Memoria Práctica 2 de Programación Declarativa

Kiyoshi José Omaza Saldaña
v13m080

Sergio Fontán Llamas
u12m073

Pablo Puado García
v13m047

Martes 5 de Junio, 2018

Contenidos

1 Planteamiento de la práctica

Una fábrica metalúrgica se está haciendo famosa gracias a unas cadenas que produce que han sido diseñadas por ella y resultan especialmente resistentes. Una de las claves del éxito está en los eslabones que emplean, que tienen un cierre especial en cada uno de sus dos extremos. Han diseñado cierres de tipos muy distintos, de forma que dos eslabones se pueden ensamblar entre sí por sus extremos solo si ambos extremos tienen el mismo tipo de cierre.

El departamento de calidad está sometiendo las cadenas a pruebas de resistencia, para lo que utiliza cadenas cerradas, esto es, con eslabones ensamblados en línea uno tras otro y el último ensamblado con el primero. Para ayudarse en sus pruebas han solicitado un programa informático con las siguientes características. La realización del correspondiente código Prolog es lo que se solicita a los alumnos en esta práctica.

El predicado **cierre/2** debe tener éxito cuando su primer argumento es una lista de eslabones sin repeticiones y el segundo una lista que representa una cadena cerrada de eslabones de la primera lista (todos o solo algunos). Solo es necesario que funcione correctamente cuando el primer argumento se da en la llamada.

Los eslabones se representan con términos $eslabon/2$ cuyos argumentos representan los tipos de cierre en cada uno de sus extremos. Un eslabón con cierres tipo a y b se puede representar como $eslabon(a, b)$ o $eslabon(b, a)$ pero las listas de eslabones (que no tienen repeticiones) solo contendrán uno de ellos. Los tipos de cierres de un mismo eslabón nunca son iguales. Una lista de eslabones que representa una cadena cerrada debe contenerlos en el orden en que ensamblan y el último eslabón debe poder ensamblar con el primero.

Desgraciadamente, una cadena cerrada se puede representar como lista de eslabones de muchas formas distintas, dependiendo del que se coloque como primer elemento: $[A, B, C, D]$, $[B, C, D, A]$, $[C, D, A, B]$ y $[D, A, B, C]$ representarían la misma cadena cerrada. Esto también ocurre si la cadena se recorre en un sentido o en el otro: $[D, C, B, A]$ también representa la misma cadena cerrada. Algo similar puede ocurrir por otras razones, pero en todos los casos en que ocurre la diferencia entre las listas está en el orden, no en los eslabones que contienen.

Como lo que interesa de las cadenas cerradas para las pruebas de resistencia son los eslabones que la forman, no el orden en que se ensamblan, esa multiplicidad de listas que representan la misma cadena resulta excesiva. Así que de todas las listas de cadenas cerradas que contengan los mismos elementos

pero en distinto orden basta con obtener solo una de ellas. También va a interesar obtener el número mínimo de eslabones necesarios para formar una cadena cerrada.

El predicado **cierreUnico/2** debe comportarse igual que *cierre*/2 pero de forma que cada solución que devuelva en su segundo argumento no debe contener los mismos elementos que otra de las soluciones que devuelve. Es decir, en el caso $[A, B, C, D]$ del ejemplo anterior solo debe devolver una de las (ocho) listas posibles. En cambio, el predicado *cierre*/2 debe devolver siempre todas, sin importar que solo difieran en el orden.

El predicado **cierreMinimo/2** debe tener éxito cuando su primer argumento es una lista de eslabones sin repeticiones y su segundo argumento es el mínimo número de eslabones que contiene que pueden formar cadena cerrada. En este predicado se valorará positivamente que el código no obtenga todas las soluciones posibles para luego compararlas.

Cuando se ensamblan físicamente dos eslabones se orientan de manera que los extremos de cada uno que se van a unir estén enfrentados el uno al otro. Sin embargo, esto no ocurre en el programa. Si la lista inicial de eslabones contiene $eslabon(a, b)$, éste se puede utilizar para una cadena cerrada enlazándolo mediante a con el anterior y mediante b con el siguiente, pero también al revés: mediante b con el anterior y mediante a con el siguiente, aunque el término que aparezca en ambos casos en la lista de la cadena cerrada debe ser $eslabon(a, b)$. (Es decir, no se "re-orienta" poniéndolo como $eslabon(b, a)$).

