Ejercicios de Programación Declarativa

Curso 2020/21

Hoja 7

1. Escribe una versión del predicado *sumintersec*/3 utilizando la notación `(P->Q;R)`.

   sumintersec$(L_1, L_2, N) \leftrightarrow L_1$ y L_2 son dos listas de enteros, ordenadas de menor a mayor y N es la suma de los elementos que están en la intersección de las dos listas.

```
sumintersec([],L,0).
sumintersec(L,[],0).
sumintersec([X|L1],[X|L2],N) :- !, sumintersec(L1,L2,N1), N is N1 + X.
sumintersec([X|L1],[Y|L2],N) :-
      (X > Y -> sumintersec([X|L1],L2,N);sumintersec(L1,[Y|L2],N)).
```

2. *Implementación de conjuntos usando listas.* Escribe los siguientes predicados en Prolog:

   (a) *nomiembro*$(X, L) \leftrightarrow X$ no es un elemento de la lista L.

```
nomiembro(X,[]) :- !
nomiembro(X,[Y|Xs]) :-X \= Y,nomiembro(X,Xs).
```

   (b) *hazconjunto*$(L, C) \leftrightarrow C$ es un conjunto, representado por medio de una lista, que tiene los mismos elementos que L (no importa en que orden, pero sin que ninguno esté repetido.

```
hazconjunto([],[]) :- !.
hazconjunto([X|Xs],[X|C]) :- nomiembro(X,Xs), !, hazconjunto(Xs,C).
hazconjunto([X|Xs],C) :- hazconjunto(Xs,C).
```

   (c) *union*$(C_1, C_2, C) \leftrightarrow C$ es el conjunto formado por los elementos de la unión de los conjuntos C_1 y C_2.

```
union([],C,C) :- !.
union([X|C1],C2,[X|C]) :- nomiembro(X,C2), !, union(C1,C2,C).
union([X|C1],C2,C) :- union(C1,C2,C).
```

   (d) *interseccion*$(C_1, C_2, C) \leftrightarrow C$ es el conjunto formado por los elementos de la intersección de los conjuntos C_1 y C_2.

```
interseccion([],C,[]) :- !.
interseccion([X|C1],C2,[X|C]) :- member(X,C2), !, interseccion(C1,C2,C).
interseccion([X|C1],C2,C) :- interseccion(C1,C2,C).
```

3. Define en Prolog un predicado *treeSort*(Xs, XsO), para implementar el algoritmo de ordenación de listas utilizando un árbol ordenado, que ya programaste en Haskell.

   treeSort$(Xs, XsO) \longleftrightarrow XsO$ es la lista resultante de ordenar la lista Xs utilizando un árbol ordenado.

```
treeSort(Xs,XsO) :- listArbolOrd(Xs,AO),inorden(AO,XsO).

listArbolOrd([],void ) :- !.
listArbolOrd([X|Xs],AO) :- listArbolOrd(Xs,A),insertOrd(X,A,AO).

insertOrd(X,void,arbol(X,void,void)) :- !.
insertOrd(X,arbol(Y,HI,HD),arbol(Y,NHI,HD)) :- X @< Y, !, insertOrd(X,HI,NHI).
insertOrd(X,arbol(Y,HI,HD),arbol(Y,HI,NHD)) :- insertOrd(X,HD,NHD).
```

4. Sean A_1 y A_2 dos árboles binarios ordenados. Define un predicado para determinar si A_1 está contenido en A_2. Es decir, cada nodo de A_1 es un nodo de A_2. Haz uso de que los árboles están ordenados.

```
contenido(void,A) :- !.
contenido(arbol(X,HI1,HD1),arbol(X,HI2,HD2)) :- !,
    contenido(HI1,HI2),
    contenido(HD1,HD2).
contenido(arbol(X,HI1,HD1),arbol(Y,HI2,HD2)) :-
    X @> Y, !,
    contenido(arbol(X,void,HD1),HD2),
    contenido(HI1,arbol(Y,HI2,HD2)).
contenido(arbol(X,HI1,HD1),arbol(Y,HI2,HD2)) :-
    contenido(arbol(X,HI1,void),HI2),
    contenido(HD1,arbol(Y,HI2,HD2)).
```

5. Programa en Prolog los siguientes predicados utilizando la aritmética de Prolog y el predicado del corte: $polinomio(E, X) \leftrightarrow$ la expresión E es un polinomio con incógnita X.

