Fundamentos de lenguajes de programación

Robinson Duque, Ph.D

Universidad del Valle

robinson.duque@correounivalle.edu.co

Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Este documento es una adaptación del material original de los profesores
Carlos Andres Delgado y Carlos Alberto Ramírez

Overview

1. Especificación recursiva de datos
   - Especificación inductiva
   - Especificación mediante gramáticas en forma BNF
2. Especificación recursiva de programas
   - Ejemplos
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

La *especificación inductiva* es un método para especificar un conjunto de valores:

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

(Top-Down definition) Números múltiplos de 3

Un *número natural* n está en S si y sólo si:

1. $n = 0$, o
2. $n - 3 \in S$

Sabemos que $0 \in S$, entonces $3 \in S$ dado que $(3 - 3) = 0$ y $0 \in S$. Igualmente $6 \in S$ dado que $(6 - 3) = 3$ y $3 \in S$...Podemos concluir que S es el conjunto de números naturales que son múltiplos de 3.

¿Está 1 en S?, sabemos que $1 \neq 0$, entonces la primer condición no se satisface, adicionalmente $(1 - 3) = -2$, lo cual no es un número natural y por consiguiente no es un número de S, así la segunda condición no se satisface.

Especificación inductiva

(Top-Down definition) Números múltiplos de 3

Un *número natural* n está en S si y sólo si:

1. $n = 0$, o
2. $n - 3 \in S$

Podemos utilizar esta definición para escribir un procedimiento (*predicado*) que decida si un número natural n está en S:

```
;in-S? : N -> Bool
;usage: (in-S? n) = #t if n is in S, #f otherwise

(define in-S?
  (lambda (n)
    (if (zero? n)
      #t
      (if (>= (- n 3) 0)
          (in-S? (- n 3))
          #f ))))
```

Especificación inductiva

(Bottom-up definition) Números múltiplos de 3

Definición del conjunto más pequeño de S contenido en los números naturales N:

1. $n \in 0$, y
2. si $n \in S$, entonces $n + 3 \in S$.

(Rules of inference definition) Números múltiplos de 3

Definición utilizando reglas de inferencia. Cada entrada es una regla; la línea horizontal se lee como "si - entonces"; la parte superior es la hipótesis (o antecedente); la parte inferior es la conclusión (o consecuente):

$$\frac{}{0 \in S}$$

$$\frac{n \in S}{(n + 3) \in S}$$

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 (cons n l) es una lista de números

Especificación inductiva

Especificación mediante reglas de inferencia:

Números pares

1. $2 \in S$
2. $\frac{n \in S}{(n+2) \in S}$

Lista de números

1. $empty \in S$
2. $\frac{l \in S, n \in N}{(\text{cons n l}) \in S}$

Lista de números (otra posible especificación)

1. $() \in \textit{List-of-Int}$
2. $\frac{l \in \textit{List-of-Int}, n \in \textit{Int}}{(\text{n . l}) \in \textit{List-of-Int}}$

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. $empty \in S$
2. $\frac{l \in S, n \in N}{(\text{cons 2n l}) \in S}$

Múltiplos de 5

1. $5 \in S$
2. $\frac{n \in S}{(n+5) \in S}$

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

Especificación mediante gramáticas

Lista números

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

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

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

¿cuáles de las siguientes listas son listas válidas según la gramática?

- (2 3 t 6 7)
- (2 3 5 6 7)
- (2 3 (5 6) 7)

Especificación mediante gramáticas

Listas de símbolos y listas

```
<S-list> ::= ({<S-exp>}*)
<S-exp> ::= <Symbol> | <S-list>
```

¿cuáles de las siguientes listas son válidas según la gramática?

- (a b c)
- (an (((s-list)) (with () lots) ((of) nesting)))
- (a 3 (b c) 4)

Especificación mediante gramáticas

Árbol Binario

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

¿cuáles de los siguientes árboles son válidos según la gramática?

- 2
- (f 1 2)
- (f 1 2 3)
- (b 1 (f 1 2))
- (b x (f 1 2))
- (b (b f (f 1 2)) (b 4 5))
- (b (b 1 (f 1 2)) (b 4 5))

Especificación mediante gramáticas

Expresión calculo λ

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

El **cálculo lambda** es un **lenguaje simple** que se utiliza frecuentemente para estudiar la teoría de lenguajes de programación. Consiste sólo de referencias a variables, procedimientos que toman un único argumento, y llamados a procedimientos.

