Práctica TAD - Inducción

1. Las listas finitas pueden especificarse como un TAD con los constructores:

- *nil*: Construye una lista vacía.
- *cons*: Agrega un elemento a la lista.

y las siguientes operaciones:

- *head*: Devuelve el primer elemento de la lista.
- *tail*: Devuelve todos los elementos de la lista menos el primero.

a) Dar una especificación algebraica del TAD listas finitas.

b) Asumiendo que A es un tipo con igualdad, especificar una función $inL : List\ A \to A \to Bool$ tal que $inL\ ls\ x = true$ si y sólo si x es un elemento de ls.

c) Especificar una función que elimina todas las ocurrencias de un elemento dado.

2. Dado el TAD pilas, con las siguientes operaciones:

- *empty*: Construye una pila inicialmente vacía.
- *push*: Agrega un elemento al tope de la pila.
- *isEmpty*: Devuelve verdadero si su argumento es una pila vacía, falso en caso contrario.
- *top*: Devuelve el elemento que se encuentra al tope de la pila.
- *pop*: Saca el elemento que se encuentra al tope de la pila.

Dar una especificación algebraica del TAD pilas.

3. Asumiendo que A es un tipo con igualdad, completar la siguiente definición del TAD conjunto:

$$
\begin{array}{l}
\textbf{tad}\ \textit{Conjunto}\ (A : \textit{Set})\ \textbf{where} \\
\quad \textbf{import}\ \textit{Bool} \\
\quad \begin{array}{ll}
vacio & : Conjunto\ A \\
insertar & : A \to Conjunto\ A \to Conjunto\ A \\
borrar & : A \to Conjunto\ A \to Conjunto\ A \\
esVacio & : Conjunto\ A \to Bool \\
union & : Conjunto\ A \to Conjunto\ A \to Conjunto\ A \\
interseccion & : Conjunto\ A \to Conjunto\ A \to Conjunto\ A \\
resta & : Conjunto\ A \to Conjunto\ A \to Conjunto\ A
\end{array} \\
\quad \rule{6cm}{0.4pt} \\
\quad x = y \Rightarrow insertar\ y\ (insertar\ x\ c) = insertar\ x\ c \\
\quad x \not\equiv y \Rightarrow insertar\ x\ (insertar\ y\ c) = insertar\ y\ (insertar\ x\ c) \\
\quad \ldots
\end{array}
$$

¿Que pasaría si se agregase una función $choose : Conjunto\ A \to A$, tal que $choose\ (insertar\ x\ c) = x$?

4.
El TAD *priority queue* es una cola en la cual cada elemento tiene asociado un valor que es su *prioridad* (a dos elementos distintos le corresponden prioridades distintas). Los valores que definen la prioridad de los elementos pertenecen a un conjunto ordenado. Las siguientes son las operaciones soportadas por este TAD:

- *vacia*: Construye una priority queue vacía.
- *poner*: Agrega un elemento a una priority queue con una prioridad dada.
- *primero*: Devuelve el elemento con mayor prioridad de una priority queue.
- *sacar*: Elimina de una priority queue el elemento con mayor prioridad.
- *esVacia*: Determina si una priority queue es vacía.
- *union*: Une dos priority queues.

Dar una especificación algebraica del TAD priority queue.

5. Demostrar que $(uncurry\ zip) \circ unzip = id$, siendo:

$$\begin{array}{ll}
zip & :: [a] \to [b] \to [(a, b)] \\
zip\ [\,]\ ys & = [\,] \\
zip\ (x : xs)\ [\,] & = [\,] \\
zip\ (x : xs)\ (y : ys) & = (x, y) : zip\ xs\ ys \\
unzip & :: [(a, b,)] \to ([a], [b]) \\
unzip\ [\,] & = ([\,], [\,]) \\
unzip\ ((x, y) : ps) & = (x : xs, y : ys) \\
 & \quad \textbf{where}\ (xs, ys) = unzip\ ps
\end{array}$$

6. Demostrar que $sum\ xs \leqslant length\ xs * maxl\ xs$, sabiendo que xs es una lista de números naturales y que $maxl$ y sum se definen

$$\begin{array}{ll}
maxl\ [\,] & = 0 \\
maxl\ (x : xs) & = x\ `max`\ maxl\ xs
\end{array}
\qquad
\begin{array}{ll}
sum\ [\,] & = 0 \\
sum\ (x : xs) & = x + sum\ xs
\end{array}$$

7. Dado el siguiente tipo de datos

$$\textbf{data}\ Tree\ a = H\ a \mid N\ a\ (Tree\ a)\ (Tree\ a)$$

a) Dar el tipo y definir la función *size* que calcula la cantidad de elementos que contiene un $Tree\ a$.

b) Demostrar la validez de la siguiente propiedad: $\forall\ \ t :: Tree\ a\ .\ \exists k \in \mathbb{N}.\ size\ t = 2\ k + 1$

c) Dar el tipo y definir la función *mirror* que dado un árbol devuelve su árbol espejo.

d) Demostrar la validez de la siguiente propiedad: $mirror \circ mirror = id$

e) Considerando las siguientes funciones:

$$\begin{array}{ll}
hojas\ :: Tree\ a \to Int & \\
hojas\ (H\ x) & = 1 \\
hojas\ (N\ x\ t_1\ t_2) & = hojas\ t_1 + hojas\ t_2
\end{array}$$

$$\begin{array}{l} altura :: Tree\ a \to Int \\ altura\ (H\ x) \qquad = 1 \\ altura\ (N\ x\ t_1\ t_2) = 1 + (altura\ t_1\ `max`\ altura\ t_2) \end{array}$$

Demostrar que para todo árbol finito $t :: Tree\ a$ se cumple que $hojas\ t < 2^{(altura\ t)}$

8. Dado el siguiente tipo de dato:

$$\begin{array}{l} \textbf{data}\ GTree\ a = E \mid N\ a\ [GTree\ a] \\ toList\ E = [\,] \\ toList\ (N\ x\ xs) = x : concat\ (map\ toList\ xs) \\ size\ E = 0 \\ size\ (N\ x\ xs) = 1 + (sum \circ map\ size)\ xs \end{array}$$

a) Enunciar el principio de inducción estructural para *Gtree*

b) Demostrar que $size\ t = length\ (toList\ t)\ \forall\ \ t :: GTree\ a$

9. Dadas las funciones $insert :: Ord\ a \Rightarrow a \to Bin\ a \to Bin\ a$, que agrega un elemento a un BST dado, y $inorder :: Ord\ a \Rightarrow Bin\ a \to [a]$, que realiza un recorrido inorder sobre un BST, dadas en clase de teoría, probar las siguientes propiedades sobre las funciones:

a) Si t es un BST, entonces $insert\ x\ t$ es un BST.

b) Si t es un BST entonces $inorder\ t$ es una lista ordenada.

10. Dadas las definiciones de funciones que implementan leftist heap, dadas en clase, probar que si *l1* y *l2* son leftist heaps, entonces *merge l1 l2* es un leftist heap.