Seminario de Matemáticas - Práctica 1

15 de abril de 2013 Julio Ariel Hurtado Alegría

1. Objetivo General

Complementar el aprendizaje teórico respecto a la inducción matemática haciendo uso de un lenguaje de programación cuyo fundamento se encuentra en el razonamiento inductivo.

2. Desarrollo

Este es el conjunto de tareas a desarrollar en esta práctica:

1. Desarrollar los ejercicios dirigidos.
2. Análizar y desarrollar los ejercicios propuestos.

3. Ejercicios Dirigidos

(a) Dada una lista de naturales, diseñar e implementar en Scheme una función que rote la lista como si fuese una lista circular. Ejemplo:

```
(rotar '(1 2 3)) => (3 1 2)
(rotar '(a b c d e f)) => (f a b c d e)
(rotar '(z)) => (z)
(rotar '( )) => ( )
```

Solución:
Por inducción estructural se tiene que la función $Rotar(L) : L_{\mathbb{N}} \to L_{\mathbb{N}}$

a) $\emptyset$ si $L = \emptyset$

b) $k \to L'$, siendo $L = L' \to k, k \in \mathbb{N}$

Solución en Scheme: para solucionarlo entonces nos basamos en el diseño, el cual nos sugiere que debemos averiguar si la lista es vacía o no. Si ésta no lo es, entonces debemos extraer el último elemento de la lista, y a continuación agregarle la lista resultante de calcular el prefijo de la lista. Usando las funciones de Scheme, sería extraer la cabecera(car) de la lista invertida(reverse) y el sufijo(cdr) invertido(reverse) de la lista invertida(reverse nuevamente). Luego concatenar(cons) estos dos resultados.

```
(define (rotar lst)
        (cond
           ((null? lst) null)
           (else (cons (car (reverse lst)) (reverse (cdr (reverse lst)))))
        )
 )
```

(b) Defina inductivamente e implemente en Scheme el tipo abstracto cola de prioridad. Para ello define las funciones *make-queue*, para crear una cola, *put* para agregar un elemento a la cola, *best* para obtener el elemento de mayor prioridad, y *remove-best* para eliminar el elemento de mayor prioridad y *empty* para averiguar si la cola está vacía o no.

- Sea $Qp_{\mathbb{N}}$ el conjunto de todas las colas de prioridad de los naturales y se define como $(p, L) \in Qp_{\mathbb{N}}$, donde p es una función de prioridad, $L \in \mathbb{L}$ y $\forall k_i \in L$ se cumple que $p(k_{i-1}, K_i) = true$.
  Esta definición es usada para programar el constructor make-queue en Scheme(En este caso p es un procedimiento tal que (p a b) indica si a tiene mejor prioridad que b):

  ```
  (define (make-queue p)
          (cons p '())
  )
  ```

  Por ejemplo:

  ```
  (define q0 (make-queue (lambda (a b) (< a b))))
  ```

- Se define la función put para agregar un elemento a la cola de prioridad de la siguiente manera: Por inducción estructural se tiene que la función $Put(Q) : \mathbb{N}, Qp_{\mathbb{N}} \to Qp_{\mathbb{N}}$

  a) (p, e) si $L = \emptyset$ siendo Q=(p L).

  b) $(p, k \to L)$ si $p(k, Cab(L)) = true$ siendo $Q = (p, L)$

  c) $(p, cab(L) \to Sufijo(Put(Q))$ si $p(k, Cab(L)) = false$ siendo $Q = (p, L)$

  Usando esta definición definimos en Scheme la función *put* como:

  ```
  (define (put e q)
          (cond
          ((null? q) '())
          ((= (length q) 1) (append q (list e)))
          (((car q) e (cadr q)) (cons (car q) (cons e (cdr q))))
          ((cons (car q) (cons (cadr q) (cdr (put e (cons (car q) (cddr q)))))))
  )
  ```

4. Ejercicios Complementarios

- Defina mediante la inducción estructural e implemente en Scheme las funciones faltantes para la operación con colas de prioridad, es decir, las funciones *empty*, *best* y *remove-best*.

- Un conjunto de persona se representa mediante una lista de pares nombre y edad. Programe en Scheme un procedimiento show que use la cola de prioridad (y no otro algoritmo de ordenamiento) para que dado un conjunto de personas, entregue una lista con los nombres de las personas ordenadas de mayor a menor edad. Por ejemplo:

  ```
  (mostrar '((pedro 25) (ximena 16) (juan 17) (diego 32)))
  ```

  Mostrará

  ```
  (diego pedro juan ximena)
  ```

5. Problema

Una máquina de estado es un sistema que consume una lista de simbolos en entrada y retorna true si los acepta, false si no. A medida que consume simbolos, va cambiando de estado de acuerdo a ciertas transiciones entre estados. Para simplificar, consideramos que cada estado es un estado de aceptación: o sea, una máquina acepta cualquier input que pueda consumir enteramente. Además, en vez de trabajar con símbolos, trabajaremos con strings. Por ejemplo, el siguiente código Scheme define una máquina que reconoce la expresión regular $(a^+b)^*$ (por ej. "ab", "abaaabab"). Tiene dos estados, waita (el estado inicial) y waitab.

```
(define maquina-simple '(waita
                         (waita  (``a" waitab))
                         (waitab (``a" waitab)
                         (``b" waita))))
```

Resolver:

- Responder ¿cómo estan representados los estados de la máquina? ¿cómo están representadas las transiciones? ¿Cómo se especifica el estado inicial?.

- Basado en lo anterior utilice la inducción estructural para definir una máquina de estados con estas simplificaciones.

- Defina por inducción estructural y programe en Scheme una función llamada validar-maquina que, dado una máquina y un string de entrada, retorne true si la máquina acepta la entrada o false en caso contrario.