**Practica 2**

Memoria de la practica de Programación Declarativa: Lógica y restricciones

**Miembros:**

**Parra Garcia, Alejandro Carmelo, Y16I028**

**Vladimirov Stoyanov, Petko, Y16I019**

**Revuelta Martinez, Alvaro, Y16I009**

1. **CODIGO:**

alumno\_prode('Parra','Garcia','Alejandro Carmelo','Y16I028').

alumno\_prode('Vladimirov','Stoyanov','Petko','Y16I019').

alumno\_prode('Revuelta','Martinez','Alvaro','Y16I009').

%---------------------------------------------------------------------

**%Parte 1**

%Menor crea una estructura X de aridad 2 con el argumento Comp como functor, e introduce

%A y B como argumentos de la estructura para hacer un call y así poder evaluar la expresión

menor(A,B,Comp,M):-

functor(X,Comp,2),

arg(1,X,A),

arg(2,X,B),

(call(X) -> M=A; M=B ).

%---------------------------------------------------------------------

%**Parte 2**

%Caso base en el que si A o B son variables libres es true,

%pues "Una variable libre es igual a cualquier otro termino"

menor\_o\_igual(A,B):-

var(A);

var(B).

%Usamos functor/3 para obtener el termino y la ariedad tanto de A como de B

%Comparamos el termino de A y de B, Si el de A es menor que el de B ya se cumple

%la condicion y por tanto es TRUE. Por el contrario si no es menor miramos si

%son iguales, si es asi tenemos que comparar la ariedad. Si la ariedad de A es

%menor que la ariedad de B se cumple que A es menor\_o\_igual de B. Si la ariedad de A

%no es menor que el de B, comparamos si es igual. Si ambas ariedades son iguales

%llamamos a menor\_o\_igual\_aux/3.

%En el caso que:

%A es menor que B devolvemos TRUE.

%A es igual a B, y la ariedad de A es menor que la de B, devolvemos TRUE.

%A es igual a B, y la ariedad de A y la de B son iguales, llamamos a menor\_o\_igual\_aux/2.

menor\_o\_igual(A,B):-

functor(A,NA,\_),

functor(B,NB,\_),

NA @< NB. %NA @< NB no hace falta evaluar el resto.

menor\_o\_igual(A,B):-

functor(A,NA,La),

functor(B,NB,Lb),

NA == NB, %si son iguales evaluo la airedad

(

La < Lb; %al ser un or si La < Lb no evalua el resto

(

La =:= Lb, %si la ariedad es la misma llamo

(

menor\_o\_igual\_aux(A,B,1) %a menor\_o\_igual\_aux/2.

)

)

).

%Predicado auxiliar al cual solo se le llama si dos terminos son iguales y con la misma ariedad

%Este predicado va elemento por elemento comparandolos. Si el de A es menor que el de B ya cumple

%la condicion(segun lo especificado en el enunciado). si son iguales compruebo el siguiente termino,

%En cualquier otro caso es false

%Si la ariedad de A es 0 significa que ya hemos evaluado todos los terminos y son iguales por tanto A es igual a B y TRUE

menor\_o\_igual\_aux(A,\_,N):-

functor(A,\_,La),

N1 is N-1,

La =:= N1.

%Sacamos el termino N de A y de B, si son iguales hago recursividad avanzando N para comparar el siguiente termino.

menor\_o\_igual\_aux(A,B,N):-

arg(N,A,ElemA),

arg(N,B,ElemB),

soy\_igual(ElemA,ElemB),!,

N1 is N+1,

menor\_o\_igual\_aux(A,B,N1).

%En este caso sabemos que el termino N de A y B no es igual, por lo que si es menor devolvemos true y no hace falta comparar mas.

menor\_o\_igual\_aux(A,B,N):-

arg(N,A,ElemA),

arg(N,B,ElemB),

menor\_o\_igual(ElemA,ElemB).

%Predicado auxiliar que se usa para evaluar si dos terminos son iguales

%Dos terminos son iguales si alguno de ellos es variable libre.

%Dos terminos son iguales si los nombres de ambos son identicos, tienen la misma ariedad y sus argumentos son identicos.

soy\_igual(A,B):-

var(A);

var(B).

%Miramos que los nombres y la ariedad de A y B sean iguales, si es asi llamo a soy/igual\_aux/3 que avalua cada argumento.

soy\_igual(A,B):-

functor(A,NA,La),

functor(B,NB,Lb),

NA == NB,

La =:= Lb,

soy\_igual\_aux(A,B,1).

