Definiciones locales y alcance léxico

Fundamentos de programación

EISC-Facultad de Ingeniería
Universidad del Valle
Ángela Villota Gómez
avillota@eisc.univalle.edu.co

Contenido

- La clase pasada vimos:
- Arboles
- Esta clase veremos:
  - Recorrido de árboles
  - Conceptos de localidad y alcance léxico
  - Sintaxis de **local**
  - Ejemplos

Ejercicio

- Escribir en scheme el árbol binario de los ejemplos anteriores.
- Diseñe la función cuantos? Que tiene como entrada un árbol binario y retorna el número de nodos que tiene dicho árbol.
  - EL total es igual a 1 (la raíz) +los nodos del subárbol izquierdo + los nodos del subárbol derecho
- Diseñe la función es-padre? Que toma como entrada dos nodos n1 y n2 y retorna true, en caso en que n1 sea el papá o la mamá de n2.
  - N1 puede ser el padre o la madre de n2
- Diseñe la función ancestros que dado un nodo, retorna la lista de nombres de los ancestros de dicho nodo.
  - La lista de ancestros de un nodo son: el papá y la mamá del nodo unido a la lista de ancestros del padre unido a la lista de ancestros de la madre. Cree la función unir-3 que permite concatenar 3 listas.

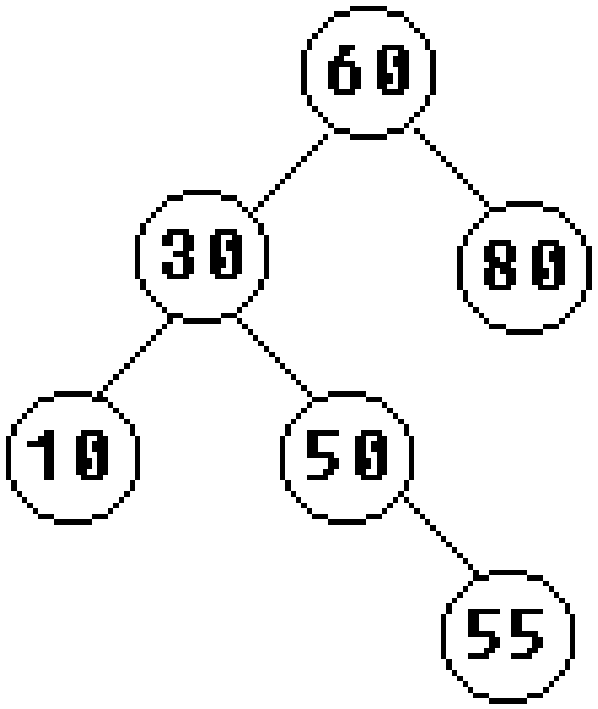
Ejercicio

- Diseñe la función hermano? que tiene como entrada un par de nodos y retorna true si tienen el mismo padre, o la misma madre, false en el caso contrario.
  - En árboles de ancestros, hay que preguntar si tienen el mismo padre o la misma madre
  - En árboles binarios es necesario primero desarrollar la función padre? que retorna el padre de un nodo
- Diseñe la función hijos que retorna la lista de nombres de los hijos de un nodo. La entrada es un nodo.
- Con la ayuda de las dos funciones anteriores, diseñe la función primos, que dado un árbol de ancestros retorne la lista de nombres de los primos de un nodo.

Recorrido de los árboles

- Recorrer un árbol es visitar todos sus nodos. Existen 3 órdenes posibles:
  - **Preorden:**
    - Examinar la raíz.
    - Recorrer el subárbol izquierdo en preorden.
    - recorrer el subárbol derecho en preorden.
  - **Postorden:**
    - Recorrer el subárbol izquierdo en postorden.
    - Recorrer el subárbol derecho en postorden.
    - Examinar la raíz.
  - **Inorden:**
    - Recorrer el subárbol izquierdo en inorden.
    - Examinar la raíz.
    - Recorrer el subárbol derecho en inorden.

Ejemplos



Recorridos

- **inorden:** 10 , 30 , 50 , 55 , 60 , 80
- **preorden:** 60 , 30 , 10 , 50 , 55 , 80
- **postorden:** 10 , 55 , 50 , 30 , 80 , 60

DEFINICIONES LOCALES Y ALCANCE LÉXICO

Introducción

- ¿ Qué es algo local?
- ¿ Cual es la diferencia entre algo local y algo global?

