Ejs-redes-semanticas-resueltos.pdf

JDTadasGOT

Inteligencia Artificial II

3º Grado en Ingeniería Informática

Facultad de Informática
Universidad Complutense de Madrid

Ejercicios IA - 2017/2018

Tema 6.1

1) Se tiene la siguiente red semántica representada en Prolog:

```
es_un(elefante_circense, elefante).
tiene_parte(elefante, cabeza).
tiene_parte(elefante, trompa).
tiene_parte(cabeza, boca).
es_un(elefante, animal).
tiene_parte(animal, corazón).
es_un(elefante_circense, acróbata).
tiene_parte(acróbata, disfraz).
es_un(disfraz, ropa).
```

¿Qué reglas generales sería necesario añadir para que el sistema pudiera contestar afirmativamente a las siguientes preguntas?

```
?- es(elefante_circense, animal).
?- tiene(elefante_circense, corazón).
?- tiene(elefante_circense, boca).
?- tiene(elefante_circense, ropa).
```

2) Resuelve los siguientes apartados:

a) Representa en una red semántica los siguientes hechos:

Juan le prestó su coche a María.

Después María se lo prestó a Pablo.

Esto provocó que Juan se enfadara con María.

b) Representa en Prolog la red semántica del apartado anterior.

c) Añade a la representación en Prolog los siguientes hechos:

El coche de Juan es un Seat.

Los Seat son coches.

Los coches son vehículos.

Los coches tienen sistema eléctrico.

Los sistemas eléctricos tienen batería.

Las baterías tienen ácido.

El ácido es un producto químico.

d) ¿Qué reglas generales sería necesario añadir para que el intérprete de Prolog pudiera contestar afirmativamente a las siguientes preguntas?

¿El coche de Juan es un vehículo?

¿El coche de Juan tiene sistema eléctrico?

¿El coche de Juan tiene batería?

¿El coche de Juan tiene un producto químico?

3) Se quiere construir una red semántica que incluya relaciones *es_un* y *tiene_parte* entre los objetos representados. Supongamos que el concepto *B* es una especialización del concepto *A* porque existe un camino de *B* a *A* formado por *N* aristas *(N>=1)* etiquetadas con la relación *es_un*. Del concepto *A* sale una arista etiquetada con la propiedad *tiene_parte* que acaba en el concepto *C*. Queremos que el concepto *B* herede esta misma propiedad, para lo cual vamos a considerar que cada arista viene representada por un predicado *arista/3*:

```
arista(NombreRelación, Origen, Destino)
```

Por ejemplo, *arista(tiene_parte, A, C).*

Se pide definir una regla totalmente general en Prolog que permita heredar cualquier propiedad a través de cualquier relación. Las propiedades concretas que queramos que se hereden a través de relaciones concretas se establecerán como hechos con el predicado *hereda/2*. Por ejemplo, en este caso, tendríamos *hereda(tiene_parte, es_un)* para indicar que se hereda la propiedad *tiene_parte* a través de la relación *es_un*, pero la regla general debe permitir gestionar la herencia para cualquier propiedad y relación.

4) Se quiere implementar en Prolog un sistema basado en redes semánticas que permita representar cualquier tipo de relación binaria entre conceptos. Para ello se pide:

a) Implementar un predicado que, dados dos conceptos, determine si están conectados por un camino formado por N aristas etiquetadas igual (y ninguna otra distinta). Es decir, si los dos conceptos están relacionados por una misma relación repetida N veces:

```
relacion_repetida(+Concepto1, +Concepto2, ?Relación, ?NumRepeticiones)
```

El predicado devolverá la correspondiente relación y el número de veces que se repite.