%caso base en el que ya se han comparado todos los argumentos, por lo tanto son iguales

%La ariedad de A es 0

soy\_igual\_aux(A,\_,N):-

functor(A,\_,La),

N1 is N-1,

La =:= N1.

%Sacamos el argumento N de A y B y miramos si son iguales llamando a soy\_igual/2

%si son iguales avanzamos N y hacemos recursividad

soy\_igual\_aux(A,B,N):-

arg(N,A,ElemA),

arg(N,B,ElemB),

soy\_igual(ElemA,ElemB),

N1 is N+1,

soy\_igual\_aux(A,B,N1).

%---------------------------------------------------------------------

%**Parte 3**

%%%%%%%%%%%%%%%%%%%

%%%lista\_hojas/2%%%

%%%%%%%%%%%%%%%%%%%

lista\_hojas([],[]).

lista\_hojas([H|L],[tree(H,void,void)|HOJAS]):-

lista\_hojas(L,HOJAS).

%%%%%%%%%%%%%%%%%%%

%%%hojas\_arbol/3%%%

%%%%%%%%%%%%%%%%%%%

%Si esta vacio la lisa de hojas no hay arbol.

hojas\_arbol([],\_,\_).

%Si solo hay una hoja, esa es el arbol

hojas\_arbol([X],\_,X).

%Si hay mas hojas se llama a hojas\_arbol\_aux/4 para que genere el arbol

hojas\_arbol(Lista,Comp,Arbol):-

hojas\_arbol\_aux(Lista,[],Comp,Arbol).

%El primer termino es una lista con las hojas que quedan por fusionarse, una vez se fususionan se pasan a la

%segunda lista, cuando la primera lista esta vacia se pasan todos los elementos de la segunda a la primera y

%se vacia la segunda. Si queda 1 elemento en la primera lista este se pone al final de la segunda y despues se

%pasa la segunda lista a la primera y se vacia la segunda.

%El proceso se repite hasta que solo quede un elemento en la primera y la segunda lista este vacia, que significa que ya esta el arbol creado

%Si solo queda 1 hoja, y la segunda esta vacia, se devuelve esa hoja como el arbol

hojas\_arbol\_aux([X],[],\_,X).

%Si la primera lista esta vacia se vuelca el contenido de la segunda en la primera y se vacia la segunda. Y se vuelve a llamar a hojas\_arbol\_aux/4

hojas\_arbol\_aux([],Lista,Comp,Arbol):-

hojas\_arbol\_aux(Lista,[],Comp,Arbol).

%Si solo queda 1 elemento en la primera lista se añade al final de la segunda. Posteriormente la segunda lista se vuelca en la primera y se vacia la segunda.

%Y se vuelve a llamar a hojas\_arbol\_aux/4

hojas\_arbol\_aux([X],Lista,Comp,Arbol):-

insertar(Lista,X,Lista1),

hojas\_arbol\_aux(Lista1,[],Comp,Arbol).

%si hay mas elementos se sacan los dos priemos y se construye un arbol con ellos. Despues se añaden al final de la segunda lista.

%Se vuelve a llamar a hojas\_arbol\_aux/4, pero en la primera lista ya no estan los dos primeros elementos.

hojas\_arbol\_aux([tree(E\_A,H1\_A,H2\_A),tree(E\_B,H1\_B,H2\_B)|Hojas],Lista,Comp,Arbol):-

menor(E\_A,E\_B,Comp,M),

insertar(Lista,tree(M,tree(E\_A,H1\_A,H2\_A),tree(E\_B,H1\_B,H2\_B)),Lista1),

hojas\_arbol\_aux(Hojas,Lista1,Comp,Arbol).

%Inserta un Item al final de una lista y lo devuelve en Solucion.

insertar(Lista,Item,Solucion):-

length(Lista,X),

X =:= 0,

Solucion = [Item].

insertar([E|Lista],Item,[E|Solucion]):-

insertar(Lista,Item,Solucion).

%%%%%%%%%%%%%%%%%%

%%%ordenacion/3%%%

%%%%%%%%%%%%%%%%%%

%Predicado que genera una lista ordenada a partir de un arbol

%Si el arbol esta vacio se devuelve una lista vacia

ordenacion(void,\_,[]).

%Llamamos a reflotar/3 que dado un arbol y un Comp genera un arbol reflotado.

