


# Inducción, Recursión y alcance



# Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

# Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

# Especificación inductiva de tipos de datos

- Tipo de datos:
  - Conjunto de valores
  - Conjunto de operaciones sobre dichos valores
- Especificación inductiva de tipos de datos
  - ejemplo: Sea  $S$  el conjunto más pequeño tal que:
    - (1)  $0 \in S$
    - (2) Si  $x \in S$  entonces  $x + 3 \in S$
- Especificación no inductiva
  - Argumento: 0 está en  $S$  por (1); por (2) 3 está en  $S$ , y 6 y 9, etc. así como  $M$ , el conjunto de múltiplos de 3. Entonces  $M \subseteq S$ , pero por definición  $S$  satisface (1) y (2) y  $S \subseteq M$ ; por lo tanto  $S = M$ .

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

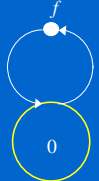
# Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

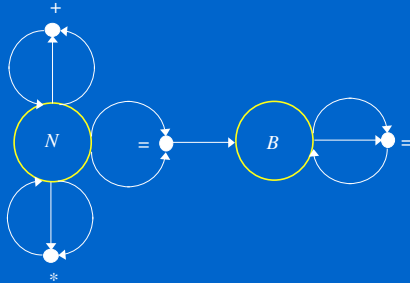
# Especificación inductiva de tipos de datos

- Especificación inductiva de tipos de datos
  - (1) Especificación de valores "iniciales" en  $S$
  - (2) Regla de producción



Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Especificación inductiva de tipos de datos



Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Especificación inductiva de tipos de datos

- Tipo de datos:
  - Conjunto de valores
  - Conjunto de operaciones sobre dichos valores
- Especificación inductiva de tipos de datos
  - Sea  $S$  el conjunto más pequeño tal que:
    - (1) ciertos valores están en  $S$
    - (2) Si  $x$  está en  $S$  entonces ciertos  $y$ 's están en  $S$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Especificación inductiva de tipos de datos

- Tipo de datos: lista-de-números (*list-num*)
  - (1) Lista vacía
  - (2) Si  $l$  es una lista y  $n$  es un número entonces  $(n . l)$  es una *list-num*
- Ejemplo:
  - $()$  es *list-num* por (1)
  - $(14 . ())$  es *list-num* por (2)
  - $(3 . (14 . ()))$  es *list-num*
  - $(-7 . (3 . (14 . ())))$  es *list-num*
  - Nada que no tenga este patrón es una lista de números
- En notación de lista:
  - $()$ ,  $(14)$ ,  $(3 14)$ ,  $(-7 3 14)$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - **Forma Backus-Naur (BNF) y derivaciones sintácticas**
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Forma Backus-Naur (BNF) y derivaciones sintácticas

- BNF: Sistema notacional
  - Para especificar tipos de datos o categorías sintácticas (no-terminales)
  - Para especificar la sintaxis de los lenguajes de programación
  - Especificación mediante *reglas de producción* o de *re-escritura*
- Sintaxis BNF
  - $\langle \text{tipo-de-dato} \rangle ::= \langle \text{tipo-de-dato} \rangle \mid \langle \text{tipo-de-dato} \rangle$
  - El lado derecho contiene símbolos sincategorámicos “(”, “)”, “.”
  - Kleen star form  $\{ \dots \}^*$ : repetir contenido (...) cero o más veces
  - Kleen plus form  $\{ \dots \}^+$ : repetir contenido (...) una o más veces

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Forma Backus-Naur (BNF) y derivaciones sintácticas

- Ejemplos:
  - $\langle \text{list-num} \rangle ::= ()$
  - $\langle \text{list-num} \rangle ::= (\langle \text{número} \rangle . \langle \text{list-num} \rangle)$
- Con barra:
  - $\langle \text{list-num} \rangle ::= () \mid (\langle \text{número} \rangle . \langle \text{list-num} \rangle)$
- Usando Kleen star form
  - $\langle \text{list-num} \rangle ::= (\{ \langle \text{número} \rangle \}^*)$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Derivaciones sintácticas

