pattern

Scheme Definition

```
(define (cart2 x y)
  (if (null? x) '()
      (append (couple (car x) y)
              (cart2 (cdr x) y))))
```

```
(define (couple a y)
  (if (null? y) '()
      (cons (list a (car y))
            (couple a (cdr y)))))
```

cs480(Prasad)

L14Recur

7

Alternate Syntax : lambda

```
(define cart2
  (lambda (x y)
    (if (null? x) '()
        (append (couple (car x) y)
                (cart2 (cdr x) y)))))
```

```
(define couple
  (lambda (a y)
    (if (null? y) '()
        (cons (list a (car y))
              (couple a (cdr y)))))
```

cs480(Prasad)

L14Recur

8

Powerset

```
> (powerset '(2))
( () (2) )

> (powerset '(1 2) )
( (1) (12) () (2) )

> (powerset '(0 1 2))
( (01) (012) (0) (02)
  (1) (12) () (2) )
```

Subsets of {0,1,2} with 0 are *equinumerous* with subsets of {0,1,2} without 0.

cs480(Prasad)

L14Recur

9

```
(define (powerset s)
  (if (null? s) '()
      (append (ins (car s)
                    (powerset (cdr s)))
              (powerset (cdr s)))
  ))

(define (ins a ss)
  (if (null? ss) '()
      (cons (cons a (car ss))
            (ins a (cdr ss)))
  ))
```

cs480(Prasad)

L14Recur

10

Alternate Syntax : let

```
(define (powerset s)
  (if (null? s) '()
      (let ( (aux (powerset (cdr s))) )
        (append (ins (car s) aux)
                aux)
      )
  ))

(define (ins a ss)
  (if (null? ss) '()
      (cons (cons a (car ss))
            (ins a (cdr ss)))
  ))
```

cs480(Prasad)

L14Recur

11

Related problems; Different Patterns

Remove-First-TopLevel

```
(define (rm-fst-top sym lis)
  (if (null? lis) '()
      (if (eq? sym (car lis)) (cdr lis)
          (cons (car lis)
                (rm-fst-top sym (cdr lis)))))
)
)
>(rm-fst-top 'a '(b (b a) a b a)) = (b (b a) b a)
• Linear recursion
```

cs480(Prasad)

L14Recur

13

Alternate Syntax : if => cond

```
(define (rm-fst-top sym lis)
  (cond ((null? lis) '())
        ((eq? sym (car lis)) (cdr lis))
        (else (cons (car lis)
                     (rm-fst-top sym (cdr lis)))))
)
)
>(rm-fst-top 'a '(b (b a) a b a)) = (b (b a) b a)
• Linear recursion
```

cs480(Prasad)

L14Recur

14

Remove-All-TopLevel

```
(define (rm-all-top sym lis)
  (cond ((null? lis) '())
        ((eq? sym (car lis)) (rm-all-top sym (cdr lis)))
        (else (cons (car lis)
                     (rm-all-top sym (cdr lis)))))
)
)
>(rm-all-top 'a '(b (b a) a b a)) = (b (b a) b)
>(rm-all-top '(b a) '(b (b a) a b a)) = (b (b a) a b a)
• Linear recursion
```

cs480(Prasad)

L14Recur

15

Remove-All-Expression-TopLevel

```
(define (rm-all-top exp lis)
  (cond ((null? lis) '())
        ((equal? exp (car lis)) (rm-all-top exp (cdr lis)))
        (else (cons (car lis)
                     (rm-all-top exp (cdr lis)))))
)
)
>(rm-all-top '(b a) '(b (b a) a b a)) = (b a b a)
• Linear recursion
```

cs480(Prasad)

L14Recur

16

Remove-All

```
(define (rm-all sym l1)
  (cond ((null? l1) '())
        ((symbol? (car l1))
         (if (eq? sym (car l1))
             (rm-all sym (cdr l1))
             (cons (car l1)
                   (rm-all sym (cdr l1)))))
        (else (cons (rm-all sym (car l1))
                    (rm-all sym (cdr l1)))))
  )
)
> (rm-all 'a '(b (b a) a b a)) = (b (b) b)
- Double recursion
```

cs480(Prasad)

L14Recur

17

rm-all : Structural recursion

- Empty list (Basis case)
(rm-all 'a '())
- First - Atom (Recursive case)
(rm-all 'a '(a b c a))
(rm-all 'a '(b b c a))
- First - Nested list (Recursive case)
(rm-all 'a '((a b c) d (a)))

cs480(Prasad)

L14Recur

18

Insertion sort (Note: creates a sorted copy)

```
(define (insert n lon)
  (cond ((null? lon) (list n))
        ((> n (car lon))
         (cons (car lon) (insert n (cdr lon))))
        (else (cons n lon)))
  )
)
(define (ins-sort lon)
  (if (null? lon) '()
      (insert (car lon) (ins-sort (cdr lon))))
  )
)
```

- Precondition: second arg to insert ordered.
- Postcondition: insert returns an ordered list.

cs480(Prasad)

L14Recur

19

Subset (uses OR and AND instead of IF)

```
(define (subset? ss s)
  (or (null? ss)
      (and (member (car ss) s)
            (subset? (cdr ss) s) ) )
  )
)
1=> (subset? '(a b c) '(A p 1 C 2 B q r))
#t
2=> (subset? '(a b c) '(p))
#f
```

cs480(Prasad)

L14Recur

20

Expression evaluation : A simple syntax directed translation

```
expr -> x | y | z  
expr -> (+ expr expr)  
expr -> (if expr expr expr)
```

Write a recursive definition for a function *ops* that counts the number of “+”s.

cs480(Prasad)

L14Recur

21

```
(define (ops e)  
; e is assumed to be a symbol or a list  
(cond  
  ((symbol? e) 0)  
  ((eq? (car e) '+)  
   (+ 1 (ops (cadr e))  
       (ops (caddr e))))  
  ((eq? (car e) 'if)  
   (+ (ops (cadr e))  
       (ops (caddr e))  
       (ops (caddr e))))  
  (else (display 'ILLEGAL))  
)  
)
```

cs480(Prasad)

L14Recur

22

Examples

```
(ops 'x)  
0  
(ops '(+ y z))  
1  
(ops '(if x (+ y z) (+ x z)))  
2
```

cs480(Prasad)

L14Recur

23