%despues hacemos recursividad, con el nuevo arbol. Vamos añadiendo los elemntos a la lista a la que volvemos

ordenacion(tree(N,I,D), Comp, [N|Orden]):-

reflotar(tree(N,I,D),Comp,ArbolReflotado),

ordenacion(ArbolReflotado, Comp, Orden).

%predicado auxiliar que usamos para relotar un arbol

%Caso base en el que solo hay una hoja en el arbol, el arbol resultado es void.

reflotar(tree(\_,void,void),\_,X):-

X=void.

%Casos en los que uno de los hijos es la hoja con el elemnto.

%En este caso se devuelve el otro hijo, que puede ser una sola hoja o un arbol.

reflotar(tree(E,tree(E,void,void),tree(E2,I2,D2)),\_,X):-

X = tree(E2,I2,D2).

reflotar(tree(E,tree(E1,I1,D1),tree(E,void,void)),\_,X):-

X = tree(E1,I1,D1).

%Se distingue dos casos, en el cual el nodo con la hoja esta en un lado o en otro.

%Una vez sabemos por que hijo se encuentra el nodo hoja llamams a reflotar con ese hijo (que puede ser una sola hoja o un arbol)

%Creamos un nuevo arbol con el otro hijo y el arbol que nos ha devuelto la llamada a reflorar y ponemos de elemento de la raiz de este

%nuevo arbol el elemento que nos devuelve menor entre el otro hijo y el del nuevo arbol.

reflotar(tree(E,tree(E1,I1,D1),tree(E2,I2,D2)),Comp,X):-

(E == E1 ->

(reflotar(tree(E1,I1,D1),Comp,tree(X1Elem,XI1,XD1)),

menor(E2,X1Elem,Comp,M),

X=tree(M,tree(X1Elem,XI1,XD1),tree(E2,I2,D2))

);

(reflotar(tree(E2,I2,D2),Comp,tree(X2Elem,XI2,XD2)),

menor(E1,X2Elem,Comp,M),

X=tree(M,tree(E1,I1,D1),tree(X2Elem,XI2,XD2))

)

).

%%%%%%%%%%%%%%%%%%%

%%%%%ordenar/3%%%%%

%%%%%%%%%%%%%%%%%%%

%Dada una Lista y un Comp nos devuelve una lista ordenada en Orden

%Hace la llamada a lista\_hojas/2 que nos crea unas hojas en funcion de los elementos de la lista

%llama a hojas\_arbol, con la lista de hojas previamente creadas y el comparador, esta llamada nos devuelve un arbol flotante

%Llamada a ordenacion con el Arbol y el Comp, este predicado nos genera una lista ordenada a partir del Arbol

ordenar(Lista,Comp,Orden):-

lista\_hojas(Lista,Hojas),

hojas\_arbol(Hojas,Comp,Arbol),

ordenacion(Arbol,Comp,Orden).

----------------------------------------------------------------------

1. **Pruebas:**

*Parte 1:*

Pruebas de ***menor***:

?- menor(3,5,<,M).

M = 3 ?

yes

?- menor(5,3,<,M).

M = 3 ?

yes

?- menor(5,3,>,M).

M = 5 ?

yes

?- menor(5,3,>,3).

no

?- menor(5,3,>,5).

yes

?- menor(5,3,>=,4).

no

?- menor(5,5,=,M).

M = 5 ?

yes

?- menor(5,5,=,M).

M = 5 ? ;

no

?- menor(5,4,=<,M).

M = 4 ?

yes

?- menor(5,4,=<,4).

Yes

*Parte 2:*

Pruebas de ***menor\_o\_igual***:

?- menor\_o\_igual(p(X,m),p(b,e)).

no

?- menor\_o\_igual(p(q(a)),p(q(a))).

yes

?- menor\_o\_igual(p(q(a)),p(q(b))).

yes

?- menor\_o\_igual(p(q(c)),p(q(b))).

no

?- menor\_o\_igual(p(q(X)),p(q(b))).

yes

?- menor\_o\_igual(p(q(a,b)),p(q(a,b))).

yes

?- menor\_o\_igual(p(q(a,b)),p(q(a,a))).

no

?- menor\_o\_igual(p(q(a,b)),p(q(a,c))).

yes

?- menor\_o\_igual(p(a,X),p(Y,b)).

yes

?- menor\_o\_igual(p(...),q(...)).

