PROGRAMACIÓN DECLARATIVA: LÓGICA Y RESTRICCIONES
Grado en Ingeniería Informática / Grado en Matemáticas e Informática

9 de Junio de 2014 **Convocatoria de Junio**

DURACIÓN: 120 minutos

Ejercicio 1. ***Listas***

Definir el predicado **lista_a_conjunto/2** (lista_a_conjunto(+L,-C)) que se verifique si C es el conjunto correspondiente a la lista L (es decir, C contiene los mismos elementos que L en el mismo orden, pero si L tiene elementos repetidos sólo se incluye en C la última aparición de cada elemento). (*1.25 puntos*)

Ejemplo: ?- lista_a_conjunto([b,a,b,d], C).

C= [a,b,d]

Definir el predicado **lista_comprimida/2** (lista_comprimida(+L1,-L2)) que se verifique si L2 es la lista obtenida sustituyendo cada sucesión de un elemento de L1 por dicho elemento. (*1.25 puntos*)

Ejemplo: ?- lista_comprimida([a,b,b,a,a,a,c,c,b,b,b],L).

L = [a,b,a,c,b]

```
lista_a_conjunto([],[]).
lista_a_conjunto([X|Xs],S):-
        lista_a_conjunto(Xs,S0),
        incluye_o_no(S0,X,S).

incluye_o_no(S0,X,S):-
        member(X,S0), !,
        S = S0.
incluye_o_no(S,X,[X|S]).
```

```
lista_comprimida([],[]).
lista_comprimida([X],[X]).
lista_ comprimida([X,X|Xs],L):- ,
        lista_ comprimida([X|Xs],L).
lista_ comprimida([X1,X2|Xs],[X1|L]):-
        X1 \== X2,
        lista_ comprimida ([X2|Xs],L).
```

El predicado member/2 es el habitual: member(X,L) si X pertenece a la lista L.

Ejercicio 2. ***Polígonos***

Suponiendo que un polígono se representa por su nombre y las longitudes de sus lados.

Definir el predicado **es_equilátero /1** (es_equilátero(+P)) que se verifica si el polígono P es equilátero (es decir, que todos sus lados son iguales). (*1.25 puntos*)

Ejemplo: ?- es_equilatero(triangulo(4,4,4)).

yes

?- es_equilatero(cuadrilatero(3,4,5,3)).

no

Definir el predicado **seleccion_poligonos/2** (seleccion_poligonos(+Poligonos,-Seleccion)) que dada una lista de polígonos (Poligonos), lee el polígono favorito del usuario (Tipo) y se verifica si Seleccion es la lista de los polígonos favoritos (es decir, de tipo Tipo). (*1.25 puntos*)

Ejemplo: ?- seleccion_poligonos([triangulo(4,4,4), cuadrilatero(3,4,5,3), cuadrado(3,3,3,3), rombo(2,2,2,2), romboide(2,3,2,3), triangulo(3,4,3), triangulo(4,5,6), cuadrado(2,2,2,2), rectangulo(4,1,4,1)], Seleccion).

|: triangulo.

Seleccion = [triangulo(4,4,4), triangulo(3,4,3), triangulo(4,5,6)]

```
es_equilatero(Poligono):-
        Poligono =.. [_,L|Lados],
        iguales(Lados,L).

iguales([],_).
iguales([X|Xs],X):-
        iguales(Xs,X).
```

```
seleccion_poligonos(Ps,L):-
        read(P),
        seleccion(Ps,P,L).

seleccion([],_,[]).
seleccion([X|Xs],P,L):-
        functor(X,F,_),
        selecciona(F,P,X,L,L1),
        seleccion_poligonos(Xs,P,L1).

selecciona(F,F,X,L,L1):- !,
        L = [X|L1].
selecciona(_,_,_,L,L).
```

Ejercicio 3. Programar el predicado **fib/2** (fib(+N,-F)) que calcula el N-ésimo número F en la sucesión de Fibonacci, de forma que el código sólo tenga una llamada recursiva.

Se admite que son ciertos fib(0,0) y fib(1,1). (*2,5 puntos*)

```
fib(0,0).
fib(1,1).
fib(N,F):- N>1,
        fib(1,N,0,1,F).

fib(N,N,_,F1,F):- !,
        F = F1.
fib(N,M,F1,F2,F):- N<M,
        N1 is N+1,
        F3 is F1+F2,
        fib(N1,M,F2,F3,F).
```

Ejercicio 4. Supongamos que queremos encontrar la solución del siguiente criptograma:

```
  ABC
+  CB
---------
  CCA
```

El siguiente predicado **suma/3**, en CLP(R), no da ninguna solución (a pesar de que la solución existe), debido al orden de los objetivos en el cuerpo. Explicar como se deberían reordenar los objetivos del cuerpo de suma/3 para que el predicado si de una solución, y razonar el porque de esta diferencia de comportamiento. (*2,5 puntos*)

```
suma(A,B,C) :-
    natural(A), natural(B), natural(C),
    A .>. 0, A .<=. 9, B .>=. 0, B .<=. 9, C .>. 0, C .<=. 9,
    100*A+10*B+C+10*C+B .=. 100*C+10*C+A.


natural(0).
natural(N) :- N .>. 0, natural(N-1).
```

El orden que sí da la solución es:

```
suma(A,B,C) :-
    A .>. 0, A .<=. 9, B .>=. 0, B .<=. 9, C .>. 0, C .<=. 9,
    100*A+10*B+C+10*C+B .=. 100*C+10*C+A,
    natural(A), natural(B), natural(C).
```

Las llamadas al predicado natural/1 actúan de generador de posibles soluciones (números naturales). Al invertir el orden pasamos de un algoritmo generar-y-comprobar a otro restringir-y-generar. Al generar antes de comprobar, se produce primero A=B=C=0 e inmediatamente A.>.0 falla; esto provoca vuelta atrás en la llamada natural(C), que devuelve infinitas soluciones pero siempre A.>.0 falla (ya que A no cambia), así que la ejecución se embucla. Al generar después de las restricciones, se generan solo valores de las variables que satisfacen las restricciones inmediatas (las inecuaciones) y se produce vuelta atrás sobre la generación de números hasta que los valores satisfacen la ecuación (obteniéndose la solución).