

# Fundamentos de lenguajes de programación

## La relación entre Inducción y Programación

EISC. Facultad de Ingeniería. Universidad del Valle

Junio de 2020

# Contenido

- 1 Especificación Recursiva de datos
  - Especificación inductiva
  - Especificación mediante gramáticas en forma BNF
- 2 Especificación recursiva de programas
- 3 Los conceptos de Alcance y ligadura de una variable

# Contenido

- 1 Especificación Recursiva de datos
  - Especificación inductiva
  - Especificación mediante gramáticas en forma BNF
- 2 Especificación recursiva de programas
- 3 Los conceptos de Alcance y ligadura de una variable

# Especificación Recursiva de datos

contrato  
↙

- Cuando se escribe un procedimiento, se debe definir que clase de valores se espera como entrada y como salida.
- Ejemplo, la función suma tiene como entrada dos números naturales y tiene como salida un número natural.
- Los datos en las funciones recursivas, pueden tener también definiciones recursivas que faciliten la programación.

# Especificación Recursiva de datos

## Técnicas

Existe dos técnicas para la definición recursiva de datos:

- 1 Especificación inductiva
- 2 Especificación mediante gramáticas.

# Especificación inductiva

## Definición

Se define un conjunto  $S$ , el cual es el conjunto más pequeño que satisface las siguientes dos propiedades:

- 1 Algunos valores específicos que deben estar en  $S$ .
- 2 Si algunos valores están en  $S$ , entonces otros valores también están en  $S$ .

# Especificación inductiva

## Números pares

- 1 Si  $n = 2$  entonces  $n$  es par
- 2 Si  $n$  es par, entonces  $n + 2$  también es par.

## Lista de números

- 1 *empty* es una lista de números
- 2 Si  $n$  es un número y  $l$  es una lista entonces  $(n \ l)$  es una lista de números

# Especificación inductiva

## Especificación formal

Ahora formalmente:

## Números pares

1  $2 \in S$

2  $\frac{n \in S}{(n+2) \in S}$

## Lista de números

→ 1  $() \in S$

2  $\frac{l \in S, n \in N}{(n\ l) \in S}$





# Especificación inductiva

## Ejemplo

Demuestre que  $(1(2(3()))))$  es una lista de números.

## Solución

$$1 \quad 1 \in \mathbb{N}, \underline{(2(3()))} \in S$$

$$2 \quad (1(2(3())))) \in S$$

Se puede seguir hasta llegar al caso fundamental  $() \in S$

$$\frac{2 \in \mathbb{N}, (3()) \in S}{(2(3())) \in S}$$

$$\frac{3 \in \mathbb{N}, \boxed{()} \in S}{(3()) \in S}$$

# Especificación inductiva

## Especificación formal

Ahora realicemos la especificación inductiva de:

- 1 Una lista de número pares
- 2 Múltiplos de 5

$$5 \in S$$

$$\frac{n \in S}{n+5 \in S}$$

$$() \in S$$

$$\frac{1 \in S, x \in P}{(x \ 1) \in S}$$

$$2 \in P$$

$$\frac{n \in P}{n+2 \in P}$$

} Esp números pares

## 1) Listas de números múltiplos de 5

$$\begin{array}{l} (c) \in S \\ x \in \mathbb{I}, l \in S \\ \hline (x+1) \in S \end{array}$$

$$5 \in \mathbb{I} \leftarrow$$

$$\begin{array}{l} n \in \mathbb{I} \\ \hline n+5 \in \mathbb{I} \end{array}$$

## 2) Números impares

$$\begin{array}{l} 1 \in S \\ n \in S \\ \hline n+2 \in S \end{array}$$

1, 3, 5, ...