yes

?- menor\_o\_igual(p(a,b,c),p(a,a,a,a)).

yes

?- menor\_o\_igual(p(a,b,c),p(a,a,a,a)).

yes

?- menor\_o\_igual(p(d,b,c),p(a,a,a,a)).

yes

?- menor\_o\_igual(p(a,X,c,a),p(a,b,c,d)).

yes

?- menor\_o\_igual(p(q(r(a))),p(r(a))).

yes

?- menor\_o\_igual(p(X),p(a)).

yes

?- menor\_o\_igual(p(a,b(c,m)),p(a,b(c,e))).

no

?- menor\_o\_igual(p(a,b(c,X)),p(a,b(c,e))).

yes

?- menor\_o\_igual(p(a,b(c,a)),p(a,b(c,e))).

yes

?- menor\_o\_igual(p(a,b(c,m,a)),p(a,b(c,e))).

no

?- menor\_o\_igual(p(a,b(c,d)),p(a,b(c,e))).

yes

?- menor\_o\_igual(p(a,b(c(X,4),d)),p(a,b(c(1,Y),d))).

yes

?- menor\_o\_igual(p(a,b(c(X,4),d)),p(a,b(c(1,5),d))).

yes

?- menor\_o\_igual(p(a,b(c(X,4),d)),p(a,b(c(1,4),d))).

yes

?- menor\_o\_igual(p(a,b(c(X,4),d)),p(a,b(c(1,3),d))).

no

?- menor\_o\_igual(p(a,b(c(X,4),d)),p(a,b(c(1,3,4),d))).

yes

Pruebas del predicado auxiliar ***soy\_igual***:

?- soy\_igual(5,5).

yes

?- soy\_igual(A,B).

yes

?- soy\_igual(4,5).

no

?- soy\_igual(A,5) .

yes

?- soy\_igual(p(q(c)),p(q(b))).

no

?- soy\_igual(p(q(b)),p(q(b))).

yes

?- soy\_igual(p(...),q(...)).

no

?- soy\_igual(p(...),r(...)).

no

?- soy\_igual(r(...),r(...)).

yes

?- soy\_igual(p(a,b(c(X,4),d)),p(a,b(c(1,4),d))).

yes

?- soy\_igual(p(a,b(c,m)),p(a,b(c,e))).

no

?- soy\_igual(p(a,b(c,e)),p(a,b(c,e))).

yes

?- soy\_igual(p(a,b(B,A)),p(a,b(c,e))).

yes

?- soy\_igual(p(a,b(B,A)),p(a,b(D,E))).

yes

?- soy\_igual(p(a,b(5,6)),p(a,b(6,5))).

no

*Parte 3:*

Pruebas de los predicados principales

Pruebas de ***lista\_hojas***:

?- lista\_hojas([1,2,3],Hojas).

Hojas = [tree(1,void,void),tree(2,void,void),tree(3,void,void)] ?

yes

?-lista\_hojas([1,2,3,4],Hojas).

Hojas = [tree(1,void,void),tree(2,void,void),tree(3,void,void),tree(4,void,void)] ?

yes

?- lista\_hojas([1,2,A,4],Hojas).

Hojas = [tree(1,void,void),tree(2,void,void),tree(A,void,void),tree(4,void,void)] ?

yes

?- lista\_hojas([A,C,A,E],Hojas).

Hojas = [tree(A,void,void),tree(C,void,void),tree(A,void,void),tree(E,void,void)] ?

yes

?- lista\_hojas([5,7,5,5,4,7],Hojas).

Hojas = [tree(5,void,void),tree(7,void,void),tree(5,void,void),tree(5,void,void),tree(4,void,void),tree(7,void,void)] ?

yes

?- lista\_hojas([p,q,e,t,l,s],Hojas).

Hojas = [tree(p,void,void),tree(q,void,void),tree(e,void,void),tree(t,void,void),tree(l,void,void),tree(s,void,void)] ?

yes

?- lista\_hojas([4,2,3,1,5],Hojas).

Hojas = [tree(4,void,void),tree(2,void,void),tree(3,void,void),tree(1,void,void),tree(5,void,void)] ?

yes

?- lista\_hojas([4,4,7,7,7],Hojas).

Hojas = [tree(4,void,void),tree(4,void,void),tree(7,void,void),tree(7,void,void),tree(7,void,void)] ?

yes

?- lista\_hojas([A,P,L,Y,W],Hojas).