1.1 Código utilizado

1.1.1 Predicado *remove*/3

```
remove(E, [E|T], T).
remove(E, [H|T], [H|TE]):-
  remove(E, T, TE).
```

Este predicado elimina el elemento que se le indique de dentro de una lista.

1.1.2 Predicado *belong*/2

```
belong(E, [E|_]).
belong(E, [_|T]):-
  belong(E, T).
```

El predicado determina si un elemento pertenece a determinada lista.

1.1.3 Predicado *do_list*/3

```
do_list(N, L):- do_list1(N, [], L).
do_list1(2, L, L):-
  !.
do_list1(N, R, L):-
  N > 0, N1 is N-1, do_list1(N1, [N|R], L).
```

Este predicado genera una lista de 3 hasta N elementos.

1.1.4 Predicado *second_last*/2

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

El predicado obtiene el penúltimo elemento de una lista dada.

1.1.5 Predicado *comb*/3

```
comb(0, _Xs, []).
comb(N, [X|Xs], [X|Cs]):-
  N > 0,
  N1 is N-1,
  comb(N1, Xs, Cs).
comb(N, [_X|Xs], Cs):-
  N > 0,
  comb(N, Xs, Cs).
```

Este predicado devuelve todas las posibles combinaciones de longitud N de una lista dada.

1.1.6 Predicado *perm*/2

```
perm([], []).
perm([X|Xs], PXs):-
  perm(Xs, Ps),
  remove(X, PXs, Ps).
```

Este predicado devuelve todas las posibles permutaciones de una lista dada.

1.1.7 Predicado $comb_N/3$

```
comb_N(N, L, X):-
    do_list(N, Ln),
    belong(A, Ln),
    apply_CyP(A, L, X).
```

Este predicado es una combinación de tres otros predicados (aparecen en el código) y devuelve todas las combinaciones, ya permutadas, de tamaño N de una lista dada.

1.1.8 Predicado $comb_one/3$

```
comb_one(N, L, X):-
    do_list(N, Ln),
    belong(A, Ln),
    comb(A, L, X).
```

Al igual que el predicado anterior, este está formado por otros del código. Y en este caso se obtienen varias combinaciones distintas, cada una de un tamaño distinto, desde tamaño 3 a N.

1.1.9 Predicado $apply_CyP/3$

```
apply_CyP(N, L, X):-
    comb(N, L, Xs),
    perm(Xs, X).
```

Es una combinación al igual que los anteriores, y en este caso se obtienen todas las posibles combinaciones, ya permutadas, de una lista.

1.1.10 Predicado $is_linked/3$

```
is_linked(eslabon(X,_), eslabon(X,_), X).
is_linked(eslabon(X,_), eslabon(_,X), X).
is_linked(eslabon(_,X), eslabon(_,X), X).
is_linked(eslabon(_,X), eslabon(X,_), X).
```

Se utiliza para comprobar si dos eslabones están enlazados entre ellos.

1.1.11 Predicado $is_band/3$

```
is_band(A, B, C):-
  is_linked(A, B, X),
  is_linked(B, C, Y),
  X\=Y.
```

Este predicado comprueba si 3 eslabones dados son adyacentes entre ellos.

1.1.12 Predicado $vertex_same/1$

```
vertex_same([eslabon(X, Y)]):-
  X==Y.
vertex_same([eslabon(X, Y)|_]):-
  X==Y.
vertex_same([eslabon(_,_)|Xs]):-
  vertex_same(Xs).
```

Este predicado comprueba si los extremos del eslabón son iguales o distintos.

1.1.13 Predicado $eslabon_same/1$

```
eslabones_same([eslabon(A, B), Y|Xs]):-
  remove(eslabon(A, B), [eslabon(A, B), Y|Xs], [Y|Xs]),
  belong(eslabon(A, B), [Y|Xs]).
eslabones_same([eslabon(A, B), Y|Xs]):-
  remove(eslabon(A,B), [eslabon(A, B), Y|Xs], [Y|Xs]),
  belong(eslabon(B, A), [Y|Xs]).
eslabones_same([_|Xs]):-
  eslabones_same(Xs).
```

Este predicado comprueba si los eslabones son iguales ($eslabon(a,b) = eslabon(b,a)$).