b) Definir otro predicado que permita buscar la relación entre dos conceptos

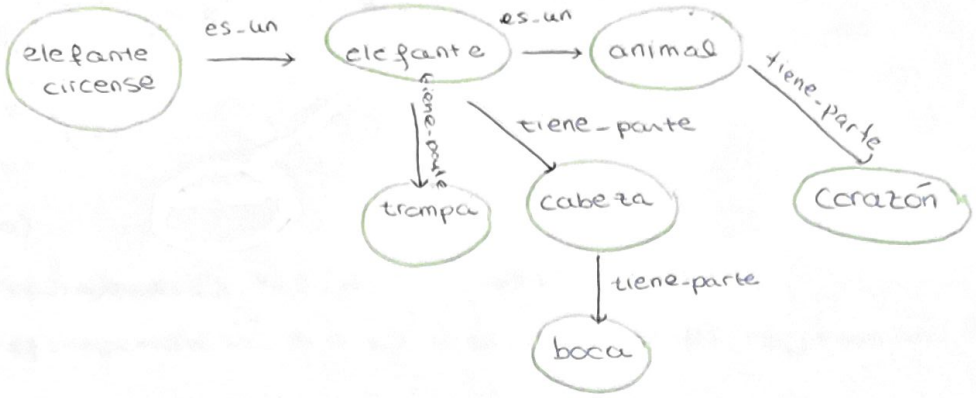
```
busq_relacion(+Concepto1, +Concepto2, ?CadenaDeRelaciones)
```

El predicado devolverá una lista formada por los nombres de los conceptos y de las relaciones que constituyen un camino entre Concepto1 y Concepto2 (si existe). Por ejemplo, al preguntar la relación entre el concepto coche1 y el concepto miguel, una posible respuesta sería [coche1, tiene_dueño, juan, hermano_de, miguel].

Para simplificar, asúmase que no existen ciclos.

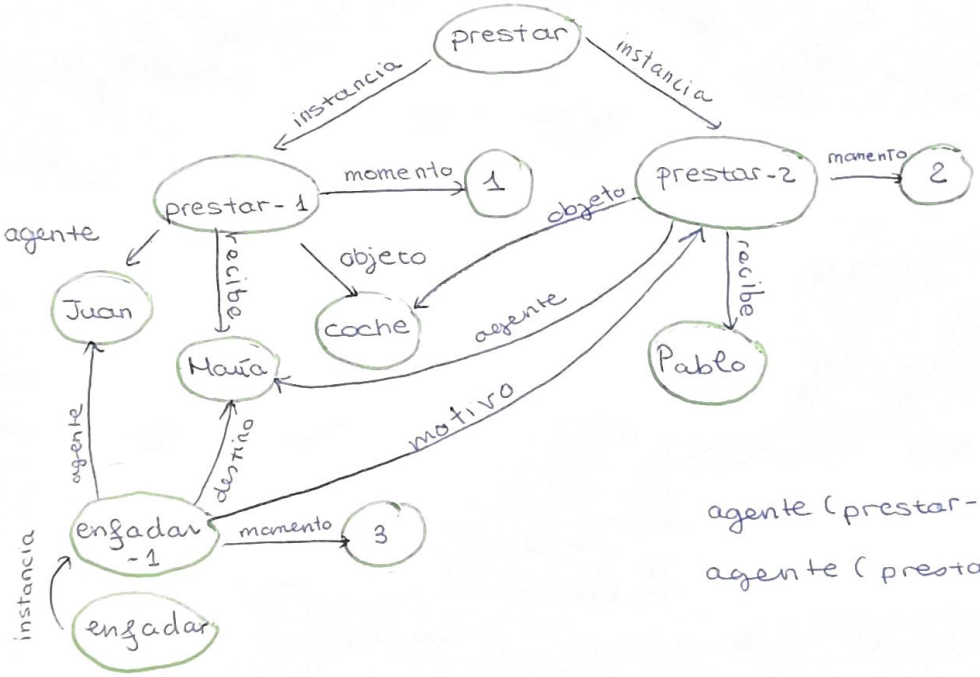
Hoja 4 T6

1)



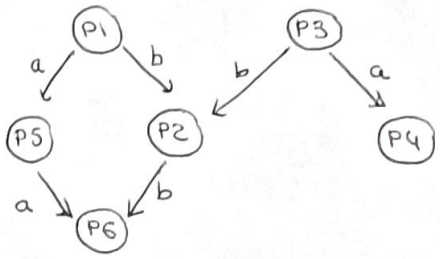

```
es_un_trans(X,Y) :- es_un(X,Y).
es_un_trans(X,Y) :- es_un(X,Z), es_un(Z,Y).

hereda_tiene_parte(X,Y) :- tiene_parte_trans(X,Y).
hereda_tiene_parte(X,Y) :- es_un_trans(X,Z), hereda_tiene_parte(Z,Y).
```

2)



agente(prestar-1, Juan).
agente(prestar-2, Maria).