- Ejemplos:
  - `<list-num>`
  - $\Rightarrow$  `( <número> . <list-num> )`
  - $\Rightarrow$  `( 14 . <list-num> )`
  - $\Rightarrow$  `( 14 . ( ) )`
- La notación es declarativa (el orden no es significativo)
  - `<list-num>`
  - $\Rightarrow$  `( <número> . <list-num> )`
  - $\Rightarrow$  `( <número> . ( ) )`
  - $\Rightarrow$  `( 14 . ( ) )`

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF**
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Especificación de datos mediante BNF

- Especificación formal de tipos de datos en Scheme:
  - `<list> ::= ({ <datum> }*)`
  - `<dotted-datum> ::= ({ <datum> }+ . <datum>)`
  - `<vector> ::= #({ <datum> }*)`
  - `<datum> ::= <number> | <symbol> | <boolean> | <string> | <list> | <dotted-datum> | <vector>`

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Especificación de datos mediante BNF

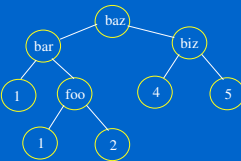
- Ejemplo: derivar `(#t (foo . ( ) ) 3)`
  - `<list> ::= (<datum> <datum> <datum>)`; Kleen star form
  - `<list> ::= (<boolean> <datum> <datum>)`
  - `<list> ::= (#t <datum> <datum>)`
  - `<list> ::= (#t <dotted-datum> <datum>)`
  - `<list> ::= (#t ({ <datum> }+ . <datum> ) <datum>)`
  - `<list> ::= (#t (<symbol> . <datum> ) <datum>)`
  - `<list> ::= (#t (foo . <datum> ) <datum>)`
  - `<list> ::= (#t (foo . <list> ) <datum>)`
  - `<list> ::= (#t (foo . ( ) ) <datum>)`
  - `<list> ::= (#t (foo . ( ) ) <number>)`
  - `<list> ::= (#t (foo . ( ) ) 3)`

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Otros tipos de datos

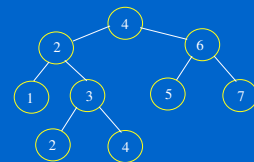
- Listas con un símbolo al frente y una lista similar
  - `<s-list> ::= ({ <symbol-expression> }*)`
  - `<symbol-expression> ::= <symbol> | <s-list>`
  - Ejemplos: `(a b c)`, `(an ((s-list)) (with lots) ((of) nestings))`
- Arboles binarios (con números en las hojas)
  - `<tree> ::= <number> | (<symbol> <tree> <tree>)`
  - Ejemplos
    - 1
    - (foo 1 2)
    - (bar 1 (foo 1 2))
    - (baz (bar 1 (foo 1 2)) (biz 4 5))

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.



## Sensitividad al contexto

- Arboles para búsqueda binaria
  - `<bin-search-tree> ::= ( ) | (<key> <bin-search-tree> <bin-search-tree>)`



- Restricciones en el contenido no pueden ser especificadas libre del contexto
- Se requiere una gramática más poderosa o anotaciones al margen

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - **Inducción**
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción

- La descripción inductiva de los tipos permite:
  - Probar teoremas acerca de los tipos de datos
  - Escribir programas para manipular dichos datos
- Ejemplo de prueba de teoremas acerca de los tipos de datos
  - Teorema: Sea  $s \in \langle \text{tree} \rangle$ ;  $s$  tiene un número non de nodos
  - Prueba por inducción en el tamaño de  $s$  (número de nodos)
  - Caso inicial:  $IH(0)$  ... No hay árboles con 0 nodos
  - Hipótesis inductiva: Si  $IH(k)$  entonces  $IH(k+1)$ 
    - $s$  tiene la forma  $(\text{sym } s1 \ s2)$  donde  $s1$  y  $s2$  son árboles
    - Si  $s$  tiene  $\leq k+1$  nodos  $s1$  y  $s2$  deben tener  $\leq k$  nodos
    - Supongamos  $s1$  y  $s2$  tienen  $2n_1+1$  y  $2n_2+1$  nodos respectivamente
    - Nodos en  $s$ :  $(2n_1+1) + (2n_2+1) + 1 = 2(n_1+n_2+1) + 1$ , que es non.