Alcance léxico

- Considere las siguientes definiciones:

```
(define (f x) (+ (* x x) 25))
(define (g x) (* 12 (expt x 5)))
```

  - ¿Son las mismas x (rojas y verdes)?
    - no son las mismas, se refieren a cosas distintas.
  - ¿Por qué?
    - Porque están en funciones distintas
  - ¿Si cambiamos las x por y's cambia el resultado del programa?
    - No el resultado del programa debe ser el mismo.

Alcance léxico (2)

- La región del programa en donde un nombre referencia a un valor en particular se llama alcance léxico.
- Fuera de su alcance, el nombre no significa la misma cosa.
- Ejemplo:

```
(define (f x) (+ (* x x) 25))
(* x x) ; -> genera un error
x: name is not defined, not an argument, and not a primitive name
```

Alcance léxico (3)

- El alcance léxico depende de cómo se hizo la definición de un nombre:
  - Para una variable global, para una función y para una estructura
    - **El alcance es total se puede usar en las funciones y en la ventana de ejecución**
  - Para una variable dentro de una función (entradas): el alcance es **local porque solo tiene significado dentro de la función**

Definiciones locales

- Hasta ahora solo hemos trabajado con definiciones globales, y algunas locales (entradas de las funciones)
- En scheme se pueden hacer definiciones locales por medio de la instrucción **local**
- **El alcance léxico de las definiciones hechas con local es la región delimitada con paréntesis**

local

- Se pueden definir localmente funciones o datos (variable, estructuras)
- Las definiciones locales se usan para que en el caso de que una función haga uso de datos especiales, o funciones auxiliares, estos puedan ser definidos dentro del cuerpo de la función.
- Las funciones o datos definidos localmente solo pueden ser usados por la función que los tiene definidos. (Alcance léxico)

local: sintaxis

- Para hacer definiciones locales usamos la expresión local, primero recordemos que es una expresión en scheme:

```
<var> = x | area-of-disk | perimeter | ...
<con> = true | false
        'a | 'doll | 'sum | ...
        1 | -1 | 3/5 | 1.22 | ...
<prm> = + | - | ...
```

Figure 20: Beginning Student Scheme: The core vocabulary

```
<exp> = <var>
      | <con>
      | (<prm> <exp> ...<exp>)
      | (<var> <exp> ...<exp>)
      | (cond (<exp> <exp>) ...(<exp> <exp>))
      | (cond (<exp> <exp>) ...(else <exp>))
```

Figure 21: Beginning Student Scheme: The core grammar

```
<def> = (define (<var> <var> ...<var>) <exp>)
      | (define <var> <exp>)
      | (define-struct <var> (<var> ...<var>))
```

Figure 51: Scheme definitions

local: sintaxis

- local es una expresión que tiene la siguiente forma:

  <exp> = (**local** (<def-1> ...<def-n>) <exp>)

  En donde def-1...def-n son definiciones en scheme.

  Como **local** es una expresión puede ir en cualquier parte en que se pueda poner una expresión.

Ejemplos:

- Cuales son los nombres locales?
- Cuales son los nombres globales?
- Cual es el cuerpo del local?

4. (local ((define x (* y 3)))
   (* x x))
6. (local ((define (f x) (g x (+ x 1)))
   (define (g x y) (f (+ x y))))
   (+ (f 10) (g 10 20)))

Ejemplos

1.
```
(local
  ((define (odd an)
     (cond
       [(zero? an) false]
       [else (even (sub1 an))]))
   (define (even an)
     (cond
       [(zero? an) true]
       [else (odd (sub1 an))])))
 (even a-nat-num))
```

Ejemplos

```
; mult10 : lista-numeros  ->  lista-numeros
;; crea una lista de números multiplicando cada uno de los
   elementos de la lista por una potencia de 10 (expt 10 p) en
   donde p es la cantidad de elementos que quedan en la lista
(define (mult10 lista)
  (cond
    [(empty? lista) empty]
    [else (local ((define un-digito (first lista))
                  (define p (length (rest lista))))
        ;; ---------------------------------------------------
            (cons (* (expt 10 p) un-digito) (mult10 (rest lista))))]))

(mult10 (list 1 1))
```