Fundamentos de la Computación
Ejercicios de árboles

Cátedra de Teoría de la Computación

Junio de 2014

?1. Considere la siguiente definición del tipo (polimórfico) `AB a` de árboles con información de tipo `a` en las hojas:

```
data AB a where { Hoja :: a -> AB a ;
                  Nodo :: AB a -> AB a -> AB a}
```

Defina las siguientes funciones utilizando recursión estructural sobre AB:

1. `hojas::AB a -> [a]`, la cual, dado un árbol binario devuelve una lista con sus hojas.
2. `cantHojas::AB a -> N`, la cual, dado un árbol binario calcula la cantidad de hojas que tiene.
3. `cantNodos::AB a -> N`, la cual, dado un árbol binario calcula la cantidad de nodos internos que tiene.
4. `altura:: AB a -> N`, la cual, dado un árbol binario calcula su altura, es decir la longitud de alguna de sus ramas más largas.
5. `espejo:: AB a -> AB a`, la cual, dado un árbol binario devuelve el árbol espejo del mismo.
6. `maxHoja::Ord a => AB a -> a`, la cual, dado un árbol binario calcula el máximo de sus hojas.
7. `mapAB::(a->b) -> AB a -> AB b`, la cual, dado un árbol binario les aplica una función a todas sus hojas.
8. `subtrees::AB a -> [AB a]`, la cual, dado un árbol binario devuelve la lista de todos sus subárboles.

?2. Demuestre por inducción en AB las siguientes propiedades.:

1. $(\forall$ `t`$\Downarrow$`::AB a) cantHojas t = S(cantNodos t)`.

2. $(\forall$ `t`$\Downarrow$`::AB a)`$(\forall$ `f`$\Downarrow$`::a ->b`) cantNodos (map f t) = cantNodos t.

3. $(\forall$ `t`$\Downarrow$`::AB a) hojas (espejo t) = reverse (hojas t)`

?3. Considere ahora el tipo (polimórfico) `ABB a` de árboles con información de tipo `a` en los nodos:

```
data ABB a where { Vacio ::  ABB a ;
                   Unir  ::  ABB a -> a -> ABB a -> ABB a}
```

Defina las siguientes funciones utilizando recursión sobre ABB:

1. `inOrder::ABB a -> [a]`, la cual, dado un árbol binario lista sus elementos de izquierda a derecha: para cada nodo, aparecen primero en la lista todos los elementos que están a su izquierda, luego el nodo y después los que están a su derecha (siguiendo el mismo criterio).

2. `preOrder::ABB a -> [a]`, la cual, dado un árbol binario lista sus elementos del siguiente modo: para cada nodo, primero aparece el mismo, luego la lista todos los elementos que están a su izquierda y los que están a su derecha (siguiendo el mismo criterio).

3. `member::Eq a => a -> ABB a -> Bool` la cual, dado un árbol binario y un elemento verifica si este pertence al árbol.

4. `ordenado::Ord a=> ABB a -> Bool`, la cual, dado un árbol binario verifica si éste está ordenado, o sea, si cada nodo cumple la siguiente propiedad: todos los elementos que están a su izquierda son menores o iguales que él, y todos los que están a su derecha son mayores o iguales que él.

5. `insert::Ord a=> a -> ABB a -> ABB a` que inserta un elemento en un árbol ordenado, de modo tal que el árbol resultante también queda ordenado.

6. `list2ABB::Ord a=> [a] -> ABB` a que genere una árbol binario de búsqueda a partir de una lista, insertando los elementos uno por uno.

7. `treeSort::Ord a=> [a] -> [a]`, que ordena una lista insertando sus elementos en un árbol ordenado y listando sus nodos en el orden adecuado.

?4. Demuestre por inducción en ABB que:
`(∀t⇓::ABB a)(∀x⇓::a)(member x (insert x t) = True)`

?5. Para representar expresiones aritméticas simples se utiliza el siguiente tipo de árboles, donde las hojas son números naturales y los nodos internos se corresponden con las operaciones de suma y multiplicación:

```
data Exp where { Num :: N ->Exp;
                 Sum :: Exp -> Exp -> Exp;
                 Mul :: Exp -> Exp -> Exp }
```

1. Codificar la expresión $(2+3)*(-4*5)$ como un elemento de tipo Exp.

2. Programar una función `eval::Exp -> N` que calcule el valor de una expresión. Se podrán usar las operaciones correspondientes sobre naturales.

3. Programar una función `set::Exp -> N -> Exp` que reemplace todas las hojas de una expresión por un número natural dado, dejando el resto de la expresión sin cambiar.

4. Demuestre que para toda (∀e⇓::Exp) `eval(set e 0) = 0`