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- **Programas especificados recursivamente**
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- El uso de BNF permite definir tipos de datos compuestos a partir de los datos más simples
- La misma idea puede aplicarse para la construcción de procedimientos que manipulen dichos conjuntos
- El procedimiento consiste en definir:
  - La funcionalidad asociada a la parte más simple del procedimiento
  - Reutilizar dicha funcionalidad para definir conductas más complejas

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Definir la función exponencial  $e(n, x) = x^n$  ( $n \geq 0, x \neq 0$ )

$$e_0(x) = 1$$

$$e_1(x) = x * e_0(x)$$

$$e_2(x) = x * e_1(x)$$

$$e_3(x) = x * e_2(x)$$

En general:

$$e_n(x) = x * e_{n-1}(x)$$

Decurriendo:

$$\text{Si } n = 0, e(n, x) = 1; \quad x^0 = 1$$

$$\text{Sino } e(n, x) = x * e(n-1, x); \quad x^n = x * x^{n-1}$$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Probar por inducción:
  - (1) caso base: Si  $n = 0, e(0, x) = x^0 = 1$
  - (2) paso inductivo: suponer que el procedimiento trabaja para  $n = k$   
 $e(k, x) = x^k$  para cualquier  $k$  entonces debe ser cierto que  $e(k+1, x) = x^{k+1}$
  - Calculemos:  
 $e(k+1, x) = x * e(k, x) = x * x^k = x^{k+1}$
- Expresado en Scheme:

```
(define e
  (lambda (n x)
    (if (zero? n)
        1
        (* x (e (- n 1) x))))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- La estructura del programa sigue la estructura de los datos
  - Definir el predicado *list-of-numbers?*
  - Definición de list-num:  $\langle \text{list-num} \rangle ::= () \mid \langle \text{number} \rangle . \langle \text{list-num} \rangle$

```
(define list-of-numbers?  
  (lambda (lst)  
    (if (null? lst)  
        #t  
        (if (pair? lst)  
            (if (number? (car lst))  
                (list-of-numbers (cdr lst))  
                #f)  
            #f))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Construcción del programa siguiendo la estructura de datos:
  - Definición de list-num:  $\langle \text{list-num} \rangle ::= () \mid \langle \text{number} \rangle . \langle \text{list-num} \rangle$

```
(define list-of-numbers?  
  (lambda (lst)  
    (if (null? lst)  
        #t  
        ...))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Construcción del programa siguiendo la estructura de datos:
  - Definición de list-num:  $\langle \text{list-num} \rangle ::= () \mid \langle \text{number} \rangle . \langle \text{list-num} \rangle$

```
(define list-of-numbers?  
  (lambda (lst)  
    (if (null? lst)  
        #t  
        (if (pair? lst)  
            ...  
            #f))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Definición del procedimiento principal:
  - La operación sobre la lista
  - Definición de list-num:  $\langle \text{list-num} \rangle ::= () \mid \langle \text{number} \rangle . \langle \text{list-num} \rangle$

```
(define list-of-numbers?  
  (lambda (lst)  
    (if (null? lst)  
        #t  
        (if (pair? lst)  
            (if (number? (car lst))  
                (list-of-numbers (cdr lst))  
                #f)  
            #f))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Prueba por inducción:
  - Por inducción en la longitud de la lista
  - Definición de list-num:  $\langle \text{list-num} \rangle ::= () \mid \langle \text{number} \rangle . \langle \text{list-num} \rangle$

```
(define list-of-numbers?  
  (lambda (lst)  
    (if (null? lst)  
        #t  
        (if (pair? lst)  
            (if (number? (car lst))  
                (list-of-numbers? (cdr lst))  
                #f)  
            #f))))
```

Prueba:

(1) Trabaja para listas vacías y para objetos que no son listas

(2) Si trabaja para listas de long.  $k$ , también para listas de long.  $k + 1$ : Si long. de *lst* es  $k + 1$ , long. *cdr lst* es  $k$ ; por lo tanto, podemos determinar la membresía en la lista de long.  $k$  con el *cdr* de *lst*.

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 2: obtiene el  $n$ ésimo elemento de una lista  
 $> (\text{nth-elt } '(\text{a b c}) 1)$   
 $> \text{b}$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 2: obtiene el enésimo elemento de una lista

