PLP - Práctica 7: Programación Lógica

Gianfranco Zamboni

20 de marzo de 2018

El motor de búsqueda de prolog

7.1. Ejercicio 1

Base de conocimiento:

```
padre(juan, carlos).   %(1)
padre(juan, luis).     %(2)
padre(carlos, daniel). %(3)
padre(carlos, diego).  %(4)
padre(luis, pablo).    %(5)
padre(luis, manuel).   %(6)
padre(luis, ramiro).   %(7)
abuelo(X,Y) :-         %(8)
    padre(X,Z),
    padre(Z,Y).
```

I. La consulta `abuelo(X, manuel)`, devuele `X = juan`.

II.

```
hijo(X, Y) :-       %(9)
    padre(Y, X).

hermano(X, Y) :-    %(10)
    padre(Z, X),
    padre(Z, Y),
    X \= Y.
```

```
descendiente(X, Y) :-    %(11)
    hijo(X,Y).
descendiente(X,Y) :-     %(12)
    hijo(X, Z),
    descendiente(Z, Y).
```

IV. `abuelo(juan,X)`

V. `hermano(pablo,X)`

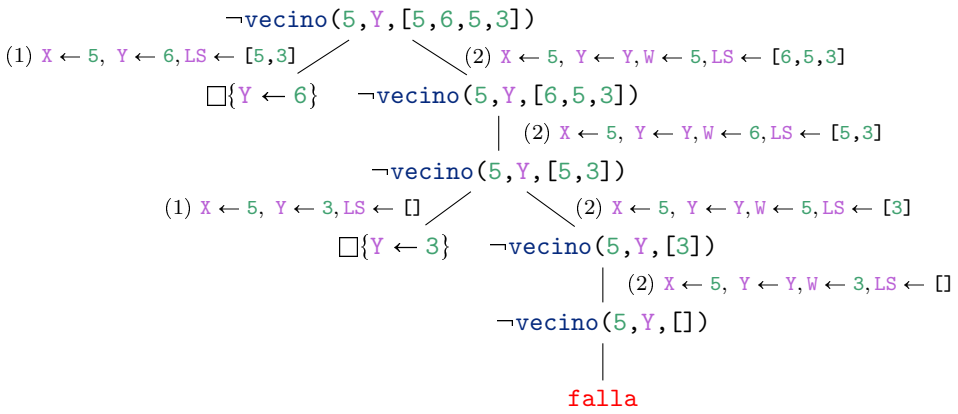
VII. `ancestro(juan,X)` devuelve `X = juan` por la primer regla y luego, si se pide otro resultado, se cuelga porque `Z` no está bien instanciado.

VIII. Con dar vuelta las condiciones de la segunda regla se arregla este problema

```
ancestro(X, X). \\
ancestro(X, Y) :- padre(X, Z), ancestro(Z, Y).
```

7.2. Ejercicio 2

I.



II. En este caso, si invertimos las reglas, obtendremos exactamente los mismos resultado y el árbol generado por prolog, es el espejo del inciso anterior.

7.3. Ejercicio 3

Bases de conocimiento

```
natural(0).
natural(suc(X)) :- natural(X).

menorOIgual(X, suc(Y)) :- menorOIgual(X, Y).
menorOIgual(X,X) :- natural(X).
```

I. Prolog unifica `menorOIgual(0,X)` con la regla `menorOIgual(X, suc(Y))` remplazando `X` (de la regla) por `0` e instanciando el `X` de la consulta con `suc(Y)`, obteniendo la resolvente `menorOIgual(0, Y)` que se resuelve de la misma manera. Entonces, entra en un ciclo infinito en el que siempre unifica con la primer regla.

II. Esto se debe a que Prolog no tiene suficiente información sobre `X` como para descartar la primer regla (`X` no está correctamente instanciada) por lo que siempre es posible hacerla coincidir con cualquier regla.

