

# Fundamentos de lenguajes de programación

## La relación entre Inducción y Programación

EISC. Facultad de Ingeniería. Universidad del Valle

Diciembre 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

- 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  $S$ .

Puede tener una regla recursiva para definir los valores de un tipo de dato

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





# Especificación inductiva

## Ejemplo

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

## Solución

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

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

$$\begin{array}{l} () \in S \\ \hline l \in S, n \in \mathbb{N} \\ \hline (n \text{ } l) \in S \end{array}$$

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

$$\begin{array}{l} \swarrow 1 \in \mathbb{N}, (2(3())) \in S \\ \swarrow 2 \in \mathbb{N}, (3()) \in S \\ \swarrow 3 \in \mathbb{N}, () \in S \end{array}$$

$$4, 3 \in \mathbb{N} \times$$

# Especificación inductiva

## Especificación formal

Ahora realicemos la especificación inductiva de:

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

$$1) \quad 5 \in S$$

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

$$\begin{array}{l} 1) \quad () \in S \\ 2) \quad \frac{1 \in S, n \in P}{(n1) \in S} \\ 3) \quad \frac{n \in P}{n+2 \in P} \end{array}$$

# Especificación inductiva

## Lista de números pares

1  $2 \in P$

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

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



$$(n, 7n+1) \quad \text{genera!}$$

## Ejercicios

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

Tuplas:  $(2, 15)$  ( )

1  $(0, 1) \in S, \frac{(n, k) \in S}{(n+1, k+7) \in S}$

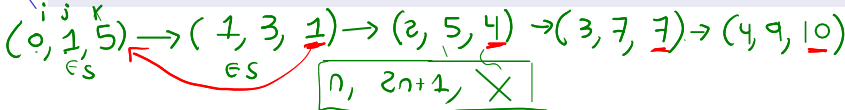
2  $(0, 1) \in S, \frac{(n, k) \in S}{(n+1, 2k) \in S}$



$$(n, 2^n)$$

3  $(0, 1, 5) \in S, \frac{(i, j, k) \in S}{(i+1, j+2, i+j) \in S}$

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



# 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

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

Ejemplos de árboles binarios son:

1

(foo 1 2)

(foo 1 (bar 2 3) 3)



# Especificación mediante gramáticas

## Expresión calculo $\lambda$

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

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

Ejemplo de cálculo  $\lambda$ :

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

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

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

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

Condición de parada

¿Como se comporta la función el caso base/para da? R/ list-length debería responder: 0

¿Como es el comportamiento  $\text{num } \langle \text{lista} \rangle$ ?  
+ 1 (list-length resto)

Primer resto

# 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

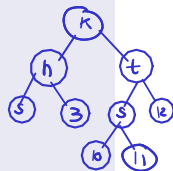
## Un árbol binario

Recordando la definición de los árboles vista anteriormente:

$\langle \text{arbol-binario} \rangle ::= \langle \text{int} \rangle$  *Caso 1*  
 $::= (\langle \text{simbolo} \rangle \langle \text{arbol-binario} \rangle \langle \text{arbol-binario} \rangle)$  *Caso 2*  
*(car) (cdr) (caddr)*

Este árbol será representado así:

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



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



¿Que quieren hacerle a los árboles binarios?

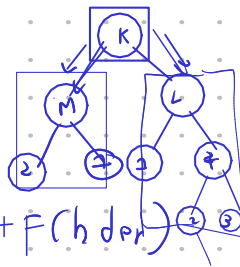
1) Función que calcule la h

- Caso 1

- Caso 2

R/O

$\max(1 + F(h_{izq}), 1 + F(h_{der}))$



2) Función recorridos

- Preorden R-I-D

- Inorden I-R-D

- Posorden I-D-R

} list + 9

Caso 1 ¿Que pasa en cualquier recorrido? retorna el mismo nodo

Caso 2 ¿Que pasa en este caso?

Preorden (list sim (f hizq) (fhder))

Inorden (list (f hizq) sim (fhder))

Posorden (list (f hizq) (fhder) sim)

¿Como hago para saber si un arbol es balanceado

R/ Si la altura en en sus diferentes ramas en h  
o h-1

Caso 0: Si es solo un nodo es **BALANCEADO**

Caso 1.

(or

1) 4 → 1  
↘ 3

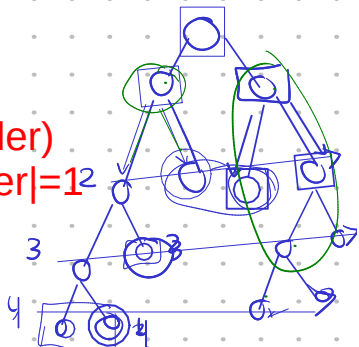
2) 4

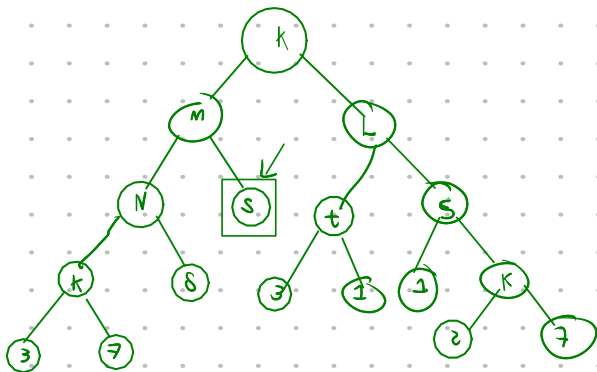
(altura-hiz = altura-der)