```
> (define nth-elt?
  (lambda (lst n)
    (if (null? lst)
        ("error: la lista es nula")
        ...
    )))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 2: obtiene el enésimo elemento de una lista

```
> (define nth-elt?
  (lambda (lst n)
    (if (null? lst)
        ("error: la lista es nula")
        (if (zero? n)
            (car lst)
            ...
        ))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 2: obtiene el enésimo elemento de una lista

```
> (define nth-elt?
  (lambda (lst n)
    (if (null? lst)
        ("error: la lista es nula")
        (if (zero? n)
            (car lst)
            (nth-elt (cdr lst) (- n 1))))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 2: obtiene el enésimo elemento de una lista

```
> (define nth-elt?
  (lambda (lst n)
    (if (null? lst)
        ("error: la lista es nula")
        (if (zero? n)
            (car lst)
            (nth-elt (cdr lst) (- n 1))))))

> (nth-elt '(a b c) 1)
> b
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 3: obtiene la longitud de una lista

```
> (define list-length
  (lambda (lst)
    (if (null? lst)
        0
        ... )))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 3: obtiene la longitud de una lista

```
> (define list-length
  (lambda (lst)
    (if (null? lst)
        0
        (+ 1 (list-length (cdr lst))))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 3: obtiene la longitud de una lista

```
> (define list-length
  (lambda (lst)
    (if (null? lst)
        0
        (+ 1 (list-length (cdr lst))))))

> (list-length '(a b c))
> 3
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 3: obtiene la longitud de una lista

```
> (define list-length
  (lambda (lst)
    (if (null? lst)
        0
        (+ 1 (list-length (cdr lst))))))

> (list-length '(a b c))
> 3
> (list-length '((x) ( )))
> 2
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Programas especificados recursivamente

- Ejemplo 3: obtiene la longitud de una lista

```
> (define list-length
  (lambda (lst)
    (if (null? lst)
        0
        (+ 1 (list-length (cdr lst))))))

> (list-length '(a b c))
> 3
> (list-length '((x) ( )))
> 2
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

### Ejemplo 1: remove-first Remueve la primera ocurrencia de un símbolo en una lista

```
> (remove-first 'a '(a b c))
> (b c)
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

### Ejemplo 1: remove-first Remueve la primera ocurrencia de un símbolo en una lista

```
> (define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        ...)))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 1: remove-first  
Remueve la primera ocurrencia de un símbolo en una lista

```
> (define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            ... ))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 1: remove-first  
Remueve la primera ocurrencia de un símbolo en una lista

```
> (define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            (cdr los)
            ... ))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 1: remove-first  
Remueve la primera ocurrencia de un símbolo en una lista

```
> (define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            (cdr los)
            (cons (car los) (remove-first s (cdr los) ))))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 1: remove-first  
Remueve la primera ocurrencia de un símbolo en una lista

```
> (define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            (cdr los)
            (cons (car los) (remove-first s (cdr los) ))))))

> (remove-first 'a '(a b c))
> (b c)
> (remove-first 'b '(e f g))
> (e f g)
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 2: remove  
Remueve todas las ocurrencias de un símbolo en una lista

```
> (remove 'a '(c1 a d1 a))
> (c1 d1)
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 2: remove  
Remueve todas las ocurrencias de un símbolo en una lista

```
> (define remove
  (lambda (s los)
    (if (null? los)
        '()
        ... )))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.



Ejemplo 2: remove  
Remueve todas las ocurrencias de un símbolo en una lista

```
> (define remove
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            ... ))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 2: remove  
Remueve todas las ocurrencias de un símbolo en una lista

```
> (define remove
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            (remove s (cdr los))
            ... ))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 2: remove  
Remueve todas las ocurrencias de un símbolo en una lista

```
> (define remove
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            (remove s (cdr los))
            (cons (car los) (remove s (cdr los) ))))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 2: remove  
Remueve todas las ocurrencias de un símbolo en una lista

```
> (define remove
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? (car los) s)
            (remove s (cdr los))
            (cons (car los) (remove s (cdr los) ))))))

