

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:

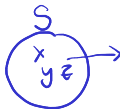
- 1 Especificación inductiva
- 2 Especificación mediante gramáticas.

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 .



Especificación inductiva

Números pares

- 1 Si $n = 2$ entonces n es par $\textcircled{2}^s$
- 2 Si n es par, entonces $n + 2$ también es par. $2, 4, 6, 8, 10, \dots$

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 $(\)$ $(5 \ (\))$ $(4 \ (5 \ (\)))$

Especificación inductiva

Especificación formal

Ahora formalmente:

Números pares

1 $2 \in S$

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

Regla recursiva

Lista de números

1 $() \in S$

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



Universidad
del Valle

Especificación inductiva

Ejemplo

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

Solución

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

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

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

4 $(4()) \in S, 3 \in \mathbb{N}$

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

Especificación inductiva

Especificación formal

Ahora realicemos la especificación inductiva de:

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

Especificación inductiva

Lista de números pares

1 $2 \in P$

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

Especificación
números pares

1 $() \in S$

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

car cdr

Múltiplos de 5

1 $5 \in S$

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

Especificación inductiva

Ejercicios

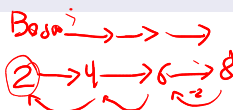
Indique que tipo de conjuntos están definidos por las siguientes reglas:

1 $(0, 1) \in S, \frac{(n, k) \in S}{(n+1, k+7) \in S}$ $(0, 1) \xrightarrow{x} (1, 8) \xrightarrow{x} (2, 15) \xrightarrow{x} (3, 22) \xrightarrow{x} \dots$
 $\hookrightarrow x \cdot 7 + 1$ $(x), 7x+1$

2 $(0, 1) \in S, \frac{(n, k) \in S}{(n+1, 2k) \in S}$ $(0, 1) \xrightarrow{x} (1, 2) \xrightarrow{x} (2, 4) \xrightarrow{x} (3, 8) \xrightarrow{x} \dots$
 $x \in \mathbb{N}$

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

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

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

Handwritten notes: "Vacio" with an arrow pointing to the empty list rule. Red circles around $\langle \text{int} \rangle$ and $\langle \text{lista-de-enteros} \rangle$ in the recursive rule. A red arrow points from the recursive rule back to the empty list rule.

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

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

Handwritten notes: A red arrow points from the asterisk to a red circle containing "()", indicating the base case. Another red arrow points from the asterisk to a box containing four squares, representing a sequence of elements.



Especificación mediante gramáticas

Árbol Binario

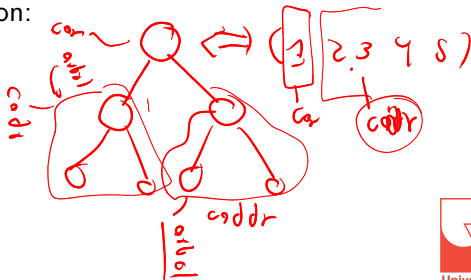
`<arbol-binario> ::= <int>`
`<arbol-binario> ::= (<simbolo> <arbol-binario> <arbol-binario>)`

Ejemplos de árboles binarios son:

1

(foo 1 2)

(foo 1 (bar 2 3) 3)



Especificación mediante gramáticas

Expresión calculo λ

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

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

Ejemplo de cálculo λ :

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

$x, y, \text{foo}, \text{holy},$
variables

data { 1 7 True
8 False

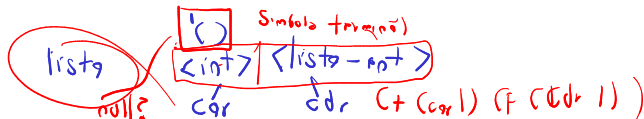
lambda (variable)

Función

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

$::= ()$
 $::= \underline{\hspace{2cm}}$

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.

Especificación recursiva de programas

if <pregunta> <rot> <resf>)

