UNIVERSIDAD POLITÉCNICA DE MADRID

Programación Declarativa: Prolog

Grado Matemáticas e informática

GARCÍA MARTÍN, IGNACIO **u12m071**
MOYA LÓPEZ, AITOR **v13m022**
RANILLA CORTINA, SANDRA **v13m046**
ROSADO GONZÁLEZ, RODRIGO **v13M018**

31/5/2016

Índice

Problema de Enumeración

- Código y Explicación

Para el desarrollo de este ejercicio de la práctica hemos diseñado el siguiente código de Prolog:

Tras varias lecturas del enunciado y una breve explicación en el aula del ejercicio hemos decidido construir el esqueleto del código con 12 predicados, de los cuales 11 son funciones auxiliares.

El primer predicado construido, nodos/2, tiene como primer argumento un grafo de la forma [A-B, B-C] y como segundo argumento, la lista de elementos, sin repeticiones setof/3, que son parte de la lista dada como primer argumento. Atom(X), nos permite que esos elementos tengan aridad 0 y no sean numéricos.

```
% " Metodo auxiliar nodos " %
nodos(X,Y):-setof(N,(subterm(N,X),atom(N)),Y).
```

Para seguir, hemos construido el predicado subterm/2, que comprueba que un término es sub-término de otro, es decir, que el primer argumento está contenido en el segundo. Para desarrollar el predicado hemos necesitado de un predicado auxiliar, subtermList/2.

```
% " Metodo auxiliar subterm " %
subterm(Term,Term).
subterm(Sub,Term)  :-compound(Term),
                     Term =.. [F|Args],
                     subtermList(Sub, Args).

subtermList(Sub,[Arg|Args]):-subterm(Sub,Arg).
subtermList(Sub,[Arg|Args]):-subtermList(Sub,Args).
```

Para continuar, se construye el predicado **listanumero/3**, cuyos primer y segundo argumento son dos números, tales que el primero ha de ser menor que el segundo y a partir de ahí en el tercer argumento se tiene la lista de elementos contenidos entre el primer y segundo argumento.

```
% " Metodo auxiliar listanumero " %
listanumero(X,X,[X]).
listanumero(X,Y,[X|L]):- X < Y,
                         X1 is X + 1,
                         listanumero(X1,Y,L).
```

Construimos también el predicado **noesmiembro/2** que será cierto si el primer argumento no es miembro de la lista que se tiene como segundo argumento.

```
% " Metodo auxiliar noesmiembro " %
noesmiembro(X,[X|_]):- !, false.
noesmiembro(X,[_|L]):- !,noesmiembro(X,L).
noesmiembro(_,[]).
```

Los predicados visto con anterioridad tienen un uso fuera de este ejercicio. Los que vamos a ver a continuación son predicados auxiliares pero diseñados para el predicado principal **enumerar/3**.

El primer predicado construido, **enumeraNodo/3**, que tiene como primer argumento una lista de elementos **enum(N, X)**, como segundo argumento una lista de los nodos del grafo y como último argumento una lista de números desde el 1 hasta la longitud de la lista del grafo + 1. En este predicado lo único que se realiza es una comprobación de que los números que contienen los nodos del grafo no están repetidos y que son un número del 1 al tamaño del grafo +1.

```
% " Metodo auxiliar enumeraArco " %
enumeraArco([],[],_LA).
enumeraArco([X|Arcos],[enum(N,X)|EnumArcos],LA):- member(N,LA),
                                                  enumeraArco(Arcos,EnumArcos,LA),
                                                  noesmiembro(enum(N,_L),EnumArcos).
```

Para seguir, hemos construido el predicado **enumeraArco/3**, que como el predicado anterior, tiene como argumentos la lista de arcos, es decir, el grafo, una lista de elementos **enum(N, X-Y)** y una lista de números desde el 1 hasta la longitud de la lista del grafo. Se comprueba que los números que contienen los nodos del grafo no estén repetidos y sean un número del 1 al tamaño del grafo.

```
% " Metodo auxiliar enumeraArco " %
enumeraArco([],[],_LA).
enumeraArco([X|Arcos],[enum(N,X)|EnumArcos],LA):- member(N,LA),
                                                   enumeraArco(Arcos,EnumArcos,LA),
                                                   noesmiembro(enum(N,_L),EnumArcos).
```

