CONCATENAR

```
concat([],L,L).

concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).
```

INVERTIR

```
invertir([ ], [ ]).

invertir([Car|Cdr], Invertir) :- invertir(Cdr, Invertir_Cdr),

concatenar(Invertir_Cdr,[Car],Invertir).
```

IF THEN ELSE

```
if_then_else(P,Q,R) :- P, !, Q.

if_then_else(P,Q,R) :- R.
```

SUMA

```
suma(0,X,X).
Suma(s(X),Y,s(Z)):- suma(X,Y,Z).
```

PRODUCTO

```
Producto(0,X,0).
Producto(s(0),X,X).
Producto(s(X),Y,Res):- producto(X,Y,Z),suma(Z,Y,Res).
```

UNION

```
union(Conjunto,[],Conjunto).

union(Conjunto,[Car|Cdr],Res) :- miembro(Car,Res),

                                 union(Conjunto,Cdr,Res),!.

union(Conjunto,[Car|Cdr],[Car|Res]) :-union(Conjunto,Cdr,Res).
```

INVERTIR_DL

```
invertir(Lista,Invertir) :- invertir_dl(Lista,Invertir\[]).

invertir_dl([],Invertir\Invertir).
invertir_dl([Car|Cdr],Invertir\Cola) :- invertir_dl(Cdr,Invertir\[Car|Cola]).
```

DUPLICAR

```
Duplicar([],[]).

Duplicar([Car|Cdr],[Car,Car|Res]):- duplicar(Cdr,Res).
```

APLANAR

```
aplanar([],[]).

aplanar(Atomo,[Atomo]) :- atomic(Atomo), Atomo \== [].

aplanar([Car|Cdr], Resultado) :- aplanar(Car,Car_aplanado),

                                 aplanar(Cdr,Cdr_aplanado),

concatenar(Car_aplanado,Cdr_aplanado,Resultado).
```

APLANAR_DL

```
aplanar(Lista,Resultado) :- aplanar_dl(Lista,Resultado\[]).

aplanar_dl([],Lista\Lista).

aplanar_dl(Atomo,[Atomo|Cdr]\Cdr) :- atomic(Atomo), Atomo \== [].

aplanar_dl([Car|Cdr],Cabeza\Dif) :- aplanar_dl(Car,Cabeza\Dif_car),

                                    aplanar_dl(Cdr,Dif_car\Dif).
```

GET_ASOC

```
get_asoc(Llave,[Llave|Valor],Valor).
get_asoc(Llave,[Car|_],Res):-get_asoc(Llave,Car,Res),!.
get_asoc(Llave,[_|Cdr],Res):-get_asoc(Llave,Cdr,Res).
```

ESCALAR

```
escalar([],_,0).
escalar([Car1|Cdr1],[Car2|Cdr2],Resultado):-
escalar(Cdr1,Cdr2,Aux),
                                    Resultado is Aux+Car1*Car2.
```

ELIMINAR

```
eliminar([],_,[]).
eliminar([_|Y],1,Y):- !.
eliminar([Car|Cdr],X,[Car|R]) :- Y is X-1, eliminar(Cdr,Y,R).
```

AÑADIR

```
anadir(X,[],[X]).
anadir(X,Lista,Lista):-miembro(X,Lista),!.
anadir(X,Lista, Res):-insertar(X,Lista,Res),!.
```

MIEMBRO

```
miembro(Car,[Car|_]):- !.
miembro(Car,[_|Cdr]):- miembro(Car,Cdr).
```

INSERTAR

```
insertar(X,[],[X]).
insertar(X,[Car|Cdr],[X,Car|Cdr]):- X < Car.
insertar(X,[Car|Cdr],[X|Cdr]) :- X=Car.
insertar(X,[Car|Cdr],[Car|CdrX]):- X>Car,insertar(X,Cdr,CdrX).
```

INSERTAR CEROS

```
insert_ceros([],[]).
insert_ceros([Car|Cdr],[Car,0|R]) :- insert_ceros(Cdr,R).
```

BORRAR UNICOS

```
Borrar_unicos([],[]).
Borrar_unicos([Car|Cdr],Res):- miembro(Car,Cdr),!,borrar_unicos(Cdr,Res).
Borrar_unicos([Car|Cdr],Res):- borrar_unicos(Cdr,Res).
```

COMPLEMENTARIO COMP([3,1],[1,3,2,6],X) X=[2,6]

```
comp(_,[],[]).
comp(conj,[Car|Cdr],Res):- miembro(Car,Conj),!,comp(conj,Cdr,Res).
comp(conj,[Car|Cdr],Res):- comp(conj,Cdr,Res).
```