Hojas = [tree(A,void,void),tree(P,void,void),tree(L,void,void),tree(Y,void,void),tree(W,void,void)] ?

yes

?- lista\_hojas([q(2),t(3),q(6),q(8),t(3)],Hojas).

Hojas = [tree(q(2),void,void),tree(t(3),void,void),tree(q(6),void,void),tree(q(8),void,void),tree(t(3),void,void)] ?

yes

?- lista\_hojas([q(2,5),t(3),q(6),q(8,3),t(3)],Hojas).

Hojas = [tree(q(2,5),void,void),tree(t(3),void,void),tree(q(6),void,void),tree(q(8,3),void,void),tree(t(3),void,void)] ?

yes

?- lista\_hojas([q(2,5),t(3),q(6,6),q(8,3),t(3)],Hojas).

Hojas = [tree(q(2,5),void,void),tree(t(3),void,void),tree(q(6,6),void,void),tree(q(8,3),void,void),tree(t(3),void,void)] ?

yes

?- lista\_hojas([q(2,5),B,q(6,6),q(8,3),D],Hojas).

Hojas = [tree(q(2,5),void,void),tree(B,void,void),tree(q(6,6),void,void),tree(q(8,3),void,void),tree(D,void,void)] ?

yes

?- lista\_hojas([u],Hojas).

Hojas = [tree(u,void,void)] ?

yes

?- lista\_hojas([PPPP],Hojas).

Hojas = [tree(PPPP,void,void)] ?

yes

?- lista\_hojas([],Hojas).

Hojas = [] ?

?- lista\_hojas([13],Hojas).

Hojas = [tree(13,void,void)] ?

yes

?- lista\_hojas([aa],Hojas).

Hojas = [tree(aa,void,void)] ?

yes

Pruebas de ***hojas\_arbol:***

?- hojas\_arbol([tree(1,void,void),tree(2,void,void),tree(3,void,void)],=<,X).

X = tree(1,

tree(1,

tree(1,void,void),

tree(2,void,void)),

tree(3,void,void)) ?

yes

?- hojas\_arbol([tree(4,void,void), tree(2,void,void), tree(3,void,void),tree(1,void,void), tree(5,void,void)],=<,X).

X = tree(1,

tree(1,

tree(2,

tree(4,void,void),

tree(2,void,void)),

tree(1,

tree(3,void,void),

tree(1,void,void))),

tree(5,void,void)) ?

yes

?- hojas\_arbol([tree(1,void,void), tree(2,void,void), tree(3,void,void),tree(4,void,void), tree(5,void,void),tree(6,void,void)],>,X).

X = tree(6,

tree(4,

tree(2,

tree(1,void,void),

tree(2,void,void)),

tree(4,

tree(3,void,void),

tree(4,void,void))),

tree(6,

tree(5,void,void),

tree(6,void,void))) ?

yes

?- hojas\_arbol([tree(1,void,void), tree(1,void,void), tree(1,void,void),tree(2,void,void) ],>=,X).

X = tree(2,

tree(1,

tree(1,void,void),

tree(1,void,void)),

tree(2,

tree(1,void,void),

tree(2,void,void))) ?

yes

?- hojas\_arbol([tree(4,void,void), tree(5,void,void), tree(3,void,void),tree(2,void,void) ],<,X).

X = tree(2,

tree(4,

tree(4,void,void),

tree(5,void,void)),

tree(2,

tree(3,void,void),

tree(2,void,void))) ?

yes

?- hojas\_arbol([tree(2,void,void), tree(8,void,void), tree(4,void,void),tree(2,void,void),tree(5,void,void),tree(6,void,void) ],>=,X).

X = tree(8,

tree(8,

tree(8,

tree(2,void,void),

tree(8,void,void)),

tree(4,

tree(4,void,void),

tree(2,void,void))),

tree(6,

tree(5,void,void),

tree(6,void,void))) ?

yes

?- hojas\_arbol([tree(2,void, void),tree(1,void,void), tree(2,void,void)],<,X).

X = tree(1,

tree(1,

tree(2,void,void),

tree(1,void,void)

),

tree(2,void,void)) ?

yes

?- hojas\_arbol([tree(2,void, void),tree(1,void,void), tree(2,void,void)],>,X).

X = tree(2,

tree(2,

tree(2,void,void),

tree(1,void,void)

),

tree(2,void,void)) ?

yes

?- hojas\_arbol([tree(1,void, void),tree(3,void,void), tree(5,void,void), tree(7,void,void), tree(9,void,void)],<,X).