1.1.14 Predicado $eslabon_all/1$

```
eslabones_all([A, B, C]):-
  is_band(A, B, C),
  is_band(B, C, A),
  is_band(C, A, B).
eslabones_all([A, B, C|Cs]):-
  Cs\=[],
  last([A, B, C|Cs], U),
  second_last([A, B, C|Cs], P),
  is_band(P, U, A),
  is_band(U, A, B),
```

```
11    right_band([A, B, C|Cs]).
```

Este predicado recibe como argumento una lista. El predicado devuelve todas las cadenas posibles que se pueden obtener a partir de la lista de eslabones dada.

1.1.15 Predicado $right_band/1$

```
1  right_band([A, P, U]):-
2    is_band(A, P, U).
3  right_band([A, B, C|Cs]):-
4    Cs\=[],
5    is_band(A, B, C),
6    right_band([B, C|Cs]).
```

Este predicado esta hecho para recorrer las cadenas que se van formando.

1.1.16 Predicado $join_eslabon/2$

```
1  join_eslabon(L, X):-
2    perm(L, X),
3    eslabones_all(X),
4    !.
```

Este predicado une las cadenas dadas en una misma.

1.1.17 Predicado $join_eslabon/2$

```
1  not(Goal) :-
2       call(Goal),
3       !,
4       fail.
5  not(_).
```

Simple predicado auxiliar para negar un elemento.

1.1.18 Predicado $cierre/2$

```
1  cierre(Le, X):-
2    length(Le, T), T >= 3,
3    not(vertex_same(Le)),
4    not(eslabones_same(Le)),
```

```
5    comb_N(T, Le, X),
6    eslabones_all(X).
```

Este es uno de los tres métodos principales dados por la práctica. Este predicado recibe una lista de eslabones y devuelve otra lista con todas las posibles cadenas que se pueden formar con los eslabones del primer argumento. Las cadenas que se obtienen son todas correctas y cumplen los requisitos establecidos por la práctica.

1.1.19 Predicado *cierreMinimo*/2

```
1 cierreMinimo(L, N):-
2    cierre(L, S), !, size(S, N).
```

Este es otro de los tres métodos principales dados por la práctica. Este predicado es una extensión del método *cierre*/2, pero en este caso se devuelve el número mínimo de eslabones de la lista del argumento uno con los que se puede formar una cadena correctamente.

1.1.20 Predicado *cierreUnico*/2

```
1 cierreUnico(LE, Xs):-
2    length(LE, T), T >= 3,
3    not(vertex_same(LE)),
4    not(eslabones_same(LE)),
5    comb_one(T, LE, X),
6    join_eslabon(X, Xs).
```

Este es otro de los tres métodos principales dados por la práctica. Al igual que *cierreUnico*/2 es una extensión de *cierre*/2, pero en esta ocasión el predicado devuelve una lista de cadenas en las que los eslabones ya usados en una cadena no van a poder estar en otra cadena.

1.2 Pruebas realizadas

2 Cuadrados fantásticos y secretos

Un cuadrado fantástico es una tabla o matriz de números enteros pares, de tal forma que la suma de los números por columnas y por filas es la misma (es decir, todas las columnas y todas las filas suman la misma cantidad). Los números pares empleados para rellenar las casillas de dicho cuadrado son consecutivos, de 2 a $(2 * N)$, siendo N el número de columnas y filas del

cuadrado fantástico.
Por ejemplo, las siguiente matriz es un cuadrado fantástico en el que todas las columnas y todas las filas suman 12.

2	4	6
6	2	4
4	6	2

Un cuadrado fantástico es secreto cuando cumple la propiedad de las tres esquinas. Esta propiedad se verifica cuando 2 esquinas están rellenas con el mismo número par (A), y la suma de las otras dos esquinas es igual a dicho par ($B + C = A$). Dicho número par (A) se denomina número secreto.
Por ejemplo, la siguiente matriz es un cuadrado fantástico secreto ($A = 8; B = 2; C = 6$) cuyo número secreto es 8.