INTERSECCIÓN

```
inter([],C,[]).
Inter([Car|Cdr],C,[Car|Res]):- miembro(Car,C),!,inter(Cdr,C,Res).
inter([Car|Cdr],C,Res):- inter(Cdr,C,Res).
```

INCLUSION

```
inclusion([],Conjunto).
inclusion([Car|Cdr],Conjunto) :- miembro(Car,Conjunto),
inclusion(Cdr,Conjunto).
```

PREORDEN

```
:-op(600,xfy,[\]).

preorden(X,R) :-preorden_dl(X,R\[]).

preorden_dl([],X\X).
preorden_dl([Valor | [HijoIzq,HijoDer] ], [Valor|Izq]\Der ) :-
preorden_dl(HijoIzq,Izq\Z), preorden_dl(HijoDer,Z\Der).
```

ENORDEN

```
:-op(600,xfy,[\]).

enorden(X,R) :-enorden_dl(X,R\[]).
enorden_dl([],X\X).
enorden_dl([Car,Izq,Der],X\Z) :- enorden_dl(Izq,X\[Car|Y]),
                                 enorden_dl(Der,Y\Z).
```

QUICKSORT

```
quicksort([],[]).
quicksort([Car|Cdr], Res) :- partir(Car,Cdr,Izq,Der),
                             quicksort (Izq, OrdenI),
                             quicksort(Der, OrdenDer),
                             concat(OrdenI,[Car|OrdenDer],Res).

partir(_,[],[],[]).
partir(Pivote,[Car|Cdr],[Car|Izq],Der) :- Car =< Pivote,partir(Pivote,Cdr,Izq,Der).
partir(Pivote,[Car|Cdr],Izq,[Car|Der]) :- Car  >  Pivote,partir(Pivote,Cdr,Izq,Der).

concat([],L,L).

concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).
```

QUICKSORT_DL

```
:- op(600,xfy,[\]).

partir(_,[],[],[]).
partir(Pivote,[Car|Cdr],[Car|Izq],Der) :- Car =< Pivote,partir(Pivote,Cdr,Izq,Der).
partir(Pivote,[Car|Cdr],Izq,[Car|Der]) :- Car  >  Pivote,partir(Pivote,Cdr,Izq,Der).

quicksort(Lista,Orden) :- quicksort_dl(Lista,Orden\[]).
quicksort_dl([],X\X).
quicksort_dl([Car|Cdr], OrdenI\X) :- partir(Car,Cdr,Izq,Der),
                                     quicksort_dl(Izq, OrdenI\[Car|OrdenD]),
                                     quicksort_dl(Der,OrdenD\X).
```

SUBARBOL(SUBARBOL,ARBOL_BINARIO) ARBOL(IZQ,DER)

```
Subárbol(arbol(L,R), arbol(L,R)).
Subárbol(S,arbol(S,_)) :- !.
Subárbol(S,arbol(_,S)) :- !.
Subárbol(S,árbol(L,R)):-arbol(S,L), arbol(S,R)
```

CARTESIANO

```
Cart([],Conj,[]).
Cart([Car|Cdr],Conj,Res):- línea(Car,Conj,Linea),
                           Cartesiano(Cdr,Conj,Resto)
                           Concat(Linea,Resto,Res).
Línea(Elem,[],[]).
Línea(Elem,[Car|Cdr],[Elem,Car|Res]):- línea(elem,Cdr,Res).

concat([],L,L).

concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).
```

HANOI

```
:- op(600,yfx,a).

concat([],L,L) :- !.
concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).

hanoi(1,A,B,_,[A a B]).
hanoi(N,A,B,C,Movs) :- N > 1, N1 is N - 1,
                                 hanoi(N1,A,C,B,Movs_1),
                                 hanoi(N1,C,B,A,Movs_2),

                                 concat(Movs_1,[A a B|Movs_2],Movs).
```

HANOI ASSERTA

```
:- op(600,yfx,a).

concat([],L,L) :- !.
concat([Car|Cdr],L,[Car|R]) :- concat(Cdr,L,R).

hanoi(1,A,B,_,[A a B]).
hanoi(N,A,B,C,Movs):- N>1, N1 is N-1,
                                 hanoi(N1,A,C,B,Movs_1),
                                 asserta((hanoi(N1,A,C,B,Movs_1):-!)),
                                 hanoi(N1,C,B,A,Movs_2),
                                 retract((hanoi(N1,A,C,B,Movs_1):-!)),

                                 concat(Movs_1,[A a B|Movs_2],Movs).
```

VENTAJAS DESVENTAJAS DE LA RESOLUCION SLD RELACION CONSTR TRANSVER DEL ARBOL DE RESOLCION

Nos ahorramos la recursividad de izquierdas, es decir, no caeremos en un bucle infinito.

Fail rompe la lectura transversal, backtraking o fails darían problema con ese tipo de recorrido.