## 3) Listas que contengan parejas, cuyo primer elemento es par y el segundo múltiplo de 7.

$$\begin{array}{l} (c) \in S \\ a \in P, b \in W, l \in S \\ \hline ((a, b) \mid l) \in S \end{array}$$

$$\begin{array}{l} \text{Pares} \\ 2 \in P \\ x \in P \\ \hline x+2 \in P \end{array}$$

$$\begin{array}{l} \text{Múltiplos 7} \\ 7 \in W \\ y \in W \\ \hline y+7 \in W \end{array}$$

# Especificación inductiva

## Lista de números pares

1  $2 \in P$

2  $\frac{n \in S}{(n+2) \in P}$

Corregir

1  $() \in S$

2  $\frac{l \in S, x \in P}{(x \ l) \in S}$

## Múltiplos de 5

1  $5 \in S$

2  $\frac{n \in S}{(n+5) \in S}$

# Especificación inductiva

## Ejercicios

Indique que tipo de conjuntos están definidos por las siguiente reglas:

- 1  $(0, 1) \in S, \frac{(n,k) \in S}{(n+1, k+7)} \in S$   $(0, 1) \rightarrow (1, 8) \rightarrow (2, 15) \rightarrow (3, 23)$
- 2  $(0, 1) \in S, \frac{(n,k) \in S}{(n+1, 2k)} \in S$   $(0, 1) \rightarrow (1, 2) \rightarrow (2, 4) \rightarrow (3, 8)$
- \* 3  $(0, 1, 5) \in S, \frac{(i,j,k) \in S}{(i+1, j+2, i+j)} \in S$   $(0, 1, 5) \rightarrow (1, 3, 4) \rightarrow (2, 5, 4) \times$

Para cada una de las especificaciones dé dos ejemplos numéricos que las cumplan.

$$V \rightarrow V+2 \rightarrow V+2+2 \rightarrow V+2+2+2 \rightarrow$$

5

100

# Especificación mediante gramáticas

- Una forma sencilla de especificar datos recursivos es con gramáticas regulares en forma Backus-Nour.
- Las gramáticas se componen de:
  - 1 Símbolos no terminales, que son aquellos que se componen de otros símbolos, son conocidos como categorías sintácticas
  - 2 Símbolos terminales: corresponden a elementos del alfabeto
  - 3 Reglas de producción

# Especificación mediante gramáticas

- Alfabeto: Conjunto de símbolos, ejemplo  $\Sigma = \{a, b, c, \dots\}$
- Reglas de producción: Construcción del lenguaje:
  - Cerradura de Kleene:  $\{a\}^* = \{\epsilon, \{a\}, \{a, a\}, \{a, a, a\} \dots\}$
  - Cerradura positiva:  $\{b\}^+ = \{\{b\}, \{b, b\}, \{b, b, b\} \dots\}$

# Especificación mediante gramáticas

## Lista números

$\langle \text{lista-de-enteros} \rangle ::= ()$   
 $\quad ::= (\langle \text{int} \rangle \langle \text{lista-de-enteros} \rangle)$

$\langle \text{lista-de-enteros} \rangle ::= () | (\langle \text{int} \rangle \langle \text{lista-de-enteros} \rangle)$

$\langle \text{lista-de-enteros} \rangle ::= (\langle \text{int} \rangle)^*$



# Especificación mediante gramáticas

## Árbol Binario

$\langle \text{arbol-binario} \rangle ::= \langle \text{int} \rangle$   
 $\quad ::= (\langle \text{simbolo} \rangle \langle \text{arbol-binario} \rangle \langle \text{arbol-binario} \rangle)$

Ejemplos de árboles binarios son:

→ 1

→ (foo 1 2)

→ (foo 1 (bar 2 3) 3)

Simb arb 916

Simb (simb num num) num

# Especificación mediante gramáticas

## Expresión calculo $\lambda$

$\langle \text{lambda-exp} \rangle ::= \langle \text{identificador} \rangle$   
 $::= (\text{lambda } (\langle \text{identificador} \rangle) \langle \text{lambda-exp} \rangle)$   
 $::= (\langle \text{lambda-exp} \rangle \langle \text{lambda-exp} \rangle)$

$\langle \text{identificador} \rangle ::= \langle \text{letra} \rangle^+$

Ejemplo de cálculo  $\lambda$ :