2	4	6	8
4	6	8	2
6	8	2	4
8	2	4	6

Se plantea la realización de un programa lófico puro en el que las matrices se representan como listas de listas (es decir, una matriz es una lista cuyos elementos son listas de números, en representación de Peano, que representan las filas).
Se pide que los alumnos escriban el predicado $esCuadradoFantasticoSecreto/2$ que se verifique si su primer argumento (una matriz $M * M$) representa un cuadrado fantástico 2 secreto y su segundo argumento es el número secreto.
Por ejemplo:

```
1 ?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(
      s(0))))))],[s(s(s(s(0)))),s(s(s(s(s(s(0)))))),s(s(0))],[s(s
      (s(s(s(0))))))),s(s(0)),s(s(s(s(0))))]],s(s(s(s(s(s(0)))))))).
2 yes
```

2.1 Código utilizado

2.1.1 Predicado *primerElemento*/2

```
1 primerElemento([X|_],X).
```

Este predicado tiene como argumentos una lista y un elemento dado (que puede ser una lista).
La consulta devuelve un valor true si el elemento dado es la cabeza de la lista que se le esta pasando, y en caso de que sea una matriz se devuelve la primera fila. En caso contrario devolverá false.

2.1.2 Predicado *ultimoElemento*/2

```
ultimoElemento([Y],Y).
ultimoElemento([_|Xs],Y):-ultimoElemento(Xs,Y).
```

Este predicado tiene como argumentos una lista y un elemento dado (que puede ser una lista).
La consulta devuelve true si el elemento dado es el último elemento de la lista que se le esta pasando, y en caso de que sea una matriz se devuelve la última fila. En caso contrario devolverá false.

2.1.3 Predicado *suma*/2

```
suma(X,0,X).
suma(A,s(B),s(Y)):-suma(A,B,Y).
```

A este predicado se le pasan como argumentos tres elementos. La consulta devuelve true si el tercer elemento dado es la suma de los dos anteriores. En caso contrario se devuelve false.

2.1.4 Predicado *triplicar*/2

```
triplicar(0,0).
triplicar(s(X),s(s(s(Y)))):-triplicar(X,Y).
```

A este predicado se le pasan como argumentos dos elementos. La consulta devuelve true si el segundo elemento dado tiene un valor igual al triple del primer elemento. En caso contrario devolverá false.

2.1.5 Predicado *esCuadradoFantasticoSecreto*/2

```
esCuadradoFantasticoSecreto(M,S):-
   primerElemento(M,L1),
   primerElemento(L1,X1),

   ultimoElemento(M,Ln),
   ultimoElemento(Ln,Xn),

   suma(X1,Xn,S1),

   ultimoElemento(L1,X2),
   primerElemento(Ln,Xn1),

```

```
13    suma(X2,Xn1,S2),
14    suma(S1,S2,S3),
15
16    triplicar(S,S3).
```

Este predicado tiene como argumentos una lista y un elemento dado. La consulta devuelve true si el elemento es *número secreto*. En caso contrario devolverá false.

2.2 Pruebas realizadas

```
1 ?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(s(
      s(0))))))],[s(s(s(s(0)))),s(s(s(s(s(s(0))))))),s(s(0))],[s(s(s
      (s(s(s(0))))))),s(s(0)),s(s(s(s(0))))]],s(s(s(s(s(s(0)))))))).
2 yes
3
4 ?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(s(
      s(0))))))],[s(s(s(s(0)))),s(s(s(s(s(s(0))))))),s(s(0))],[s(s(s
      (s(s(s(0))))))),s(s(0)),s(s(s(s(0))))]],S).
5 S = s(s(s(s(s(s(0))))))
6
7 ?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(s(
      s(0))))))),s(s(s(s(s(s(s(0)))))))],[s(s(s(s(0)))),s(s(s(s(s
      (s(0))))))),s(s(s(s(s(s(s(0)))))))),s(s(0))],[s(s(s(s(s(s(0)
      )))))),s(s(s(s(s(s(s(0)))))))),s(s(0)),s(s(s(s(0))))],[s(s(s
      (s(s(s(s(0)))))))),s(s(0)),s(s(s(s(0)))),s(s(s(s(s(s(0))))))
      )]],S).
8 S = s(s(s(s(s(s(s(0)))))))
```