Por último, el último predicado auxiliar construido es **comprobarPareja/2**, que junto a **comprobarParejaAux/3**, comprueba que los números de las aristas, son el resultado de la resta absoluta de los números de los nodos, los cuales son sus extremos.

```
% " Metodo auxiliar comprobarPareja " %
comprobarPareja(_EnumNodos, []).
comprobarPareja(EnumNodos, [enum(N,A1-A2)|EnumArcos]):- member(enum(N1,A1),EnumNodos),
                                                        member(enum(N2,A2),EnumNodos),
                                                        comprobarParejaAux(N,N1,N2),
                                                        comprobarPareja(EnumNodos,EnumArcos).

comprobarParejaAux(X,Y,Z):- A is Y - Z,
                            A1 is abs(A),
                            X == A1.
```

Para finalizar, explicamos el diseño del predicado principal **enumerar/3**. Como se describe en el enunciado de la práctica, el primer argumento corresponde al grafo, el segundo argumento a la lista de nodos y el tercero a la lista de arcos. Lo que se pretende con este predicado es a partir de los predicados auxiliares, utilizados en orden correcto, es generar las posibles soluciones y hacer las comprobaciones necesarias según los criterios descritos en el enunciado:

```
% " Metodo Enumerar " %
enumerar(Arcos,EnumNodos,EnumArcos):- length(Arcos,Y),
                                      X is Y + 1,
                                      listanumero(1,X,LN),
                                      listanumero(1,Y,LA),
                                      nodos(Arcos,L),
                                      enumeraNodo(EnumNodos,L,LN),
                                      enumeraArco(Arcos,EnumArcos,LA),
                                      comprobarPareja(EnumNodos,EnumArcos).
```

- Consultas

Se proporcionan todos los argumentos

```
1 ?- enumerar([a-b,b-c],[enum(3,a), enum(1,b), enum(2,c)],[enum(2,a-b), enum(1,b-c)]).
true ;
false.
```

En esta consulta comprobamos que el predicado **enumerar/3** construido comprueba de manera correcta todas las especificaciones.

Se proporciona primer argumento y segundo argumento

```
1 ?- enumerar([a-b,b-c],[enum(3,a), enum(1,b), enum(2,c)],L).
L = [enum(2, a-b), enum(1, b-c)] ;
false.
```

En esta consulta vemos que el predicado devuelve de manera correcta, dado un grafo y la enumeración de los nodos, la lista de enumeración de arcos.

Se proporciona primer argumento y tercer argumento

```
1 ?- enumerar([a-b,b-c],L,[enum(2,a-b), enum(1,b-c)]).
L = [enum(1, a), enum(3, b), enum(2, c)] ;
L = [enum(3, a), enum(1, b), enum(2, c)] ;
false.
```

En este caso, dado el grafo y la lista de enumeración de arcos, el enunciado devuelve de forma correcta las posibles listas de enumeración de nodos.

Se proporcionan todos los argumentos. Casos incorrectos

```
1 ?- enumerar([a-b,b-c],[enum(4,a), enum(1,b), enum(1,c)],[enum(3,a-b), enum(1,b-c)]).
false.

1 ?- enumerar([a-b,b-c,c-d],[enum(3,a), enum(1,b), enum(2,c)],[enum(2,a-b), enum(1,b-c)]).
false.

1 ?- enumerar([a-b,b-c],[enum(3,a), enum(1,b), enum(2,c)],[enum(1,a-b), enum(2,b-c)]).
false.
```

- Comentarios

En el desarrollo de este ejercicio hemos encontrado diferentes dificultades, entre las que prima el entendimiento del predicado enumera/3.

En lo que se refiere al enunciado y desarrollo creemos haber entendido correctamente el funcionamiento del predicado enumera/3, visualizando la lista de arcos, como un grafo, y entendiendo de forma correcta como realizar la recursividad a la hora de recorrer la lista de aristas y de nodos.

Podemos concluir con que el mayor problema que hemos encontrado es la recursividad y el orden de las llamadas a predicados auxiliares, pues ha sido frecuente la caída en bucle infinito.

Accidente Marítimo

- Código y Explicación

Para el desarrollo de este ejercicio de la práctica hemos diseñado el siguiente código de Prolog:

Tras varias lecturas del enunciado y una breve explicación en el aula del ejercicio hemos decidido construir el esqueleto del código con 9 predicados, de los cuales 8 son funciones auxiliares.