'(lambda (x) (x (y)))  
'((lambda (y) (z y)) x)  
'(x y)  
'x

( ( lambda (y) (z y) ) x )

# Contenido

- 1 Especificación Recursiva de datos
  - Especificación inductiva
  - Especificación mediante gramáticas en forma BNF
- 2 Especificación recursiva de programas
- 3 Los conceptos de Alcance y ligadura de una variable

# Especificación recursiva de programas

- La definición inductiva o mediante gramáticas de los conjuntos de datos sirve de guía para desarrollar procedimientos que operan sobre dichos datos
- La estructura de los programas debe seguir la estructura de los datos
- Para esto realizamos especificación recursiva de programas, la idea es utilizar el principio del subproblema más pequeño.

# Especificación recursiva de programas

## Ejemplo

Una función estándar de Dr Racket es `list-length`, la cual nos retorna el tamaño de una lista. Para el diseño de esta función debemos retornar a la especificación recursiva de las listas

$\text{Lista} ::= () \mid (\text{numero } \text{Lista})$

Se debe considerar entonces el caso base de la lista vacía, en el cual retornamos 0.

# Especificación recursiva de programas

## Ejemplo

De acuerdo a esto el diseño de list-length es:

```
;;list-length: lista -> numero
(define list-length
  (lambda (lst)
    (if (null? lst)
        0
        ...
    )
  )
)
```

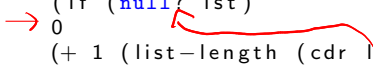
Tomando en cuenta la segunda parte de la definición, se debe sumar 1 al tamaño si no encontramos la lista vacía.

# Especificación recursiva de programas

## Ejemplo

Por lo tanto la función list-length es:

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



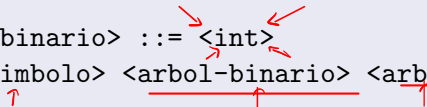
De acuerdo a la especificación recursiva de listas ¿Que se puede decir de este diseño?

# Especificación Recursiva de programas

## Un árbol binario

Recordando la definición de los árboles vista anteriormente:

```
<arbol-binario> ::= <int>  
                ::= (<simbolo> <arbol-binario> <arbol-binario>)
```

The diagram shows the recursive definition of a binary tree. Red arrows point from the recursive calls in the second line to the corresponding symbols in the first line. Specifically, an arrow points from the first '<arbol-binario>' in the second line to '<int>' in the first line. Another arrow points from the second '<arbol-binario>' in the second line to '<int>' in the first line. A third arrow points from the third '<arbol-binario>' in the second line to '<int>' in the first line. The symbols '<simbolo>', '<arbol-binario>', and '<arbol-binario>' in the second line are underlined.

Este árbol será representado así:

```
(define arbolA '(k (h 5 3) (t (s 10 11) 12)))  
(define arbolB 2)
```

En Dr Racket el operador  $( \dots )$  va generar una lista, toda palabra será convertida en símbolo y todo  $( \dots )$  será convertido en lista.



# Especificación Recursiva de programas

## Un árbol binario

Procedimiento sum-arbol:

```
(define sum-arbol
  (lambda (arbol)
    (if (number? arbol)
        arbol
        (+ (sum-arbol (cadr arbol))
           (sum-arbol (caddr arbol))
          )
      )
  )
)
```

# Especificación Recursiva de programas

## Lista números

Procedimiento para generar una lista de números a partir de un árbol

```
;;BNF
;;Entrada: <arbol-binario> ::= <int> | <simbolo> <
    arbol-binario> <arbol-binario>
;;Salida: <lista-numeros> ::= '() | <int> <lista-numeros>
(define arbol->lista
  (lambda (l)
    (if (number? l) ← condición de parada
        (cons l empty)
        (append
         (arbol->lista (cadr l)) ← Hija
         (arbol->lista (caddr l)) ← Hijo
        )
    )
  )
)
```

# Especificación recursiva de programas

## Resumen

En el diseño de programas para datos recursivos tenga en cuenta:

- 1 Identifique el caso terminal (base) o donde termina la especificación recursiva
- 2 La idea es pasar de estados no terminales a uno terminal
- 3 La estrategia es llamar recursivamente la función desde un estado no terminal para llegar a uno terminal

# Especificación recursiva de programas

## Ejercicio

Tomando en cuenta la especificación recursiva de listas, diseñe funciones:

- 1 nth-element. (nth-element '(4 5 6) 2) retorna 5.
- 2 remove-first (remove-first '(1 2 3)) retorna '(2 3)

La especificación mediante gramáticas de listas de números es:

`Lista ::= () | (numero Lista)`

# Contenido

- 1 Especificación Recursiva de datos
  - Especificación inductiva
  - Especificación mediante gramáticas en forma BNF
- 2 Especificación recursiva de programas
- 3 Los conceptos de Alcance y ligadura de una variable

# Los conceptos de Alcance y Ligadura de una variable

- El concepto de variable es fundamental en los lenguajes de programación
- Una variable puede ser declarada y posteriormente ligada o referenciada

- Declaración:

```
(lambda (x) ...)  
(let ((x ...)) ...)
```

- Referencia:

```
x ;; Como valor  
(f x y) ;; Como valor procedimiento
```

# Los conceptos de Alcance y Ligadura de una variable

- Una variable esta ligada al lugar donde se declara
- El valor ligado o referenciado por la variable es su denotación
- Cada lenguaje de programación tiene asociadas unas reglas de ligadura que determinan a qué declaración hace referencia cada variable
- Dependiendo del momento de aplicación de las reglas antes o durante la ejecución, los lenguajes se denominan de alcance estático o alcance dinámico

# Los conceptos de Alcance y Ligadura de una variable

x

- Si la expresión  $e$  es una variable, la variable  $x$  ocurre libre iff  $x$  es igual a  $e$ .
- Si la expresión  $e$  es de la forma  $(\lambda (y) e')$  entonces la variable  $x$  ocurre libre iff  $y$  es distinto que  $x$  y  $x$  ocurre libre en  $e'$ .
- Si la expresión  $e$  es de la forma  $(e_1, e_2)$ , entonces  $x$  ocurre libre iff si  $x$  ocurre libre en  $e_1$  o  $e_2$ .



# Los conceptos de Alcance y Ligadura de una variable

Dada la definición anterior, indique en las siguiente expresiones si  $x$  ocurre libre o no.

■  $'(\text{lambda } (x) (\text{lambda } (y) (x)))$

$L. y, d, x$

■  $'((\text{lambda } (z) (\text{lambda } (y) x)) (x))$

■  $'((\text{lambda } (y) (\text{lambda } (y) x)) (\text{lambda } (z) x) x)$

$(\rho_1 \ \rho_2)$   
 $L \vee L = L_{lib}$   
 $L_{lib}$

$\rho_1$   
 $\rho_2$   
 $L_{lib}$

$\rho_2$   
 $\rho_1 = L_{lib}$   
 $\rho_2 = L_{lib}$

# Los conceptos de Alcance y Ligadura de una variable

Para examinar si  $x$  ocurre libre o no en una expresión, por ejemplo:

```
'(lambda (x) (lambda (y) z))
```

Para el diseño debemos considerar la gramática del calculo  $\lambda$

```
<lambda-exp> ::= <identificador>  
::= (lambda (<identificador>) <lambda-exp>)  
::= (<lambda-exp> <lambda-exp>)
```

```
<identificador> ::= <letra>+
```

# Los conceptos de Alcance y Ligadura de una variable

## Determinar si una variable ocurre libre

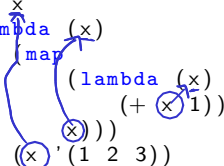
De esta forma se diseña un procedimiento que evalúa si **var** ocurre libre en **exp**.

```
(define occurs-free?  
  (lambda (var exp)  
    (cond  
      ((symbol? exp) (eqv? var exp))  
      ((eqv? (car exp) 'lambda)  
       (and (not (eqv? (caadr exp) var))  
            (occurs-free? var (caddr exp))))  
      (else (or (occurs-free? var (car exp))  
                (occurs-free? var (cadr exp)))))))
```

