Capítulo 4

Recursividad Guiada por los Datos

4.1 Recursividad Plana

Hasta ahora, hemos estado efectuando recursividad sobre los elementos de alto nivel de una lista. A esto se le llama **recursividad plana**. Por ejemplo:

```
(define nuevo-append
(lambda (ls1 ls2)
   (if (null? ls1) ls2
(cons (car ls1) (nuevo-append (cdr ls1) ls2)))))
```

El procedimiento **nuevo-append** funciona igual que el **append** original:

(nuevo-append '(a b c) '(c d)) $\Longrightarrow$ **(a b c c d)**
(nuevo-append '() '(a b c)) $\Longrightarrow$ **(a b c)**
(nuevo-append '() '()) $\Longrightarrow$ **()**

De manera análoga, podemos definir el procedimiento **nuevo-reverse**:

```
(define nuevo-reverse
(lambda (ls)
(if (null? ls) '()
(nuevo-append (nuevo-reverse (cdr ls))
  (list (car ls))))))
```

Es interesante hacer notar que en Scheme es posible realizar recursividad **mutua** entre dos procedimientos. Considere el siguiente ejemplo:

(**define par?**
(**lambda** (int)
(**if** (**zero?** int) #t
(impar? (- int 1)))))

(**define impar?**
(**lambda** (int)
(**if** (**zero?** int) #f
(par? (- int 1)))))

Este es un buen ejemplo para introducir la función de depuración **trace** con la que cuenta Scheme:

>, trace par?
>, trace impar?
> (even? 3)
[Enter (par? 3)
[Enter (impar? 2)
[Enter (par? 1)
[Enter (impar? 0)
 Leave impar? #f
 Leave par? #f
 Leave impar? #f
 Leave par? #f
#f
>, untrace

El comando **untrace** sin parámetros detiene la sesión de depuración de todos los procedimientos bajo inspección.
A continuación se muestran ejemplos adicionales del uso de estos dos procedimientos:

(par? 4) $\Longrightarrow$ **#t**
(par? 0) $\Longrightarrow$ **#t**
(impar? 3) $\Longrightarrow$ **#f**
(impar? 3) $\Longrightarrow$ **#t**
(impar? 4) $\Longrightarrow$ **#f**

4.1.1 Problemas Resueltos

1. Defina un procedimiento **insertar-izquierda** que tome como parámetros a **nuevo**, **viejo** y **ls** y construya una lista obtenida de insertar el elemento **nuevo** a la izquierda de cada ocurrencia de nivel superior del elemento **viejo** en la lista **ls**. Pruebe su procedimiento con:

   (insertar-izquierda 'z 'a '(a b a c a)) $\Longrightarrow$ **'(z a b z a c z a)**
   (insertar-izquierda 0 1 '(0 1 0 1)) $\Longrightarrow$ **'(0 0 1 0 0 1)**
   (insertar-izquierda 'perro 'gato '(mi perro es divertido))
   $\Longrightarrow$ **'(mi perro es divertido)**
   (insertar-izquierda 'dos 'uno '()) $\Longrightarrow$ **'()**

   El procedimiento es el siguiente:

   (**define** insertar-izquierda
   (**lambda** (nuevo viejo ls)
   (**cond**
   ((**null?** ls) '())
   ((**equal?** (**car** ls) viejo) (**cons** nuevo
   (**cons** viejo (insertar-izquierda nuevo viejo (**cdr** ls)))))
   (**else** (**cons** (**car** ls) (insertar-izquierda nuevo viejo (**cdr** ls)))))))

2. Defina un procedimiento **subst** que tome como parámetros **nuevo**, **viejo** y **ls**. Este procedimiento debe construir una lista obtenida de reemplazar cada ocurrencia de alto nivel del elemento **viejo** en la lista **ls** por el elemento **nuevo**. Pruebe su procedimiento con:

   (subst 'z 'a '(a b a c a)) $\Longrightarrow$ **(z b z c z)**
   (subst 0 1 '(0 1 0 1)) $\Longrightarrow$ **(0 0 0 0)**
   (subst 'perro 'gato '(mi perro es divertido)) $\Longrightarrow$ **(mi perro es divertido)**
   (subst 'dos 'uno '()) $\Longrightarrow$ **()**

   El procedimiento es el siguiente:

   (**define** subst
   (**lambda** (nuevo viejo ls)

(**cond**
((**null?** ls) '())
 ((**equal?** (**car** ls) viejo) (**cons** nuevo (subst nuevo viejo (**cdr** ls))))
(**else** (**cons** (**car** ls) (subst nuevo viejo (**cdr** ls)))))))

4.2 Recursión Profunda

Ahora consideraremos la recursividad sobre todas las sublistas de una lista. Decimos que la sublista **(b c)** está anidada en la lista **'(a (b c))**. Si un elemento no está encerrado en paréntesis, entonces tiene un nivel de anidamiento de cero. Los elementos de una lista tal como **'(a b c)** tienen un nivel de anidamiento de 1. De tal forma, **'b** tiene un nivel de anidamiento de 1, mientras que toda la lista **'(a b c)** tiene un nivel de anidamiento de 0. Por lo tanto, cada nivel adicional de paréntesis añáde 1 al nivel de anidamiento, de manera que el nivel de anidamiento del elemento **'c** en **'(a (b (c d)))** es **3**. Los objetos en la lista que tienen un nivel de anidamiento de 1 son los objetos de alto nivel de la lista.