$|altura-hiz - altura-der| = 1$

(balanceado hizq)

(balanceado hider)





# 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)
        (cons l empty)
        (append
         (arbol->lista (cadr l))
         (arbol->lista (caddr l))
         )
        )
    )
  )
)
```

# 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:

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

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

*(nth-element (S 6) 1)*

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

Dos parametros lista y tenemos un number (ind)

Caso 1: '()

(error "No tengo ese elemento socito :(")

Caso: (numero <lista>)

- ind = 1 numero
- ind > 1 f (cdr lst, indice -1)
- ind < 1 error

- 2 remove-first (remove-first '(1 2 3)) retorna '(2 3)

Caso 1: '()

Error: Porque es una lista vacía

Caso 2 num <lst>

Retorna (cdr lst)



# 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**

*asignación*

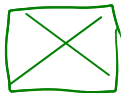
- Declaración:

```
(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

C++, Ruby(globales) da un valor por defecto  
Java, Javascript, no da un valor por defecto (Excepción)  
Python, Ruby(locales) debes declarar un valor

# Los conceptos de Alcance y Ligadura de una variable

$$\begin{aligned} \langle \text{lambda-exp} \rangle &::= \langle \text{identificador} \rangle \\ &::= (\text{lambda } \langle \text{identificador} \rangle \langle \text{lambda-exp} \rangle) \\ &::= (\langle \text{lambda-exp} \rangle \langle \text{lambda-exp} \rangle) \end{aligned}$$

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

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

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

NO  
 Si

# No

Si

SY

SI

↑ s r s r

No

No

z?  
→ No

22

y?

# 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

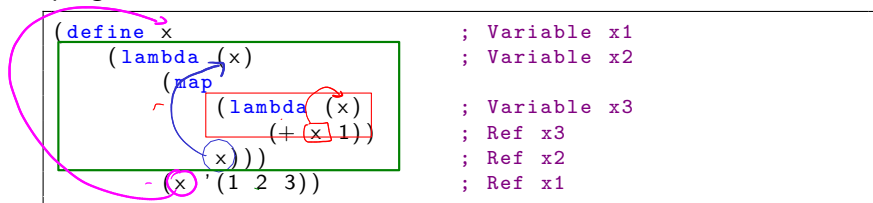
## Determinar si una variable ocurre libre

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.

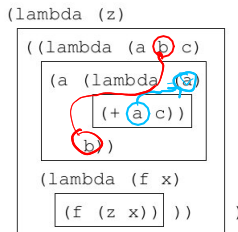




# Los conceptos de Alcance y Ligadura de una variable

Ejemplo:

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



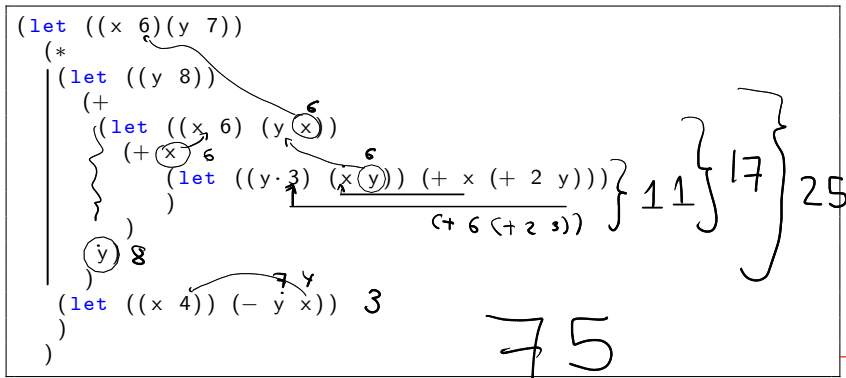
# Los conceptos de Alcance y Ligadura de una variable

Cual es el valor de la siguiente expresión:

<pre>(let ((x 3) (y 4))   (+ (let ((x) (+ y 5)))      (* x y)))</pre> <p>Handwritten annotations: A box around the inner <code>let</code> block. Arrows pointing from the <code>x</code> in <code>(x)</code> to the <code>x</code> in <code>(* x y)</code>. A circled <code>x</code> with an arrow pointing to the <code>x</code> in <code>(* x y)</code>. The number <code>36</code> is written next to the expression.</p>	<p>39</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------

# 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:

$$\begin{aligned}
 & (\text{let } ((x \ 6) \\
 & \quad (y \ 7)) \\
 & (+ (\text{let } ((x \ (- \ y \ 6))) \\
 & \quad \quad (* \ x \ (y \ 1)) \ 1) \ 7) \ 7) \ 13
 \end{aligned}$$

# Próxima sesión

.

Abstracción de datos (Capítulo 2 EOPL)