III. Otra vez, con cambiar el orden de evaluación de las reglas, el problema se arregla:

```
menorOIgual(X,X) :- natural(X).
menorOIgual(X, suc(Y)) :- menorOIgual(X, Y).
```

Estructuras, instanciación y reversibilidad

7.4. Ejercicio 4

```
%concatenar(?Lista1, ?Lista2, ?Lista3)
concatenar([], Lista2, Lista2).
concatenar([X | T1], Lista2, [X| T3]) :-
    concatenar(T1, Lista2, T3).
```

7.5. Ejercicio 5

I.

```
%last(?L, ?U)
last([ X ], X ).
last([ _ | T ], Y) :- last(T,Y).
```

II.

```
%tienenLaMismaLongitud(+L, +L1)
tienenLaMismaLongitud(L1, L2) :-
    length(L1, N ),
    length(L2, N ).

%reverse(+L, -L1)
reverse([],[]).
reverse( [ X | T ], R) :-
    tienenLaMismaLongitud([X | T], R),
    reverse(T, RT ),
    append(RT, [ X ], R).
```

III.

```
%maxlista(+L, -M).
maxlista([E],E).
maxlista([E|T],E):-
    maxlista(T,M),E>=M.
maxlista([E|T],M):-
    maxlista(T,M),E=<M.

%minlista(+L, -M).
minlista([E],E).
minlista([E|T],M):-
    minlista(T,M),E>=M.
minlista([E|T],E):-
    minlista(T,M),E=<M.
```

7.6. Ejercicio 6

```
aplanar([],[]).
aplanar([ [] | T ], Res) :-
    aplanar(T, Res).
aplanar([ [X | T1 ] | T ], Res) :-
    aplanar([ X | T1 ], Y),
    aplanar(T, RecT),
    append(Y, RecT, Res).
aplanar([ X | T ], [X | Res]) :-
    not(is_list(X)),
    aplanar(T, Res).
```

IV.

```
%prefijo(?P, +L).
prefijo([], _).
prefijo([E|T2], [E|T]):-
    prefijo(T2,T).
```

V.

```
%sufijo(?S, +L).
sufijo(S, L):-
    prefijo(P,L),
    append(P, S, L).
```

VI.

```
%sublista(?S, +L).
sublista([], _).
sublista(XS, L) :-
    prefijo(P, L),
    sufijo(S, L),
    append(P, XS, P1),
    append(P1, S, L),
    length(XS, N),
    N >= 1.
```

VII.

```
%pertenece(?X, +L).
pertenece(X, [X|_]).
pertenece(X, [Y|T]):-
    Y \= X,
    pertenece(X,T).
```

7.7. Ejercicio 7

I.

```
%palindromo(+L, -L1)
palindromo(L, L1) :-
    reverse(L, A),
    append(L,A,L1).
```

II.

```
%doble(+L, -L1)
doble([], []).
doble([ X | T], [X, X | Rec ]) :-
    doble(T, Rec).
```

III.

```
%iesimo(?I, +L, -X)
iesimoAux(1, [X | _], X).
iesimoAux(I, [_ | T], Y) :-
    I1 is I - 1,
    iesimo(I1, T, Y).

iesimo(I, L, X) :-
    length(L, N),
    between(1,N, I),
    iesimoAux(I, L, X).
```

7.8. Ejercicio 8

I. Se debe instanciar `X` en un valor especifico. Entonces devuelve todos los números a partir de ese valor. Si se instancia `Y` en un valor menor que `X`, entonces se cuelga porque siempre se puede unificar con la segunda regla. Si se instancia `Y` en un valor mayor, entonces devolverá como primer resultado el valor de `Y` y luego se colgará por la misma razón que antes.

Y si no se instancia `X` entonces tira error porque no tiene suficiente información sobre la variable como para realizar unificación.