Con estos conceptos en mente, consideremos el siguiente ejemplo:

(**define** cuenta-todo
  (**lambda** (ls)
(**cond**
  ((**null?** ls) 0)
  ((**not?** (**pair?** (**car** ls))) (add1 (cuenta-todo (**cdr** ls))))
(**else** (+ (cuenta-todo (**car** ls)) (cuenta-todo (**cdr** ls)))))))

El procedimiento **cuenta-todo** toma una lista y regresa los elementos en ella que no son pares. Por ejemplo:

(cuenta-todo '((a b) c () ((d (e))))) $\Longrightarrow$ **6**
(cuenta-todo '(() () ())) $\Longrightarrow$ **3**
(cuenta-todo '(((())))) $\Longrightarrow$ **1**
(cuenta-todo '()) $\Longrightarrow$ **0**

Este tipo de recursividad difiere de la *recursividad plana* en que cuando el **car** de la lista es un par, se aplica el procedimiento definido tanto al **car** como al **cdr** de la lista. En la recursividad plana, el procedimiento definido

se aplica únicamente al **cdr** de la lista. Cuando se aplica la recursividad tanto al **car** como al **cdr** de la lista, decimos que se efectúa **recursividad profunda**.

4.2.1 Problemas resueltos

1. Defina un pocedimiento **subst-todo** que se invoque usando **(subst-todo nuevo viejo ls)**. Este procedimiento debe reemplazar cada ocurrencia del elemento **viejo** en una lista **ls** con el elemento **nuevo**. Pruebe su procedimiento con:

   (subst-todo 'z 'a '(a (b (a c)) (a (d a))))
   $\Longrightarrow$ **(z (b (z c)) (z (d z)))**
   (subst-todo 0 '(1) '(((1) (0)))) $\Longrightarrow$ **((0 (0)))**
   (subst-todo 'uno 'dos '()) $\Longrightarrow$ **()**

   El procedimiento es:

   (**define** subst-todo
   (**lambda** (nuevo viejo ls)
   (**cond**
   ((**null?** ls) '())
   ((**equal?** (**car** ls) viejo) (cons nuevo (subst-todo nuevo viejo (**cdr** ls))))
   ((**pair?** (**car** ls)) (**cons** (subst-todo nuevo viejo (**car** ls))
   (subst-all nuevo viejo (**cdr** ls))))
   (**else** (**cons** (**car** ls) (subst-todo nuevo viejo (**cdr** ls)))))))

2. Defina un procedimiento **inserta-izq-todo** que se invoque usando **(inserta-izq-todo nuevo viejo ls)**. Este procedimiento debe insertar el elemento **nuevo** a la izquierda de cada ocurrencia del elemento **viejo** en la lista **ls**. Pruebe su procedimiento con:

   (insert-izq-todo 'z 'a '(a ((b a) ((a (c))))))
   $\Longrightarrow$ **(z a ((b z a) ((z a (c)))))**
   (insert-izq-todo 'z 'a '(((a)))) $\Longrightarrow$ **(((z a)))**
   (insert-izq-todo 'z 'a '()) $\Longrightarrow$ **()**

El procedimiento es:

```
(define insert-izq-todo
(lambda (nuevo viejo ls)
(cond
((null? ls) '())
((equal? (car ls) viejo) (cons nuevo
 (cons viejo (insert-izq-todo nuevo viejo (cdr ls)))))
((pair? (car ls)) (cons (insert-izq-todo nuevo viejo (car ls))
     (insert-izq-todo nuevo viejo (cdr ls))))
(else (cons (car ls) (insert-izq-todo nuevo viejo (cdr ls)))))))
```

3. Defina un procedimiento **ext-izquierda** que tome como argumento una lista no vacía y retorne el elemento atómico de la extrema izquierda de dicha lista. Pruebe su procedimiento con:

   (ext-izquierda '((a b) (c (d e)))) $\Longrightarrow$ **a**
   (ext-izquierda '(((((c ((e f) g) h))))) $\Longrightarrow$ **c**
   (ext-izquierda '(() a)) $\Longrightarrow$ **()**

   El procedimiento es:

```
(define ext-izquierda
(lambda (ls)
(cond
((pair? (car ls)) (ext-izquierda (car ls)))
(else (car ls)))))
```

4.3 Recursividad vs. Iteraciones

Cuando usamos recursividad, se construye una *tabla de retorno* de tal forma que la respuesta final pueda obtenerse usando sus elementos. Consideremos el siguiente ejemplo:

```
(define nueva-length
  (lambda (ls)
(cond
  ((null? ls) 0)
(else (add1 (nueva-length (cdr ls)))))))
```