X = tree(1,

tree(1,

tree(1,

tree(1,void,void),

tree(3,void,void)

),

tree(5,

tree(5,void,void),

tree(7,void,void)

)

),

tree(9,void,void)) ?

yes

?- hojas\_arbol([tree(1,void, void),tree(3,void,void), tree(5,void,void), tree(7,void,void), tree(9,void,void), tree(9,void,void), tree(8,void,void)],>,X).

X = tree(9,

tree(7,

tree(3,

tree(1,void,void),

tree(3,void,void)

),

tree(7,

tree(5,void,void),

tree(7,void,void)

)

),

tree(9,

tree(9,

tree(9,void,void),

tree(9,void,void)

),

tree(8,void,void))) ?

yes

Pruebas de ***ordenación:***

?- ordenacion(tree(1,tree(2,tree(2,void,void),tree(8,void,void)),tree(1,tree(1,void,void),tree(10,void,void))),=<,X).

X = [1,2,8,10] ?

yes

?- ordenacion(tree(1,tree(1,tree(1,tree(1,void,void),tree(2,void,void)),tree(3,tree(3,void,void),tree(4,void,void))),

tree(5,tree(5,tree(5,void,void),tree(6,void,void)),tree(7,tree(7,void,void),tree(8,void,void)))),=<,X).

X = [1,2,3,4,5,6,7,8] ?

yes

?- ordenacion(tree(1,tree(1,tree(2,tree(4,void,void),tree(2,void,void)),tree(1,tree(3,void,void),tree(1,void,void))),tree(5,void,void)),=<,X).

X = [1,2,3,4,5] ?

yes

?- ordenacion(tree(1,void,void), <, X).

X = [1] ?

yes

?- ordenacion(tree(1,tree(1,tree(2,void,void),tree(1,void,void)),tree(2,void,void)),<,X).

X = [1,2,2] ?

yes

?- ordenacion(tree(2,tree(2,tree(2,void,void),tree(1,void,void)),tree(2,void,void)),>,X).

X = [2,2,1] ?

yes

?- ordenacion(tree(1,tree(1,tree(1,tree(1,void,void),tree(3,void,void)),tree(5,tree(5,void,void),tree(7,void,void))),tree(9,void,void)),<,X).

X = [1,3,5,7,9] ?

yes

?- ordenacion(tree(9,tree(7,tree(3,tree(1,void,void),tree(3,void,void)),tree(7,tree(5,void,void),tree(7,void,void))),tree(9,tree(9,tree(9,void,void),tree(9,void,void)),tree(8,void,void))),>,X).

X = [9,9,8,7,5,3,1] ?

yes

Pruebas de ***ordenar***:

?- ordenar([3,7,1,4,9], =<, X) .

X = [1,3,4,7,9] ?

yes

?- ordenar([3,7,4,8,1,4,8,3,9], >, X) .

X = [9,8,8,7,4,4,3,3,1] ?

yes

?- ordenar([1,2,3,4,5,6], >, X).

X = [6,5,4,3,2,1] ?

yes

?- ordenar([a,c,g,r,u,p,a,r,r,a], @<, X).

X = [a,a,a,c,g,p,r,r,r,u] ?

yes

?- ordenar([aad,c,baa,rti,adc,apu,eme,abba,acab,lsd], @>, X).

X = [rti,lsd,eme,c,baa,apu,adc,acab,abba,aad] ?

yes

?- ordenar([p(q(a,b)),p(q(a,c)),p(a,b(c,a)),p(a,b(c,e)),p(q(a)),p(q(b))],menor\_o\_igual,X).

X = [p(q(a)),p(q(b)),p(q(a,b)),p(q(a,c)),p(a,b(c,a)),p(a,b(c,e))] ?

yes

?-ordenar([p(q(a,b)),p(q(a,c)),p(a,b(c,Y)),p(a,b(c,e)),p(q(a)),p(q(b))],menor\_o\_igual,X).

X = [p(q(a)),p(q(b)),p(q(a,b)),p(q(a,c)),p(a,b(c,Y)),p(a,b(c,e))] ?

yes

?- ordenar([p(q(a,b)),p(q(a,c)),p(a,b(c,a)),p(a,b(c,Z)),p(q(a)),p(q(b))],menor\_o\_igual,X).