II.

```
%desde2(+X,?Y)
desde2(X, Y) :-
    nonvar(Y),
    Y >= X.
desde2(X,Y) :-
    var(Y),
    desde(X,Y).
```

7.9. Ejercicio 9

I.

```
%interseccionAux(+L1, +L2, +L3, -L4)
interseccionAux([], _, _, []).
interseccionAux([ X | T ], L2, Usados, Resultado) :-
    not(member(X, L2)),
    interseccionAux(T, L2, Usados, Resultado).
interseccionAux([ X | T ], L2, Usados, [ X | L4 ]) :-
    member(X, L2),
    not(member(X, Usados)),
    interseccionAux(T, L2, [ X | Usados], L4).
interseccionAux([ X | T ], L2, Usados, L4 ) :-
    member(X, Usados),
    interseccionAux(T, L2, Usados, L4).

%intersección(+L1, +L2, -L3)
interseccion(L1, L2, L3) :-
    interseccionAux(L1, L2, [], L3).
```

II.

```
sufijoDeLongitud(L, N, S) :-
    sufijo(S, L),
    length(S, N).

prefijoDeLongitud(L, N, S) :-
    prefijo(S, L),
    length(S, N).

%split(+N,+L, -L1, -L2) -- N y L deben estar definidos
split(N, L, L1, L2) :-
    length(L, M),
    N1 is M - N,
    prefijoDeLongitud(L, N, L1),
    sufijoDeLongitud(L, N1, L2).
```

III.

```
%borrar(+ListaOriginal, +X, -ListaSinXs)
borrar([], _, []).
borrar([X | T], X, ListaSinXs) :-
    borrar(T, X, ListaSinXs).
borrar([ Y | T], X, [ Y | Rec ]) :-
    X \= Y,
    borrar(T, X, Rec).
```

IV.

```
%sacarDuplicados(+L1, -L2)
sacarDuplicados([], []).
sacarDuplicados([ X | T], [ X | Rec ]) :-
    borrar(T, X, T1),
    sacarDuplicados(T1, Rec).
```

V.

```
%concatenarTodas( +LL, -L)
concatenarTodas([], []).
concatenarTodas([ X | T], Res) :-
    concatenarTodas(T, Rec),
    concatenar(X, Rec, Res).

%todosSusMiembrosSonSublitas(+LListas, +L)
todosSusMiembrosSonSublitas([], _).
todosSusMiembrosSonSublitas([ X | XS], L) :-
    sublista(X, L),
    todosSusMiembrosSonSublitas(XS, L).
%reparto(+L, +N, -LListas)
reparto(L, N, LListas) :-
    length(LListas, N),
    todosSusMiembrosSonSublitas(LListas, L),
    concatenarTodas(LListas, L).
```

VI.

```
%repartoSinVacias(+L, -LListas)
repartoSinVacias(L, LListas) :-
    length(L, N),
    between(1, N, X),
    reparto(L, X, LListas),
    not(member([], LListas)).
```

7.10. Ejercicio 10

```
%intercalar(?L1, ?L2, ?L3) -- Funciona para todas las combinaciones posibles.
intercalar([], [], []).
intercalar([], L, L) :-
    length(L, N),
    N >= 1.
intercalar(L, [], L) :-
    length(L, N),
    N >= 1.
intercalar([ X | T1], [ Y | T2 ], [ X, Y | T3 ] ) :-
    intercalar(T1, T2, T3).
```

7.11. Ejercicio 11

```
%arbolEjemplo
arbolEjemplo(bin(bin(nil,1,nil),2,bin(bin(nil,10,nil),20,bin(nil,30,nil)))).

%vacio(?A)
vacio(nil).

%raiz(?A, ?R)
raiz(bin(_,X,_), X).

%altura(+A, -H)
altura(nil, 0).
altura(bin(I, _, D), H) :-
    altura(I, HI),
    altura(D, HD),
    H is max(HI, HD) + 1.

%cantidadDeNodos(+A, -N)
cantidadDeNodos(nil, 0).
cantidadDeNodos(bin(I, _, D), N) :-
    cantidadDeNodos(I, NI),
    cantidadDeNodos(D, ND),
    N is NI + ND + 1.
```

