

Programación recursiva en Prolog

Curso 2002/03, Fecha:14/12/2002

Enunciado 1 (naturales) *El siguiente predicado define la suma de números naturales representados mediante la constante 0 y la función siguiente s*

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

Una posible pregunta para calcular el valor de la suma de 3 y 2 sería:
 $suma(s(s(0)), s(s(s(0))), V)$

- *Calcular el valor V tal que el resultado de sumar 2 y V sea 5*
- *Calcular los valores V y W cuya suma sea 5*
- *Definir el predicado producto*
- *Definir el predicado potencia*
- *(Opcional). Definir otros predicados aritméticos (por ejemplo, máximo común divisor, menor, etc.)*

Enunciado 2 (pertenece) *El siguiente predicado chequea si un elemento pertenece a una lista*

$pertenece(X, [_ | _]).$
 $pertenece(X, [_ | Ys]) :- pertenece(X, Ys).$

- *Preguntar si el elemento 2 pertenece a la lista [1,2,3,4].*
- *Preguntar si existe un elemento X que pertenezca a la lista [1,2,3,4]*
- *Preguntar si existe una lista L que incluya al elemento 2*

Enunciado 3 (concatena) *Definir un predicado que permita concatenar 2 listas*

- *Utilizando dicho predicado, calcular parejas de listas que al concatenarse resulten en [1,2,3,4]*

- Definir el predicado *prefijo* que se cumple si una lista es un prefijo de otra
- Definir el predicado *sufijo* que se cumple si una lista es un sufijo de otra
- Definir el predicado *sublista* que se cumple si una lista está incluida en otra

Enunciado 4 (inserta) Definir un predicado *inserta* (X, Xs, Ys) que se cumple si la lista Ys es el resultado de insertar el elemento X en la lista Xs

?- *inserta* (1, [2, 3], V).

V = [1, 2, 3] ;
 V = [2, 1, 3] ;
 V = [2, 3, 1] ;
 No

Enunciado 5 (permutaciones) Definir un predicado *permutaciones* (Xs, Ys) que se cumple si la lista Ys es una permutación de la lista Xs

?- *permutaciones* ([1, 2, 3], V).

V = [1, 2, 3] ;
 V = [2, 1, 3] ;
 V = [2, 3, 1] ;
 V = [1, 3, 2] ;
 V = [3, 1, 2] ;
 V = [3, 2, 1] ;
 No

Enunciado 6 (ordenada) Definir un predicado *ordenada* (Xs) que se cumple si la lista Xs contiene todos los elementos ordenados de menor a mayor.

?- *ordenada* ([3, 1, 2]).

No

?- *ordenada* ([1, 3, 5]).

Yes

Enunciado 7 (ordenaSalvaje) Definir un predicado *ordena* (Xs, Ys) que se cumple si la lista Ys contiene los elementos ordenados de Xs .

?- *ordena* ([3, 1, 2], V).

V = [1, 2, 3]

Aunque existen diversos algoritmos de ordenación, en este ejercicio puede utilizarse uno de los algoritmos menos eficientes que consiste simplemente en generar permutaciones de la lista a ordenar y chequear que están ordenadas.

Enunciado 8 (nReinas) El predicado *segura* se cumple cuando las posiciones de n reinas en un tablero de ajedrez no se amenazan entre sí. En <http://www.di.uniovi.es/labra/PLF/prac/worlds/nReinas2.url> puede observarse una representación en realidad virtual de una de las posibles soluciones.

La representación del tablero se realiza mediante una lista de las alturas de cada reina. Una posible solución sería: $[4,2,7,3,6,8,5,1]$ que indica que las reinas aparecen en las coordenadas $(1,4), (2,2), (3,7), (4,3), \dots$

```
segura ([]).
segura ([R|Rs]):- segura (Rs), noAtaque (R, Rs, 1).
```

```
noAtaque (_, [], -).
noAtaque (Y, [Y1|Ys], D):-
    Y1 - Y =\= D,
    Y - Y1 =\= D,
    D1 is D + 1,
    noAtaque (Y, Ys, D1).
```

Construir un predicado *Prolog* que resuelva el problema generando todas las posibles soluciones al problema

Enunciado 9 (verNReinas(Opcional)) Almacenar las soluciones al problema de las n Reinas en un fichero VRML

Para facilitar este problema se proporciona a continuación un predicado que genera un tablero de ajedrez en realidad virtual y coloca una pieza en una posición determinada

```
main:- open ('f.wrl', write, S),
        cabecera (S),
        viewPoint (S),
        tablero (S),
        ponPieza (S, 3, 4),
        close (S).

ponPieza (S, X, Z):-
    Xr is 10 * X,
    Zr is 10 * Z,
    putCylinder (S, Xr, 3, Zr, rgb (0, 0, 1), 3, 6).

tablero (S):- filas (S, 1, 8).
```

```

filas (S,M,M):-cols (S,1,8,M).
filas (S,M,N):-M < N,
                cols (S,1,8,M),
                M1 is M + 1,
                filas (S,M1,N).

cols (S,M,M,Z):-cell (S,M,Z).
cols (S,X,M,Z):-X < M,
                cell (S,X,Z),
                X1 is X + 1,
                cols (S,X1,M,Z).

cell (S,X,Z):-
  Xr is 10 * X, Zr is 10 * Z,
  getColor (C,X,Z),
  putBox (S,Xr,0,Zr,C,10,1,10).

getColor (C,X,Y):-
  0 is (X+Y) mod 2
  → C = rgb (1,0,0)
  ; C = rgb (0,1,0).

cabecera (S):-
  write (S,'#VRML V2.0 utf8\n\n').

viewPoint (S):-
  write (S,'Viewpoint { description "Punto" \n'},
  write (S,' orientation -0.6 -0.6 -0.6 0.6\n'),
  write (S,' position 10 40 160 } \n').

putBox (S,X,Y,Z,C,SX,SY,SZ):-
  translate (S,X,Y,Z),
  color (S,C),
  box (S,SX,SY,SZ),
  endColor (S),
  endTranslate (S).

putCylinder (S,X,Y,Z,C,R,H):-
  translate (S,X,Y,Z),
  color (S,C),
  cylinder (S,R,H),
  endColor (S),
  endTranslate (S).

translate (S,X,Y,Z):-

```

```

    write(S, ' Transform { translation }',
    write3(S,X,Y,Z), nl(S),
    write(S, ' children [']).

endTranslate(S):-
    write(S, ' ] }\n').

color(S, rgb(R,G,B)):-
    write(S, ' Shape { appearance Appearance }',
    write(S, ' { material Material { diffuseColor }'),
    write3(S,R,G,B),
    write(S, ' } }').

endColor(S):-
    write(S, ' } ').

box(S,X,Y,Z):-
    write(S, ' geometry Box { size }',
    write3(S,X,Y,Z),
    write(S, ' } ').

cylinder(S,R,H):-
    write(S, ' geometry Cylinder { radius }',
    write(S,R),
    write(S, ' height '),
    write(S,H),
    write(S, ' } ').

write3(S,X,Y,Z):-
    format(S, '~2f ~2f ~2f ', [X,Y,Z]).

```