4)



arista(X, Y, R)

arista(P1, P2, ra)

arista(P5, P6, ra)

a)

```
ref-repetida(X, Y, R, N) :- arista(X, Y, Z).
ref-repetida(X, Y, R, N) :- arista(X, Z, R), refrepetida(Z, Y, R, M), N is M+1.
```

b)

```
camino(X, X, [X]).
camino(X, Y, [X, R | Resto]) :- arista(X, Z, R), camino(Z, Y, [Resto])
```

```
/*
  EJERCICIO 1

  En este ejercicio se nos da una base de hechos y se nos pide averiguar
  las reglas generales necesarias para que el sistema pueda contestar a
  ciertas preguntas de forma afirmativa.
*/

% Base de hechos

es_un(elefante_circense, elefante).
es_un(elefante, animal).
es_un(elefante_circense, acr□bata).
es_un(disfraz, ropa).

tiene_parte(elefante, cabeza).
tiene_parte(elefante, trompa).
tiene_parte(cabeza, boca).
tiene_parte(animal, coraz□n).
tiene_parte(acr□bata, disfraz).

/* Preguntas a las que queremos que se conteste afirmativamente

?- es_un(elefante_circense, animal).
?- tiene_parte(elefante_circense, coraz□n).
?- tiene_parte(elefante_circense, boca).
?- tiene_parte(elefante_circense, ropa).

Hace falta a□adir las reglas de transitividad y herencia usando
tiene_parte y es_un. Y un arreglito m□s por la □ltima consulta y
la pinta de la BC
*/

% Lo que falta es establecer la transitividad de estas relaciones y la herencia
% de la propiedad tiene_parte con respecto a la relaci□n es_un

% Para evitar los problemas de caminos infinitos, definimos otros predicados
% que ser□n con los que reformularemos las consultas.

% Cierre transitivo de la relaci□n es_un (subclase, aqu□):
% Es subir por la jerarqu□a
es_un_trans(X, Y) :- es_un(X, Y).
es_un_trans(X, Y) :- es_un(X, Z), es_un_trans(Z, Y).

% Cierre transitivo de la relaci□n tiene_parte:
% Es como acceder recursivamente a campos de atributos
tiene_parte_trans(X, Y) :- tiene_parte(X, Y).
tiene_parte_trans(X, Y) :- tiene_parte(X, Z), tiene_parte_trans(Z, Y).

% Herencia hacia abajo de la propiedad tiene_parte_trans con respecto
% a la relaci□n es_un
hereda_tiene_parte(X, Y) :- tiene_parte_trans(X, Y).
hereda_tiene_parte(X, Y) :- es_un(X, Z), hereda_tiene_parte(Z, Y).
```

```
tiene(X, Y) :- hereda_tiene_parte(X, Y).
tiene(X, Y) :- hereda_tiene_parte(X, Z), es_un_trans(Z, Y).

% Con lo anterior aseguramos la convergencia de las consultas
% Podr□amos hacer una "guarrer□a", consistente en mezclarlo todo
mezcla(X, Y) :- es_un_trans(X, Y).
mezcla(X, Y) :- tiene_parte_trans(X, Y).
mezcla(X, Y) :- es_un(X, Z), mezcla(Z, Y).
mezcla(X, Y) :- tiene_parte(X, Z), mezcla(Z, Y).

/* Preguntas reformuladas a las que el sistema contesta afirmativamente

?- es_un_trans(elefante_circense, animal).
?- tiene(elefante_circense, coraz□n).
?- tiene(elefante_circense, boca).
?- tiene(elefante_circense, ropa).

?- tiene(elefante_circense, Y).

Y = cabeza ;
Y = trompa ;
Y = boca ;
Y = coraz□n ;
Y = disfraz ;
Y = ropa ;
No

?- es_un_trans(elefante_circense, X).
X = elefante ;
X = acr□bata ;
X = animal ;
false.

?- mezcla(elefante_circense, Y).

Y = elefante ;
Y = acr□bata ;
Y = animal ;
Y = animal ;
Y = cabeza ;
Y = trompa ;
Y = boca ;
Y = coraz□n ;
Y = boca ;
Y = disfraz ;
Y = ropa ;
No
*/
```

```
/* EJERCICIO 2
  (b) Representa en Prolog una red sem□ntica formada por los siguientes hechos:
            Juan le prest□ su coche a Mar□a.
            Despu□s Mar□a se lo prest□ a Pablo.
            Esto provoc□ que Juan se enfadara con Mar□a.

  (c) A□ade a la representaci□n en Prolog los siguientes hechos:
           El coche de Juan es un Seat.
           Los Seat son coches.
           Los coches son veh□culos.
           Los coches tienen sistema el□ctrico.