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)))))
)
```

Handwritten notes illustrating the recursive process for the list (1 4 7 9 10):

cf [1 4 7 9 10] (0)

(+ 1 (cf [4 7 9 10]))

(+ 1 (+ 1 (cf [7 9 10])))

(+ 1 (+ 1 (+ 1 (cf [9 10])))

(+ 1 (+ 1 (+ 1 (+ 1 (cf [10])))))

(+ 1 (+ 1 (+ 1 (+ 1 (+ 1 (cf [])))))

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

Handwritten notes illustrating the recursive process for the list (1 4 7 9 10):

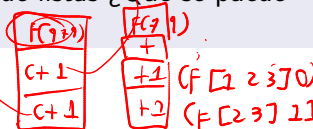
F(1st 9cc)

(define f

(lambda (lst 9cc)

(null? lst 9cc)

(f (lst (cdr lst 9cc))

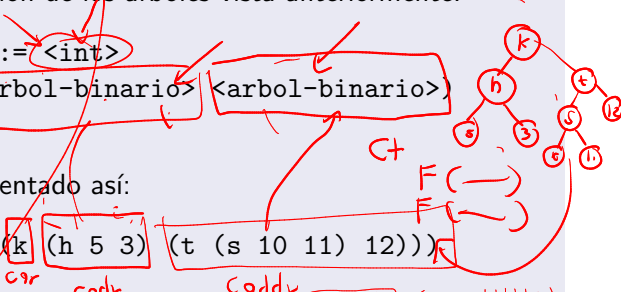


Especificación Recursiva de programas

Un árbol binario

Recordando la definición de los árboles vista anteriormente:

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



Este árbol será representado así:

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

Handwritten annotations: *cgr* (under 2), *codr* (under (h 5 3)), *caddr* (under (t (s 10 11) 12))), and *c9dddr* (under the nested list structure).

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

Especificación Recursiva de programas

B Tree

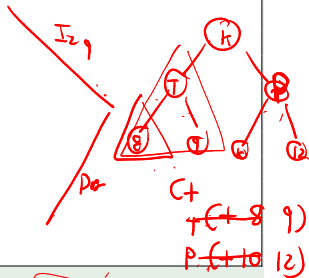
car cadr caddr caddr caddr caddr caddr
 [1 2 3 4 5 6 7] (car cadr caddr caddr caddr caddr caddr)

Un árbol binario

Procedimiento sum-arbol:

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

Handwritten notes: $I_2, 9$, $I_2, 9$ Den, P_2 , $C+$, $P+$



$$3 - 3^2 - 3^3 - 3^{10}$$



11 hijos Derecha

$O(n^d)$

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))
         )))
  )
)
```

Handwritten annotations:

- Red circles around `<int>` and `<simbolo>` in the BNF input rule.
- Red circles around `'()'` and `<int>` in the BNF output rule.
- Red circle around `(number? l)` with the note "Cond par, d9".
- Red box around the recursive calls `(arbol->lista (cadr l))` and `(arbol->lista (caddr l))`.
- Red arrows pointing from the box to the notes "H+I" and "HD".

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:

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

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

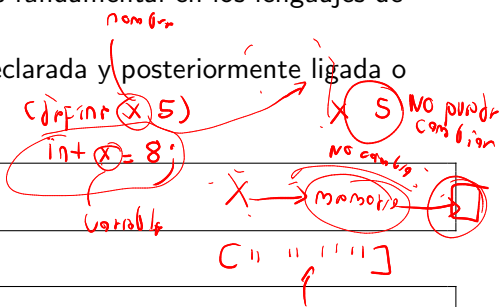
- El concepto de variable es fundamental en los lenguajes de programación
- Una variable puede ser declarada y posteriormente ligada o referenciada

- 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

```
function f(x,y)
```

```
    j = x+y
```

```
endfunction
```

```
f(4,3)
```

```
[print(j)]
```

Octave

Matlab

global
python

Block scope

```
if/for {  
    int x = 8;  
}
```

let

C++ Java
JavaScript

Function scope 2016+

def func():

```
if _____  
    a = 8  
else  
    a = 7  
  
    a = a + 8  
    ↗ var QX  
    ↖  
    JavaScript
```

Estático

Los conceptos de Alcance y Ligadura de una variable

Calculo λ

- 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 .

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

Los conceptos de Alcance y Ligadura de una variable

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

- $'(\text{lambda } (x) (\text{lambda } (y) x))$ **F**
 - $'((\text{lambda } (z) (\text{lambda } (y) x)) x)$ **T**
 - $'((\text{lambda } (y) (\text{lambda } (y) x)) ((\text{lambda } (z) x) x))$ **T**
- Handwritten annotations:* Red boxes around the x in the second and third expressions. Red lines connecting the x in the third expression to the x in the second expression. Red text "T or T" below the third expression.

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 λ

```
<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  
      1) ((symbol? exp) (eqv? var exp))  
      2) ((eqv? (car exp) 'lambda)  
         (and (not (eqv? (caadr exp) var))  
              (occurs-free? var (caddr exp))))  
      3) (else (or (occurs-free? var (car exp))  
                   (occurs-free? var (cadr exp))))))
```

Los conceptos de Alcance y Ligadura de una variable

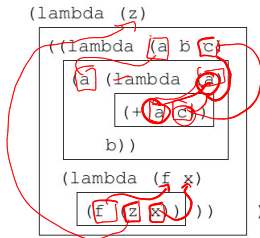
Se define como el alcance de una variable como la región dentro del programa en el cual ocurren todas las referencias a dicha variable.

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



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)
```

$(+ 9 4) = 13$
 $(* 3 13) = 39$

bounding

39

Los conceptos de Alcance y Ligadura de una variable

Cual es el valor de la siguiente expresión:

```
(let ((x 6)(y 7))
  (*
    (let ((y 8))
      (+
        (let ((x 6) (y x))
          (+ x
            (let ((y 3) (x y) (+ x (+ 2 y))))
          )
        )
      y)
    )
  (let ((x 4)) (- y x)))
```

Handwritten annotations in red:

- Arrows showing variable binding and scope resolution.
- Values written next to variables: $x=6$, $y=8$, $x=6$, $y=3$, $x=4$.
- Intermediate calculations: 14 , 17 , 8 , 25 , 75 .
- A box around the expression $(+ 2 y)$ with the value 5 written next to it.
- A box around the expression $(- y x)$ with the value 3 written next to it.

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)
)
```


Abstracción de datos (Capítulo 2 EOPL)