> (remove 'a '(c1 a d1 a))
> (c1 d1)
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 1)  
Substituye todas las ocurrencias de un símbolo en una lista

```
> (subst 'a 'b '(b c) (b d))
> ((a c) (a d))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 1)  
Substituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        ... )))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 1)  
 Substituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (if (symbol? (car slst))
            (... )
            (... )))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 1)  
 Substituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (if (symbol? (car slst))
            (if (eq? (car slst) old)
                (cons new (subst new old (cdr slst)))
                (... ))
            (... )))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 1)  
 Substituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (if (symbol? (car slst))
            (if (eq? (car slst) old)
                (cons new (subst new old (cdr slst)))
                (cons (car slst) (subst new old (cdr slst))))
            (... )))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 1)  
 Substituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (if (symbol? (car slst))
            (if (eq? (car slst) old)
                (cons new (subst new old (cdr slst)))
                (cons (car slst) (subst new old (cdr slst))))
            (cons (subst new old (car slst))
                  (subst new old (cdr slst)))))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 1)  
 Substituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (if (symbol? (car slst))
            (if (eq? (car slst) old)
                (cons new (subst new old (cdr slst)))
                (cons (car slst) (subst new old (cdr slst))))
            (cons (subst new old (car slst))
                  (subst new old (cdr slst)))))))

> (subst 'a 'b '((b c) (b d)))
> ((a c) (a d))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 2)  
 Substituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (cons (subst-symbol-exp new old (car slst))
              (subst new old (cdr slst))))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 2)  
 Sustituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (cons (subst-symbol-exp new old (car slst))
              (subst new old (cdr slst))))))

> (define subst-symbol-exp
  (lambda (new old se)
    (if (symbol? se)
        (...)
        (...))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 2)  
 Sustituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (cons (subst-symbol-exp new old (car slst))
              (subst new old (cdr slst))))))

> (define subst-symbol-exp
  (lambda (new old se)
    (if (symbol? se)
        (if (eq? se old) new se)
        (...))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 2)  
 Sustituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (cons (subst-symbol-exp new old (car slst))
              (subst new old (cdr slst))))))

> (define subst-symbol-exp
  (lambda (new old se)
    (if (symbol? se)
        (if (eq? se old) new se)
        (subst new old se))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Ejemplo 3: subst (versión 2)  
 Sustituye todas las ocurrencias de un símbolo en una lista

```
> (define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (cons (subst-symbol-exp new old (car slst))
              (subst new old (cdr slst))))))

> (define subst-symbol-exp
  (lambda (new old se)
    (if (symbol? se)
        (if (eq? se old) new se)
        (subst new old se))))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- **Propiedades estáticas de variables**
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Propiedades estáticas de variables

- Propiedades Estáticas:
  - pueden ser descubiertas mediante el análisis del texto de un programa (útiles para los compiladores)
- Propiedades dinámicas:
  - Se tienen que determinar a tiempo de ejecución
- En Scheme la relación entre un parámetro formal y la referencia de la variable co-referida es estática

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- El Cálculo Lambda ( $\lambda$ -calculus)
  - Lenguaje abstracto cuyas expresiones denotan funciones
  - La base formal de la teoría de los lenguajes de programación
- Sintaxis:
  - $\langle \text{exp} \rangle ::= \langle \text{varref} \rangle$ 
    - |  $(\text{lambda } \langle \text{var} \rangle \langle \text{exp} \rangle)$
    - |  $\langle \text{exp} \rangle \langle \text{exp} \rangle$
- Notación abstracta:
  - $\lambda \langle \text{var} \rangle . \langle \text{exp} \rangle$
  - ie.  $\lambda x . x$  ; función identidad
  - aplicación:  $((\lambda x . x) 9) \Rightarrow 9$
  - aplicación:  $((\lambda x . x) (\lambda x . x)) \Rightarrow (\lambda x . x)$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- Variable ligada (bound)
  - Si se refiere a un parámetro formal introducido en la expresión.
  - Si la variable no está ligada es ocurre libre (*free*)
  - Es un error no tener una variable ligada a tiempo de ejecución
- Ejemplos
  - $((\text{lambda } (x) x) y) \Rightarrow$  "y" es libre
  - $((\text{lambda } (y) ((\text{lambda } (x) x) y)) \Rightarrow$  "y" ya está ligada (embebida)
  - Se liga primero el contexto externo, pero se evalúa primero el interno
- Liga lexica: Si está ligada por un parámetro formal en el procedimiento
- Liga global:
  - Ligada al nivel del intérprete mediante **define**
  - Ligada por el sistema (símbolos de función)