           Los sistemas el□ctricos tienen bater□a.
           Las bater□as tienen □cido.
           El □cido es un producto qu□mico.

  (d) □Qu□ reglas generales ser□a necesario a□adir para que el int□rprete de
  Prolog pudiera contestar afirmativamente a las siguientes preguntas?
                El coche de Juan es un veh□culo.
                El coche de Juan tiene sistema el□ctrico.
                El coche de Juan tiene bater□a.
                El coche de Juan tiene un producto qu□mico.

*/
% Base de hechos (A y B)

ejemplar(prestar_1, prestar).
ejemplar(prestar_2, prestar).
ejemplar(enfadar_1, enfadar).
ejemplar(coche_1, coche).

agente(prestar_1, juan).
agente(prestar_2, mar□a).
agente(enfadar_1, juan).

beneficiario(prestar_1, mar□a).
beneficiario(prestar_2, pablo).
beneficiario(enfadar_1, mar□a).

tiempo(prestar_1, tiempo_1).
tiempo(prestar_2, tiempo_2).

mayor(tiempo_2, tiempo_1).

objeto(prestar_1, coche_1).
objeto(prestar_2, coche_1).
objeto(enfadar_1, prestar_2).

due□o(juan, coche_1).

% Hechos (C)

ejemplar(coche_1, seat).
```

```
subclase(seat, coche).
subclase(coche, vehiculo).

tiene_parte(coche, sistemaEl□ctrico).
tiene_parte(sistemaEl□ctrico, bater□a).
tiene_parte(bater□a, □cido).

subclase(□cido, productoQu□mico).

/*  (D)
Hace falta a□adir las reglas de transitividad y herencia usando
tieneparte y esun.
*/

% Para evitar los problemas de caminos infinitos, definimos otros predicados
% que ser□n con los que reformularemos las consultas.

% Definimos es_un como ejemplar o subclase (coincidiendo con una confusi□n
% habitual entre estas dos relaciones)
es_un(X, Y) :- ejemplar(X, Y).
es_un(X, Y) :- subclase(X, Y).

% Cierre transitivo de la relaci□n es_un:
es_un_trans(X, Y) :- es_un(X, Y).
es_un_trans(X, Y) :- es_un(X, Z), es_un_trans(Z, Y).

% Cierre transitivo de la relaci□n ejemplar:
ejemplar_trans(X, Y) :- ejemplar(X, Y).
ejemplar_trans(X, Y) :- ejemplar(X, Z), ejemplar_trans(Z, Y).

% Cierre transitivo de la relaci□n subclase:
subclase_trans(X, Y) :- subclase(X, Y).
subclase_trans(X, Y) :- subclase(X, Z), subclase_trans(Z, Y).

% Cierre transitivo de la relaci□n tiene_parte:
tiene_parte_trans(X, Y) :- tiene_parte(X, Y).
tiene_parte_trans(X, Y) :- tiene_parte(X, Z), tiene_parte_trans(Z, Y).

% Herencia hacia abajo de la propiedad tiene_parte_trans con respecto
% a la relaci□n ejemplar y a la relaci□n subclase
hereda_tiene_parte(X, Y) :- tiene_parte_trans(X, Y).
hereda_tiene_parte(X, Y) :- es_un(X, Z), hereda_tiene_parte(Z, Y).

% Herencia pedida
tiene(X, Y) :- hereda_tiene_parte(X, Y).
tiene(X, Y) :- hereda_tiene_parte(X, Z), es_un_trans(Z, Y).
```

```
/* Consultas:

?- due□o(juan, X), ejemplar(X, coche), es_un_trans(X, vehiculo).
X = coche_1 ;

X = coche_1 ;

No
?- tiene(coche_1, sistemaEl□ctrico).

Yes
?- tiene(coche_1, bater□a).

Yes
?- tiene(coche_1, productoQu□mico).

Yes
*/
```

```
/*  EJERCICIO 3

  Se quiere construir una red sem□ntica que incluya relaciones es_un y
  tiene_parte entre los objetos representados.
  Supongamos que el concepto B es una especializaci□n del concepto A porque
  existe un camino de B a A formado por N aristas (N>=1) etiquetadas con la
  relaci□n es_un. Del concepto A sale una arista etiquetada con la propiedad
  tiene_parte que acaba en el concepto C. Queremos que el concepto B herede
  esta misma propiedad, para lo cual vamos a considerar que cada arista viene
  representada por un predicado arista con tres argumentos:
              arista(<nombre relaci□n>, <origen>, <destino>)
  Por ejemplo, arista(tiene_parte, A, C).

