Fundamentos de lenguajes de programación

La relación entre Inducción y Programación

EISC. Facultad de Ingeniería. Universidad del Valle

Diciembre de 2020

Contenido

Contenido

Universidad del Valle

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
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

Universidad del Valle

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

Especificación inductiva

Ejemplo

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

Solución

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

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

$$2 \in P \qquad\qquad () \in S$$

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

Universidad del Valle

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

Universidad del Valle

Defina una lista de lista de simbolos

Sym = conjunto de simbolos   symbol?

$() \in Ls$

$$\frac{s \in Sym \wedge l \in Ls}{(s \;\; l) \in Ls}$$

$() \in Lss$

$$\frac{p \in Ls \wedge k \in Lss}{(p \;\; k) \in Lss}$$

For a l in Ls

[1 2 3]

Definir inductivamente las listas de parejas cuyo primer elemento es multiplo de 5 y su segundo elemento es una lista de multiplos de 4.

$5 \in M_5$

$4 \in M_4$

$(n, k) \in P_a$

$n \in M_5$

$k \in M_4$

$$\frac{p \in M_5}{p + 5 \in M_5}$$

$$\frac{k \in M_4}{k + 4 \in M_4}$$

Generar la especificación inductiva de las listas que contiene listas de números pares

$$- \quad 2 \in P$$

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

$$() \in L_p$$

$$\frac{k \in P,\ l \in L_p}{(k\ l) \in L_p}$$

$$() \in L_{pp}$$

$$\frac{t \in L_p,\ j \in L_{pp}}{(t\ j) \in L_{pp}}$$

Especificación inductiva

Ejercicios

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

1. $(0,1) \in S$, $\frac{(n,k) \in S}{(n+1,k+7) \in S}$
2. $(0,1) \in S$, $\frac{(n,k) \in S}{(n+1,2k)}$
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.

Universidad del Valle

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

Universidad del Valle

Especificación mediante gramáticas

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

Lista números

```
<lista-de-enteros> ::= ()
   ::= (<int> <lista-de-enteros>)

<lista-de-enteros> ::= ()|(<int> <lista-de-enteros>)

<lista-de-enteros> ::= (<int>)*
```

Universidad del Valle

Especificación mediante gramáticas

Árbol Binario

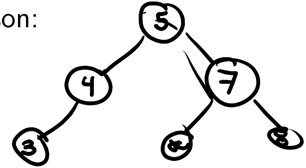
```
<arbol-binario> ::= <int>
                ::= (<simbolo> <arbol-binario><arbol-binario
                    >)
```

Ejemplos de árboles binarios son:

1

(foo 1 2)

(foo 1 (bar 2 3))



BTREE

Especificación mediante gramáticas

cadr

car (cadr
caadr

Expresión calculo λ

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

1 2 3

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

Ejemplo de cálculo λ:

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

```
<arbol-t> ::= <simbolo>
          ::= <numero>
          ::= <simbolo> <arbol-t>
                <arbol-t> <arbol-t>
          ::= <numero> <arbol-t>
                <arbol-t> <arbol-t>
'x
5
'(x 4 5 y)
'(5 (4 5 a p) (5 (4 5 6 7) 2 3) 9)
```

Contenido

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

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

Universidad del Valle

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?

Universidad del Valle

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)
```

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

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

Universidad del Valle

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

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
```

Universidad del Valle

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

- 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.

Universidad del Valle

Los conceptos de Alcance y Ligadura de una variable

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

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

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
      ((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)))))))
```

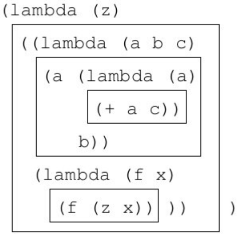
Los conceptos de Alcance y Ligadura de una variable

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)))                 ; Ref x2
        (x '(1 2 3))             ; Ref x1
```

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)
)
```

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))
   )
  )
```

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)
  )
```

Próxima sesión

Abstracción de datos (Capitulo 2 EOPL)