Especificación mediante gramáticas

Expresión calculo λ

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

¿Cuáles de las siguientes expresiones lambda son válidas según la gramática?

- x
- (s y)
- (s 1)
- (lambda (y) x)
- ((lambda (y) x) z)
- ((lambda (y) x) (lambda (u) u))
- (lambda (x) ((lambda (y) y) p))
- (lambda (x) ((lambda (y) y)))

Especificación mediante gramáticas

Expresión calculo λ

Expresiones

En el cálculo λ las funciones son ciudadanos de primera clase

```
(lambda (x) (+ x 1)) ;Como función
((lambda (x y) (* x y)) 1 2) ;Como valor
```

Funciones pueden pasarse como parámetros y retornarse al igual que los valores.

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

¡Sigue la gramática!

Especificación recursiva de programas

Gramática: Listas en Scheme

```
<List> ::= ()
       ::= (<Scheme-Value> <List>)
```

Función que calcula la longitud de una lista

```
;list-length : List -> Int
;usage: (list-length l) = the length of l
(define list-length
  (lambda (lst)
    (if (null? lst)
      0
     (+ 1 (list-length (cdr lst))))))
```

```
;Cómputo de list-length en ejecución:

(list-length '(a (b c) d))
= (+ 1 (list-length '((b c) d)))
= (+ 1 (+ 1 (list-length '(d))))
= (+ 1 (+ 1 (+ 1 (list-length '()))))
= (+ 1 (+ 1 (+ 1 0)))
= 3
```

Especificación recursiva de programas

Gramática: Listas en Scheme

```
<List> ::= ()
       ::= (<Scheme-Value> <List>)
```

Función que retorna el elemento en la posición n

```
;nth-element : List x Int -> SchemeVal
;usage: (nth-element lst n) = the n-th element of lst

(define nth-element
   (lambda (lst n)
     (if (null? lst)
        (report-list-too-short n)
        (if (zero? n)
           (car lst)
           (nth-element (cdr lst) (- n 1))))))

(define report-list-too-short
(lambda (n)
(eopl:error 'nth-element "List too short by ~s elements.~%" (+ n 1))))
```

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 con una lista, debido a la definición recursiva. Ejemplo:

```
(define arbol '(k (h 5 3) (t (s 10 11) 12)))
(define otroArbol 2)
```

Especificación Recursiva de programas

Gramática: árbol binario

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

Función que suma los elementos de un árbol binario

```
;sum-arbol : arbol-binario -> Int
;usage: (sum-arbol arbol) = suma de las hojas del árbol

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

Especificación Recursiva de programas

Gramática: lista de números

```
<lista-numeros> ::= '() | (<int> <lista-numeros>)
```

Función que suma los elementos de una Lista números

```
;;sum-lista-numeros : lista-numeros -> Int
;;usage: (sum-lista-numeros lst) = suma los elementos de la lista

(define suma-lista-numeros
  (lambda (lst)
    (if (eqv? lst empty)
        0
        (+ (car lst) (suma-lista-numeros (cdr lst))))
    )
  )
```

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 o referenciada
  - Declaración:

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

  - Referencia:

```
(f x y)
```

Los conceptos de Alcance y Ligadura de una variable

- Una variable esta ligada al lugar donde se declara
- El valor 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

- Una variable x ocurre libre en una expresión E si y solo si existe algún uso de x en E el cual no está ligado a ninguna declaración de x en E

```
((lambda (x) x) y)
```

Los conceptos de Alcance y Ligadura de una variable

Gramática de una expresión λ

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

Determinar si una variable ocurre libre

Así mismo podemos definir un procedimiento para determinar si una variable ocurre libre

```
(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.

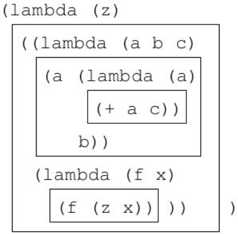
```
(define x                                           ; Variable x1
        (lambda (x)                                 ; Variable x2
                (map
                        (lambda (x)                 ; Variable x3
                                (+ x 1))            ; Ref x3
                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

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

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)
)
```

R/= 39

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)
  )
```

R/= 13

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))
   )
  )
```

R/= 75

Próxima sesión

Abstracción de datos (Capitulo 2 EOPL)