Para seguir, hemos construido el predicado **subterm/2**, que comprueba que un término es sub-término de otro, es decir, que el primer argumento está contenido en el segundo. Para desarrollar el predicado hemos necesitado de un predicado auxiliar, **subtermList/2**.

```
% " Metodo auxiliar subterm " %
subterm(Term,Term).
subterm(Sub,Term)  :-compound(Term),
                     Term =.. [F|Args],
                     subtermList(Sub, Args).

subtermList(Sub,[Arg|Args]):-subterm(Sub,Arg).
subtermList(Sub,[Arg|Args]):-subtermList(Sub,Args).
```

Construimos también el predicado **noesmiembro/2** que será cierto si el primer argumento no es miembro de la lista que se tiene como segundo argumento.

```
% " Metodo auxiliar noesmiembro " %
noesmiembro(X,[X|_]):- !, false.
noesmiembro(X,[_|L]):- !,noesmiembro(X,L).
noesmiembro(_,[]).
```

A continuación mostramos el predicado, combinatoria/2, que tiene como primer argumento un número n y como segundo la suma recursiva, es decir, la suma de los n primeros números. Este predicado es utilizado en reparación/3, para calcular la longitud de la lista de parejas de buzos.

```
% " Metodo auxiliar combinatoria " %
combinatoria(0,0).
combinatoria(1,1).
combinatoria(X,Y):- X > 1,
                    X1 is X - 1,
                    combinatoria(X1,Y1),
                    Y is Y1 + X.
```

A continuación mostramos el predicado, distinto/2, que comprueba que el primer y segundo argumento son distintos

```
% " Metodo auxiliar distinto " %
distinto(X,Y):- X \= Y.
```

Para seguir mostramos el predicado tiempo/3, que tiene como primer y segundo argumento dos tiempos, y como tercer argumento el menor de los dos tiempos anteriores. Este predicado es utilizado en reparacionAux/4, para tomar el tiempo del buzo que menos puede estar sumergido.

```
% " Metodo auxiliar tiempo " %
tiempo(TiempoX,TiempoY,TiempoX):- TiempoY >= TiempoX.
tiempo(TiempoX,TiempoY,TiempoY).
```

Para seguir, mostramos el predicado inmersion/3, que consiste en si un buzo de la pareja ha agotado su tiempo de inmersión sale del agua y se le restaura el tiempo al inicial, y si no lo ha agotado se mantiene en el agua con el tiempo del que disponga. El primer argumento es la lista con el estado actual del equipo, el segundo es la lista con los tiempos iniciales de los buzos, y el tercero es la lista con los nuevos estados del equipo.

```
% " Metodo auxiliar inmersion " %
inmersion([],[],[]).
inmersion([buzo(_X,TiempoX)|Equipo1],[buzo(X,TiempoY)|TiempoBuzo],[buzo(X,TiempoY)|EquipoInm]):- TiempoX == 0,
                                                                                                  inmersion(Equipo1,TiempoBuzo,EquipoInm).

inmersion([buzo(X,TiempoX)|Equipo1],[buzo(_X,TiempoY)|TiempoBuzo],[buzo(X,TiempoX)|EquipoInm]):- TiempoX > 0,
                                                                                                 inmersion(Equipo1,TiempoBuzo,EquipoInm).
```

A continuación mostramos el predicado actualizarTiempo/3, que consiste en, dada una pareja, la lista con los estados actuales del equipo y la lista con los estados del equipo tras la inmersión de la pareja, como su nombre indica, actualizar el tiempo de ambos miembros de la pareja con respecto a la lista de estados actuales, en el tercer argumento que será el resultado de restar tiempo a los buzos.

```
% " Metodo auxiliar actualizarTiempo " %
actualizarTiempo(_X,[],[]).
actualizarTiempo(pareja(X,Y,T),[buzo(A,TiempoA)|Equipo],[buzo(A,TiempoB)|Equipo1]):- A == X,
                                                                                     TiempoB is TiempoA - T,
                                                                                     actualizarTiempo(pareja(X,Y,T),Equipo, Equipo1).

actualizarTiempo(pareja(X,Y,T),[buzo(A,TiempoA)|Equipo],[buzo(A,TiempoB)|Equipo1]):- A == Y,
                                                                                     TiempoB is TiempoA - T,
                                                                                     actualizarTiempo(pareja(X,Y,T),Equipo, Equipo1).

actualizarTiempo(pareja(X,Y,T),[buzo(A,TiempoA)|Equipo],[buzo(A,TiempoA)|Equipo1]):- distinto(A,X),
                                                                                     distinto(A,Y),
                                                                                     actualizarTiempo(pareja(X,Y,T),Equipo, Equipo1).
```