7.12. Ejercicio 12

```
%inorder(+AB, -Lista)
inorder(nil, []).
inorder(bin(I, X, D), Inorder) :-
    inorder(I, LI),
    inorder(D, LD),
    append(LI, [X | LD], Inorder).

%arbolConInorder(-L, +AB)
arbolConInorder([], nil).
arbolConInorder(XS, bin(AI, X, AD)) :-
    reparto(XS, 2, [LI, [X | LD]]),
    arbolConInorder(LI, AI),
    arbolConInorder(LD, AD).

%abb(+T)
aBB(nil).
aBB(bin(I, _, D)) :-
    aBB(I),
    aBB(D).

%aBBInsertar(+X, +T1, -T2)
aBBInsertar(X, nil, bin(nil, X, nil)).
aBBInsertar(X, bin(AI, Y, AD), bin(AIM, Y, AD)) :-
    X =< Y,
    aBBInsertar(X, AI, AIM).
aBBInsertar(X, bin(AI, Y, AD), bin(AI, Y, ADM)) :-
    X >= Y,
    aBBInsertar(X, AD, ADM).
```

Generate and test

7.13. Ejercicio 13

```
%coprimos(-X,-Y)
coprimos(X,Y) :-
    desde(2, X),
    between(2, X, Y),
    1 is gcd(X,Y).
```

7.14. Ejercicio 14

```
listasQueSuman([],0,0).
listasQueSuman([X|XS],S,N):-
    between(0,S,X),
    N2 is N-1,
    length(XS,N2),
    S2 is S-X,
    listasQueSuman(XS,S2,N2).

cuadradoSemiLatinoAux(_,0,[],_).
cuadradoSemiLatinoAux(M,N,[X|XS],S):-
    N2 is N-1,
    listasQueSuman(X,S,M),
    cuadradoSemiLatinoAux(M,N2,XS,S).

cuadradoSemiLatino(N,XS):-
    length(XS,N),
    desde(0,S),
    cuadradoSemiLatinoAux(N,N,XS,S).
```

7.15. Ejercicio 15

(Solución dada en clases)

```
%ladoValido(+A, +B, +C)
ladoValido(A,B,C) :-
    S is B+C,
    A < S,
    D is abs(B-C),
    A > D.

%esTriangulo(+T)
esTriangulo(tri(A,B,C)) :-
    ladoValido(A,B,C),
    ladoValido(B,C,A),
    ladoValido(C,A,B).

%perim  etro(?T,?P)
perimetro(tri(A,B,C), P) :-
    desde2(3,P),
    M is P-2,
    between(1,M,A),
    N is P - A -1,
    between(1, N, B),
    C is P - A - B,
    esTriangulo(tri(A,B,C)).
```

Negación por falla

7.16. Ejercicio 16

```
%diferenciaSimétrica(Lista1, +Lista2, -Lista3)
diferenciaSimetrica([], L2, L2).
diferenciaSimetrica([ X | L1], L2, [ X | XS]) :-
    not(member(X, L2)),
    diferenciaSimetrica(L1, L2, XS).
diferenciaSimetrica([ X | L1], L2, XS) :-
    member(X, L2),
    borrar(L2, X, L2SinX),
    diferenciaSimetrica(L1, L2SinX, XS).
```

7.17. Ejercicio 17

I. La consulta busca todos los valores para los que valga $P(Y) \wedge \neg Q(Y)$.

II. Si se invierten el orden de los literales y queda `not(q(Y)),p(Y)`, entonces prolog se cuelga porque no puede instanciar `Y` correctamente.

7.18. Ejercicio 18

```
%sumList
sumList([], 0).
sumList([X | XS], N) :-
    sumList(XS, N1),
    N is N1 + X.

%diff
diff(L1, L2, DL) :-
    sumList(L1, N1),
    sumList(L2, N2),
    DL is abs(N1-N2).

%corteMásParejo(+L,-L1,-L2)
corteMasParejo(L, L1, L2) :-
    append(L1, L2, L),
    diff(L1,L2,DL),
    not((append(M1,M2,L),
        diff(M1,M2,DM),
        DM < DL
        )).
```