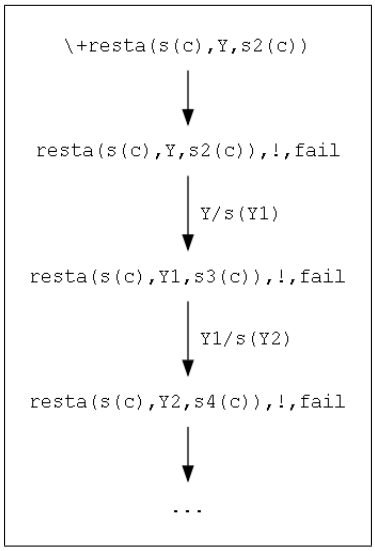
```
% utilizamos el hecho de que el operador + asocia por la izquierda
polinomio(P+M,X) :- !,
    polinomio(P,X),
    monomio(M,X).
polinomio(P-M,X) :- !,
    polinomio(P,X),
    monomio(M,X).
% un solo monomio es polinomio
polinomio(M,X) :- monomio(M,X).

% omision de coeficiente y exponente
% lo ponemos en primer lugar porque se representa con variable prolog
% (unificaria con cualquier otro caso)
monomio(Y,X) :-
    var(Y),
    !,
    Y==X.       % en la indeterminada X
% monomio estandard, explicito en coeficiente y exponente
monomio(K*Y^N,X) :-
    !,
    Y==X,       % en la indeterminada X
    number(K),  % coeficiente real
    integer(N), % exponente entero >=0
    N>=0.
% omision del coeficiente (se interpreta que es 1)
monomio(Y^N,X) :-
    !,
    Y==X,       % en la indeterminada X
    integer(N), % exponente entero >=0
    N>=0.
% omision del exponente (monomio de grado 1).
monomio(K*Y,X) :-
    !,
    Y==X,       % en la indeterminada X
    number(K).  % coeficiente real
% omision de variable X y exponente (termino independiente)
monomio(K,_X) :-
    number(K).
```

6. Considera el siguente programa Prolog:

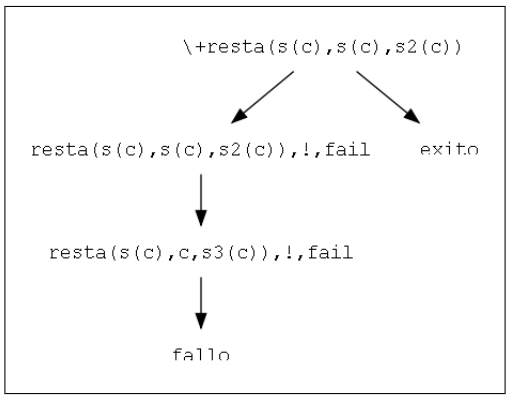
```
resta(X,c,X).
resta(X,s(Y),Z) :- resta(X,Y,s(Z)).
```

(a) Describe el árbol de búsqueda de cada uno de los siguientes objetivos a partir del programa anterior:

- `\+ resta(s(c),Y,s(s(c))).`



- `\+ resta(s(c),s(c),s(s(c))).`



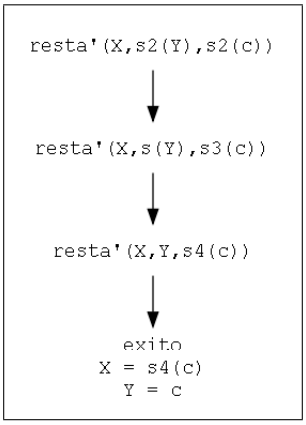
(b) Considera ahora la siguiente modificación del programa anterior:

```
resta'(X,c,X) :- !.
resta'(X,s(Y),Z) :- resta'(X,Y,s(Z)).
```

Determina razonadamente cuáles son las respuestas de los siguientes objetivos Prolog:

- `resta(X,s(s(Y)),s(s(c)))`.
  `X=s`^{4}`(c),Y=c; X=s`^{5}`(c),Y=s(c); X=s`^{6}`(c),Y=s`^{2}`(c);...`

- `resta'(X,s(s(Y)),s(s(c)))`.



- `\+ resta'(s(s(s(c))),Y,s(Z))`.
  `false`.

7. Define un predicado Prolog para conseguir la lista de todas las sublistas de una lista dada.

```
sublista(Xs,Ys) :- prefijo(Xs,Ys).
sublista(Xs,[_|Ys]) :- sublista(Xs,Ys).

prefijo([],_).
prefijo([X|Xs],[X|Ys]) :- prefijo(Xs,Ys).

sublistas(Xs,Xss) :- setof(S,sublista(S,Xs),Xss).
```