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- Combinators
  - Expresiones lambda sin variables libres
  - Tienen un significado fijo, independiente del valor que tomen las variables
- Ejemplos:
  - Función identidad:  $(\text{lambda } (x) x)$ 
    - >  $((\text{lambda } (x) x) 3)$
    - > 3
  - Aplica una función a un argumento:  $(\text{lambda } (f) (\text{lambda } (x) (f x)))$ 
    - >  $((\text{lambda } (f) (\text{lambda } (x) (f x))) 'car)$
    - >  $(\text{lambda } (x) (\text{car } x))$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- Combinators
  - Expresiones lambda sin variables libres
  - Tienen un significado fijo, independiente del valor que tomen las variables
- Ejemplos:
  - Función identidad:  $(\text{lambda } (x) x)$ 
    - >  $((\text{lambda } (x) x) 3)$
    - > 3
  - Aplica una función a un argumento:  $(\text{lambda } (f) (\text{lambda } (x) (f x)))$ 
    - >  $((\text{lambda } (f) (\text{lambda } (x) (f x))) 'car)$
    - >  $(\text{lambda } (x) (\text{car } x))$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- Una variable  $x$  ocurre libre en una expresión  $E$  si y solo si:
  - $E$  es una variable y  $E$  es  $x$
  - $E$  es de forma  $(E_1 E_2)$  y  $x$  ocurre libre en  $E_1$  o  $E_2$
  - $E$  es de forma  $(\text{lambda } (y) E')$ , donde  $y$  es diferente de  $x$  y  $x$  ocurre libre en  $E'$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- Una variable  $x$  ocurre libre en una expresión  $E$  si y solo si:
  - $E$  es una variable y  $E$  es  $x$
  - $E$  es de forma  $(E_1 E_2)$  y  $x$  ocurre libre en  $E_1$  o  $E_2$
  - $E$  es de forma  $(\text{lambda } (y) E')$ , donde  $y$  es diferente de  $x$  y  $x$  ocurre libre en  $E'$

- Ejemplo: ocurre  $x$  libre?

```
(lambda (x)
  ((lambda (x) (cons x ' ( )))
   (cons x ' ( )))
 )
```

Ocurre libre en  $E'$  pero no en  $E$ !

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- Una variable  $x$  ocurre ligada en la expresión  $E$  si y solo si:
  - $E$  es de forma  $(E_1 E_2)$  y  $x$  ocurre ligada en  $E_1$  o  $E_2$
  - $E$  es de forma  $(\text{lambda } (y) E')$ , donde  $x$  ocurre ligada en  $E'$  o  $x$  y  $y$  son la misma variable y ocurre libre en  $E'$
  - Ninguna variable está ligada en una expresión consistente de una sólo variable

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Variables libres y ligadas

- Una variable  $x$  ocurre ligada en la expresión  $E$  si y solo si:
  - $E$  es de forma  $(E_1 E_2)$  y  $x$  ocurre ligada en  $E_1$  o  $E_2$
  - $E$  es de forma  $(\text{lambda } (y) E')$ , donde  $x$  ocurre ligada en  $E'$  o  $x$  y  $y$  son la misma variable y ocurre libre en  $E'$
  - Ninguna variable está ligada en una expresión consistente de una sólo variable

- Ejemplo: ocurre  $x$  ligada?

```
(lambda (x)
  ((lambda (x) (cons x ' ( )))
   (cons x ' ( )))
 )
```

Ocurre ligada en  $E$ , pero también libre y ligada en  $E'$ !

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Alcance y dirección léxica

- Alcance (scope)
  - Región asociada a la declaración de una variable
  - La región de una declaración al nivel más alto es todo el programa
  - Alcance (scope): texto en el cual las referencias de las variables se refieren a la declaración
  - El alcance y la región pueden ser lo mismo, pero el alcance puede no incluir toda la región
  - En Scheme el alcance se determina estáticamente (lexical scope)

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Alcance y dirección léxica

- Región: la región de un parámetro formal es el cuerpo de la expresión lambda.

```
> (define add
  (lambda (x)
    (+ x 3)))
```

- Parámetro formal:  $x$
- Región:  $(+ x 3)$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Alcance y dirección léxica

- Hoyos del "alcance"
 

```
> (define x
  (lambda (x)
    (map (lambda (x) (+ x 1)) x)))
> (x '(1 2 3))
> (2 3 4)
```