Ahora, si invocamos este procedimiento con una lista de longitud 3, veamos cómo funciona la recursividad:

Primero, se construye una **tabla de retorno**:
> (**nueva-length** '(a b c))
respuesta-1 : (1 + *respuesta-2*)
respuesta-2 : (**nueva-length** '(b c)): (1 + *respuesta-3*)
respuesta-3 : (**nueva-length** '(c)) : (1 + *respuesta-4*)
respuesta-4 : (**nueva-length** '()) : 0

Posteriormente, se efectúa una **sustitución hacia atrás**:
respuesta-4 : 0
respuesta-3 : 1
respuesta-2 : 2
respuesta-1 : **3**

La respuesta final **3** es lo que se imprime. La recursividad siempre trabaja de esta manera. Sin embargo, en Scheme también es posible definir un procedimiento que no construya una tabla de retorno. En este caso, los cálculos se efectúan a cada invocación recursiva sin tener que esperar otros resultados intermedios. Posteriormente, para cuando la condición de terminación se vuelve cierta, la respuesta ya ha sido calculada y simplemente se retorna. En general, cuando la computadora realiza este tipo de cálculos sin tener que usar una tabla de retorno, el proceso computacional es denominado **proceso iterativo**. La versión iterativa de **nueva-length** es la siguiente:

```
(define length-it
(lambda (ls acu)
(cond
  ((null? ls) acu)
(else (length-it (cdr ls) (add1 acu))))))

(define nueva-length
(lambda (ls)
(length-it ls 0)))
```

> (**nueva-length** '(a b c))
(**length-it** '(b c) 0)
(**length-it** '(c) 1)
(**length-it** '() 3)

El valor final del acumulador (**3** en este caso) es lo que se imprime.

4.4 Problemas Propuestos

1. Defina un procedimiento **insertar-derecha** con parámetros **nuevo**, **viejo** y **ls** que construya una lista obtenida de insertar el elemento **nuevo** a la derecha de cada ocurrencia de alto nivel del elemento **viejo** en la lista **ls**. Pruebe su procedimiento con:

   (**insertar-derecha** 'z 'a '(a b a c a)) $\Longrightarrow$ (a z b a z c a z)
   (**insertar-derecha** 0 1 '(0 1 0 1)) $\Longrightarrow$ (0 1 0 0 1 0)
   (**insertar-derecha** 'perro 'gato '(mi perro es divertido))
   $\Longrightarrow$ (mi perro es divertido)
   (**insertar-derecha** 'dos 'uno '()) $\Longrightarrow$ ()

2. Defina un procedimiento **suma-todo** que encuentre la suma de los números de una lista que puede contener sublistas anidadas de números. Pruebe su procedimiento con:

   (**suma-todo** '((1 3) (5 7) (9 11))) $\Longrightarrow$ 36
   (**suma-todo** '(1 (3 (5 (7 (9)))))) $\Longrightarrow$ 25
   (**suma-todo** '()) $\Longrightarrow$ 0

3. Defina un procedimiento **cuenta-todos-parens** que tome como argumento una lista y cuente el número de paréntesis que abren y cierran en la lista. Pruebe su procedimiento con:

   (**cuenta-todos-parens** '()) $\Longrightarrow$ 2
   (**cuenta-todos-parens** '((a b) c)) $\Longrightarrow$ 4
   (**cuenta-todos-parens** '(((a () b) c) () ((d) e))) $\Longrightarrow$ 14

4. Defina un procedimiento **cuenta-todo-fondo** que tome como argumentos a **elem** y una lista **ls** y regrese el número de elementos en **ls** que no sean iguales que **elem**. Use el predicado de igualdad más adecuado de acuerdo al tipo de datos mostrado en los ejemplos. Pruebe su procedimiento con:

   (**cuenta-todo-fondo** 'a '((a) b (c a) d)) $\Longrightarrow$ 3
   (**cuenta-todo-fondo** 'a '((((b (((a)) c)))))) $\Longrightarrow$ 2
   (**cuenta-todo-fondo** 'b '()) $\Longrightarrow$ 0

5. Defina un procedimiento **ext-derecha** que tome como argumento una lista no vacía y regresa el elemento atómico de la extrema derecha en la lista. Pruebe su procedimiento con:

   (**ext-derecha** '((a b) (d (c d (f (g h) i) m n) u) v)) $\Longrightarrow$ v
   (**ext-derecha** '(((((((b (c)))))))) $\Longrightarrow$ c
   (**ext-derecha** '(a ())) $\Longrightarrow$ ()

6. Escriba un procedimiento iterativo **haz-lista-ents-asc** que, para cualquier entero **n** dado, produzca una lista de enteros de 1 a n en orden ascendente. Luego escriba un procedimiento iterativo **haz-lista-ents-desc** que, para cualquier entero n dado, produzca una lista de enteros de n a 1 en orden descendente.