Para finalizar con los predicados auxiliares, mostramos el predicado reparacionAux/4, que tiene como argumentos la lista con el equipo y el estado actual de su tiempo, el tiempo total del que disponen para reparar el barco, la lista con las parejas según el orden de inmersión y la lista con los tiempos iniciales de los buzos. En este predicado, se utilizan los predicados descritos anteriormente de manera recursiva y en orden adecuado para que la lista de parejas cuadre según los criterios establecidos.

```
% " Metodo auxiliar reparacionAux " %
reparacionAux(Equipo,Tiempo, [],TiempoBuzo):- Tiempo >= 0.
reparacionAux(Equipo,Tiempo,[pareja(X,Y,TiempoF)|Pareja], TiempoBuzo):- subterm(buzo(X,TiempoX),Equipo),
                                                                        subterm(buzo(Y,TiempoY),Equipo),
                                                                        tiempo(TiempoX,TiempoY,TiempoF),
                                                                        Tiempo >= TiempoF,
                                                                        distinto(X,Y),
                                                                        Tiempo1 is Tiempo - TiempoF,
                                                                        actualizarTiempo(pareja(X,Y,TiempoF),TiempoBuzo,Equipo1),
                                                                        inmersion(Equipo1,TiempoBuzo,EquipoInm),
                                                                        reparacionAux(EquipoInm,Tiempo1,Pareja,TiempoBuzo),
                                                                        noesmiembro(pareja(Y,X,T), Pareja),
                                                                        noesmiembro(pareja(X,Y,T), Pareja).
```

Por último, mostramos el predicado principal reparación/3. En este predicado, el cual tiene como argumentos el equipo del que se dispone, el tiempo del que se dispone y la lista de parejas que realizan la inmersión, se realiza la comprobación necesaria de que el número de parejas que se sumerjan sea igual a todas las combinaciones de buzos posibles de 2 en 2 sin repeticiones, es decir que la longitud del tercer argumento sea igual a la suma de los n números enteros, siendo n el número de elementos del primer argumento. A continuación se llama a reparacionAux/4, donde se realizan el resto de especificaciones ya explicadas.

```
% " Metodo reparacion " %
reparacion(Equipo, Tiempo, Pareja):-length(Equipo,NumeroMiembros),
                                    length(Pareja,Combinaciones),
                                    Comb is NumeroMiembros - 1,
                                    combinatoria(Comb,Combinaciones),
                                    reparacionAux(Equipo,Tiempo,Pareja,Equipo).
```

- Consultas

Se proporcionan todos los argumentos

```
1 ?- reparacion([buzo(gomez,45),buzo(lopez,20),buzo(garcia,40),buzo(perez,15)],80,
[pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15), p
areja(lopez, perez, 5), pareja(gomez, perez, 10), pareja(garcia, perez, 5)]).
true ;
false.
```

En esta consulta comprobamos que el predicado reparacion/3 construido comprueba de manera correcta todas las especificaciones.

Se proporciona primer argumento y segundo argumento

```
1 ?- reparacion([buzo(gomez,45),buzo(lopez,20),buzo(garcia,40),buzo(perez,15)],80,L).

L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(gomez, perez, 10), pareja(garcia, perez, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(gomez, perez, 10), pareja(perez, garcia, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(garcia, perez, 10), pareja(gomez, perez, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(garcia, perez, 10), pareja(perez, gomez, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(perez, gomez, 10), pareja(garcia, perez, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(perez, gomez, 10), pareja(perez, garcia, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(perez, garcia, 10), pareja(gomez, perez, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(lopez, perez, 5), pareja(perez, garcia, 10), pareja(perez, gomez, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(perez, lopez, 5), pareja(gomez, perez, 10), pareja(garcia, perez, 5)] ;
L = [pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
pareja(perez, lopez, 5), pareja(gomez, perez, 10), pareja(perez, garcia, 5)]
```

...

En esta consulta vemos que el predicado devuelve de manera correcta el orden en el que las parejas deben hacer la inmersión.

Se proporcionan todos los argumentos. Caso incorrecto

```
1 ?- reparacion([buzo(gomez,45),buzo(lopez,20),buzo(garcia,40),buzo(perez,15)],80
,[pareja(gomez, lopez, 20), pareja(gomez, garcia, 25), pareja(lopez, garcia, 15),
 pareja(lopez, perez, 5), pareja(gomez, perez, 10), pareja(gomez, perez, 10)]).
false.
```

- Comentarios

En el desarrollo de este ejercicio hemos encontrado diferentes dificultades, entre las que prima el entendimiento del predicado, los problemas con la recursividad y el orden de predicados para no caer en bucle infinito y sobretodo, en este ejercicio, el entendimiento de la solución del problema, pues las consultas devuelven múltiples soluciones debido a las múltiples posibilidades.