# Los conceptos de Alcance y Ligadura de una variable

Se define como el alcance de una variable como la región dentro del programa en el cual ocurren todas las referencias a dicha variable.

```
(define x                                     ; Variable x1
  (lambda (x)                                 ; Variable x2
    (map (lambda (x)                          ; Variable x3
           (+ x 1))                          ; Ref x3
         (x ' (1 2 3)))                     ; Ref x2
    ))                                       ; Ref x1
```



# Los conceptos de Alcance y Ligadura de una variable

Ejemplo:

```
(lambda (z)
  ((lambda (a b c)
    (a (lambda (a)
      (+ a c))
      b))
   (lambda (f x)
     (f (z x))))))
```

```
(lambda (z)
  ((lambda (a b c)
    (a (lambda (a)
      (+ a c))
      b))
   (lambda (f x)
     (f (z x)))))
```

# Los conceptos de Alcance y Ligadura de una variable

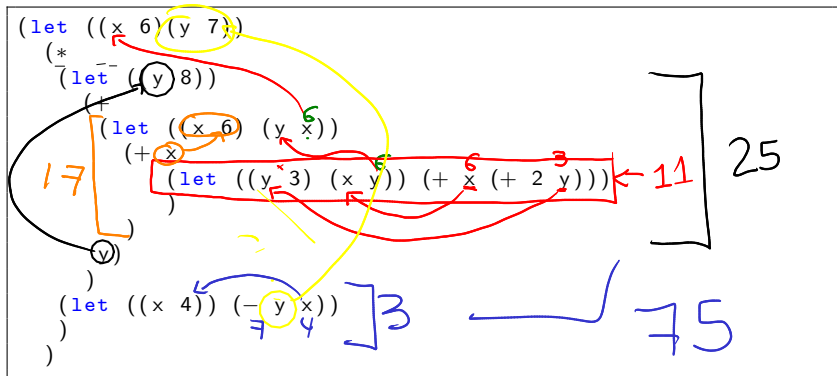
Cual es el valor de la siguiente expresión:

```
(let ((x 3) (y 4))  
  (+ (let ((x (+ y 5)))  
      (* x y))  
     x))
```

Handwritten annotations: A blue circle around the `x` in `((x 3))` with a blue arrow pointing to the `x` in `(+ x)` and the number `3` below it. A red circle around `(y 4)` with a red arrow pointing to the `y` in `(+ y 5)`. A pink circle around `(x (+ y 5))` with a pink arrow pointing to the `x` in `(* x y)`. The calculation  $9 \times 4 = 36$  is written in pink. The final result `39` is boxed in blue.

# Los conceptos de Alcance y Ligadura de una variable

Cual es el valor de la siguiente expresión:



# Los conceptos de Alcance y Ligadura de una variable

Cual es el valor de la siguiente expresión:

```
(let ((x 6)
      (y 7))
  (+ (let ((x (- y 6)))
        (* x y))
     (x)))
```

Handwritten annotations: Blue circles around the `7` in the `let` binding, the `x` in the inner `let` binding, the `x` in the inner `let` body, and the `x` in the outer body. Blue arrows show the binding of `x` to 6 in the outer scope and the binding of `x` to 1 in the inner scope. The value 13 is written below the expression.

13



(let ((x 6)(y 7)(z 8))

(+)

(let ((y 8) (z y))

(\*

(let ((x 6) (y x) (z y))

(+)

(+ ~~z~~ (\* ~~x~~ ~~y~~))

(let ((y 3) (~~x~~ ~~x~~) (~~z~~ ~~z~~))

(\* x (+ z y))

)

(let ((~~x~~ ~~z~~) (~~y~~ ~~y~~) (~~z~~ ~~x~~)) (\* x (+

y z)))

(x 7 14)

(let ((x 4)) (\* y (+ z x)))

12 \* 7 } 84

10864

110

10780

98

66

44

8

6

6

8

3

6

8

7

6

Abstracción de datos (Capítulo 2 EOPL)