X = [p(q(a)),p(q(b)),p(q(a,b)),p(q(a,c)),p(a,b(c,a)),p(a,b(c,Z))] ?

yes

Pruebas de los predicados auxiliares

Pruebas de ***insertar***:

?- insertar([a,b,c,d], e,X).

X = [a,b,c,d,e] ?

yes

?- insertar([q(a),q(b),q(c),q(d)], q(e),X).

X = [q(a),q(b),q(c),q(d),q(e)] ?

yes

?- insertar([1,1,1,1], 2,X).

X = [1,1,1,1,2] ?

yes

?- insertar([1,2,3,4,5,6,7], 3,X).

X = [1,2,3,4,5,6,7,3] ?

yes

?- insertar([a,aa,aaa,aaaa,aaa,aa], b,X).

X = [a,aa,aaa,aaaa,aaa,aa,b] ?

yes

?- insertar([1,1,1,4,8,9],22,X).

X = [1,1,1,4,8,9,22] ?

yes

?- insertar([1,1,1,1,1,1,1],q(a),X).

X = [1,1,1,1,1,1,1,q(a)] ?

yes

?- insertar([3,2,3],1,X).

X = [3,2,3,1] ?

yes

?- insertar([3,2,3],5,X).

X = [3,2,3,5] ?

yes

Pruebas de ***reflotar***:

?- reflotar(tree(1,tree(1,tree(1,tree(1,void,void),tree(2,void,void)),tree(3,tree(3,void,void),tree(4,void,void))),

tree(5,tree(5,tree(5,void,void),tree(6,void,void)),tree(7,tree(7,void,void),tree(8,void,void)))),=<,X).

X =

tree(2,

tree(2,

tree(2,void,void),

tree(3,

tree(3,void,void),

tree(4,void,void)

)

),

tree(5,

tree(5,

tree(5,void,void),

tree(6,void,void)

),

tree(7,

tree(7,void,void),

tree(8,void,void)

)

)

) ?

yes

?- reflotar(tree(2,tree(2,tree(2,void,void),tree(3,tree(3,void,void),tree(4,void,void))),

tree(5,tree(5,tree(5,void,void),tree(6,void,void)),tree(7,tree(7,void,void),tree(8,void,void)))),=<,X).

X =

tree(3,

tree(3,

tree(3,void,void),

tree(4,void,void)

),

tree(5,

tree(5,

tree(5,void,void),

tree(6,void,void)

),

tree(7,

tree(7,void,void),

tree(8,void,void)

)

)

) ?

yes

?- reflotar(tree(3,tree(3,tree(3,void,void),tree(4,void,void)),tree(5,tree(5,tree(5,void,void),tree(6,void,void)),tree(7,tree(7,void,void),tree(8,void,void)))),=<,X).

X =

tree(4,

tree(4,void,void),

tree(5,

tree(5,

tree(5,void,void),

tree(6,void,void)

),

tree(7,

tree(7,void,void),

tree(8,void,void)

)

)

) ?

yes

?- reflotar(tree(4,tree(4,void,void),tree(5,tree(5,tree(5,void,void),tree(6,void,void)),tree(7,tree(7,void,void),tree(8,void,void)))),=<,X).

X =

tree(5,

tree(5,

tree(5,void,void),

tree(6,void,void)

),

tree(7,

tree(7,void,void),

tree(8,void,void)

)

) ?

yes

?- reflotar(tree(5,tree(5,tree(5,void,void),tree(6,void,void)),tree(7,tree(7,void,void),tree(8,void,void))),=<,X).

X =

tree(6,

tree(6,void,void),

tree(7,

tree(7,void,void),

tree(8,void,void)

)

) ?

yes

?- reflotar(tree(6,tree(6,void,void),tree(7,tree(7,void,void),tree(8,void,void))),=<,X).

X =

tree(7,

tree(7,void,void),

tree(8,void,void)

) ?

yes

?- reflotar(tree(7,tree(7,void,void),tree(8,void,void)),=<,X).

X = tree(8,void,void) ?

yes

?- reflotar(tree(8,void,void),=<,X).

X = void ?

yes

?- reflotar(tree(1,tree(2,tree(2,void,void),tree(8,void,void)),tree(1,tree(1,void,void),tree(10,void,void))),=<,X).

X =

tree(2,

tree(2,

tree(2,void,void),

tree(8,void,void)

),

tree(10,void,void)

) ?

yes