Programación Declarativa Aplicada – Curso 2016-17 - Práctica 2

Evaluación: Entrega en clase: 0.30. Entrega la siguiente semana: 0.10 (más adelante no puntúa)

Escribir un módulo “pract2” con las siguientes funciones.
Nota: Las funciones deben exportarse explícitamente con la directiva -export([….]).

1) impares(X) : lista con los elementos que ocupan posiciones impares en la lista X.

Para probar:
pract2:impares([a,b,c,d,e,f]). %da [a,c,e]
pract2:impares([a,b]). %da [a]
pract2:impares([]). %Da []

2) Definir la función de Ackermann:

$$A(m,n) = \begin{cases} n+1, & \text{si } m = 0; \\ A(m-1,1), & \text{si } m > 0 \text{ y } n = 0; \\ A(m-1,A(m,n-1)), & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

Para probar: pract2:ack(3,4). da 125.

3) Vamos a representar conjuntos mediante listas. Escribir funciones:

3.1) mismoConjunto(X,Y) que indique si dos listas representan el mismo conjunto, es decir si tienen los mismos elementos sin importar repeticiones ni orden.
Idea: usar una función auxiliar contenido(X,Y) que indique si todos los elementos de X están en Y.
Para probar:
pract2:mismoConjunto([1,3,4,4],[4,3,3,1,1]). da true
pract2:mismoConjunto([1,3,4,4],[4,3,3]). da false (1 no está en el segundo conjunto)

3.2) *normal(X)* que devuelve una lista que representa el mismo conjunto que X pero en el que, como lista, no contiene valores repetidos.

3.3) *intersection(X,Y)* que devuelve la lista de elementos comunes entre X e Y (no importa si hay repetidos o no).

4) Definimos árboles binarios de la siguiente forma:
- {} es un árbol binario
- {elem, a1, a2} es un árbol binario, con elem el valor que se almacena en el nodo y a1, a2 árboles binarios.
Supongamos que el parámetro X representa un árbol binario cuyos elementos son enteros. Definir:

4.1) está(E,X) = true si el elemento E está en el árbol X; false en caso contrario.
Para probar:
pract2:está(3,{5,{3,{},{}}, {6,{},{}}}). da true. pract2:está(3, {6,{13,{},{}}, {6,{},{}}}). da false

4.2) nNodos(X) = núm. de nodos en el árbol. pract2:nNodos({5,{3,{},{}}, {6,{},{7,{},{}}}}). %4

4.3) mapTree(F,X) = devuelve el árbol que se obtiene al aplicar la función F a todos los elementos de X.

Para probar:
pract2:mapTree((fun(E)->E+1 end), {6, {5, {3,{},{}},{}},{}}). da {7,{6,{4,{},{}},{}},{}}

F = fun(E) when E rem 2 ==0 -> par;
(E) when E rem 2==1 -> impar end.
pract2:mapTree(F, {6, {5, {3,{},{}}, {4,{},{}} }, {6, {}, {8, {}, {}}}}).
da:
{par,{impar,{impar,{},{}},{par,{},{}}},{par,{},{par,{},{}}}}

5) sonMúltiplos(X,Y) : true si X e Y son enteros múltiplos o false en otro caso.

Nota 1: todo número es múltiplo de 0.
Pista: Utilizar la función infija *rem* que devuelve el resto de la divisón.

Para probar:
pract2:sonMúltiplos(0,0). da true, pract2:sonMultiplos(0,4). da true
pract2:sonMúltiplos(5,4). da false, pract2:sonMultiplos(8,4). da true
pract2:sonMúltiplos(4,8). da true

6) h(X) = devuelve la función función g(Y) = true si Y es múltiplo de X; false en otro caso

Nota 1: las funciones anónimas pueden definirse también en varias reglas e incluso tener guardas:

```
F = fun
  (N) when N =:= 42 -> true;
  (N) -> false
end.
```

Nota 2: La función del ejercicio 9 no se usa explícitamente aquí, solo se usa porque el código es muy similar

Para probar:

```
A = pract2:h(8), A(16).  da true

A = pract2:h(8), A(15).  da false

A = pract2:h(8), A(0).  da true

(pract2:h(8))(4). da false
```