  - x: loop de lectura y evaluación (incluye el cuerpo del define)
  - x: **alcance con hoyo (el map)**
  - x: cuerpo del map

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Alcance y dirección léxica

- Islas de alcance (hoyos)
 

```
> (define x
  (lambda (x)
    (map (lambda (x) (+ x 1)) x)))
> (x '(1 2 3))
> (2 3 4)
```

  - Alcance: La declaración de la variable *v* tiene como alcance la región que incluye todas las referencias a *v* que ocurren **libres** en la región asociada con la declaración
  - Existe un algoritmo para encontrar a qué declaración se refiere una variable: buscar desde dentro hacia fuera por un parámetro que ligue a la variable. Si no existe, la variable es libre.

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Alcance y dirección léxica

- Contornos: fronteras de una región.
- Profundidad léxica (lexical or static depth): Número de contornos cruzados (de adentro hacia fuera) para encontrar la declaración de una variable.
- Ejemplo:
 

```
> (lambda (x y)
  ((lambda (a) (x (a y)))
   x))
```

  - Profundidad léxica de *x*, *a*: 0
  - Profundidad léxica de *x*, *y*: 1
- Declaración de una variable:
  - profundidad léxica, número de argumento  $\Rightarrow (v : prof., pos.)$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Alcance y dirección léxica

- Dos notaciones:
  - Normal:
 

```
(lambda (x y)
  ((lambda (a) (x (a y)))
   x))
```
  - Indicando la profundidad léxica y la posición
 

```
(lambda (x y)
  ((lambda (a)
    ((x: 1 0) ((a: 0 0) (y: 1 1))))
   (x: 0 0)))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Alcance y dirección léxica

- Una notación más:
  - Indicando la profundidad léxica y la posición
 

```
(lambda (x y)
  ((lambda (a)
    ((x: 1 0) ((a: 0 0) (y: 1 1))))
   (x: 0 0)))
```
  - Indicando la profundidad léxica y la posición
 

```
(lambda 1
  ((lambda 2
    ((: 1 0) (: 0 0) (: 1 1))))
   (: 0 0)))
```

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Inducción, Recursión y Alcance

- Especificación inductiva de tipos de datos
  - Especificación inductiva
  - Forma Backus-Naur (BNF) y derivaciones sintácticas
  - Especificación de datos mediante BNF
  - Inducción
- Programas especificados recursivamente
  - Derivación de programas a partir de especificación de datos con BNF
  - Tres ejemplos
- Propiedades estáticas de variables
  - Variables libres y ligadas
  - Alcance y dirección léxica
  - Renombre de variables**

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Renombre de variables

- La conducta de los procedimientos es independiente de los nombres de las variables
- Regla de transformación de programas:
  - El significado de un programa no se altera si se cambian los nombres de las referencias a sus parámetros.
  - El nuevo nombre no debe entrar en conflicto con otros nombres
  - Este conflicto ocurre si el nuevo nombre *ocurre libre* en la expresión original:
    - (lambda (x) (cons x '( )))
    - (lambda (cons) (cons cons '( )))
  - En este caso la nueva liga de *cons* captura la referencia a otra liga de *cons*

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Renombre de variables

- Regla de transformación de programas:
  - Las referencias en hoyos de alcance no se deben cambiar  
(lambda (x)  
  (lambda (x) (cons x '( )))  
  (cons x '( )))
  - Se puede transformar a:  
(lambda (y)  
  (lambda (x) (cons x '( )))  
  (cons y '( )))
  - Pero no:  
(lambda (y)  
  (lambda (x) (cons y '( )))  
  (cons y '( )))

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

## Renombre de variables

- El resultado de substituir una expresión  $y$  por todas las *ocurrencias libres* de una variable  $x$  en una expresión  $exp$  se escribe:  
 $exp[y/x]$  (substituye  $y$  en vez de  $x$  en  $exp$ )
- Conversión  $-\alpha$ :
  - Regla para renombrar expresiones lambda con un parámetro- $\alpha$  ( $\alpha$ -conversion):  
(lambda (var) exp) = (lambda (var') exp[var'/var])  
donde  $var'$  es cualquier variable que no ocurre libre en  $exp$

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

# Fin