  Definir una regla totalmente general en Prolog que permita heredar cualquier
  propiedad a trav□s de cualquier relaci□n. Las propiedades concretas que
  queramos que se hereden a trav□s de relaciones concretas se establecer□n
  como hechos con el predicado hereda. Por ejemplo, en este caso, tendr□amos:
  hereda(tiene_parte, es_un) para indicar que se hereda la propiedad
  tiene_parte a trav□s de la relaci□n es_un, pero la regla general debe
  permitir gestionar la herencia para cualquier propiedad y relaci□n.

*/

% ejemplo de prueba
arista(tiene_parte, a, c).
arista(es_un, b, i).
arista(es_un, i, j).
arista(es_un, j, k).
arista(es_un, k, a).

% hereda(Propiedad, Relacion).
hereda(tiene_parte, es_un).




% Transitividad de la relaci□n arista/3:
arista_trans(R, X, Y) :- arista(R, X, Y).
arista_trans(R, X, Y) :-
                arista(R, X, Z),
                arista_trans(R, Z, Y).




% Regla general para la herencia que aqu□ se pide:
herencia(P, R, X, Y) :-
            hereda(P, R),
```

```
            arista_inferida(P, R, X, Y).




% S□lo reflejamos las aristas inferidas por herencia de P c.r.a R
arista_inferida(P, R, X, Y) :-
                  arista(R, X, Z),
                  arista_trans(P, Z, Y).

arista_inferida(P, R, X, Y) :-
                  arista(R, X, Z),
                  arista_inferida(P, R, Z, Y).




% Aristas existentes e inferidas por herencia:
aristas(P, X, Y) :- arista(P, X, Y).
aristas(P, X, Y) :-
           hereda(P, R),
           arista_inferida(P, R, X, Y).


/*
?- aristas(Z, X, Y).

Z = tiene_parte       X = a  Y = c ;
Z = es_un             X = b  Y = i ;
Z = es_un             X = i  Y = j ;
Z = es_un             X = j  Y = k ;
Z = es_un             X = k  Y = a ;
Z = tiene_parte       X = k  Y = c ;
Z = tiene_parte       X = b  Y = c ;
Z = tiene_parte       X = i  Y = c ;
Z = tiene_parte       X = j  Y = c ;
No

?- herencia(P, R, X, Y).
P = tiene_parte   R = es_un  X = k Y = c ;
P = tiene_parte   R = es_un  X = b Y = c ;
P = tiene_parte   R = es_un  X = i Y = c ;
P = tiene_parte   R = es_un  X = j Y = c ;
No
*/
```

```
/*
  EJERCICIO 4

  Se quiere implementar en Prolog un sistema basado en redes sem□nticas que
  permita representar cualquier tipo de relaci□n binaria entre conceptos.
  Para ello se pide:

  a)  Implementar un predicado que, dados dos conceptos, determine si est□n
  conectados por un camino formado por N aristas etiquetadas igual (y ninguna
  otra distinta). Es decir, si los dos conceptos est□n relacionados por una
  misma relaci□n repetida N veces:
        relacion_repetida(+Concepto1, +Concepto2, ?Relaci□n, ?NumRepeticiones)
  ha de devolver la relaci□n y el n□mero de veces que se repite.

  b)  Definir otro predicado que permita implementar el mecanismo de b□squeda
  de la intersecci□n
        busq_relacion(+Concepto1, +Concepto2, ?CadenaDeRelaciones)
  ha de devolver una lista formada por los nombres de los conceptos y de las
  relaciones que constituyen un camino entre Concepto1 y Concepto2 (si existe).
  Por ejemplo, al preguntar, con la red sem□ntica del ejercicio 2, la relaci□n
entre el concepto coche1 y el concepto juan, una posible respuesta ser□a
          [coche1, due□o, juan].

  Para simplificar, as□mase que no existen ciclos.

*/

/*
% ejemplo de prueba
arista(tiene_parte, a, c).
arista(es_un, b, i).
arista(es_un, i, j).
arista(es_un, j, k).
arista(es_un, k, a).
*/

/*     (A)
La implementaci□n depende por completo de c□mo se representen las relaciones.
Aqu□ suponemos que se representan como en el problema anterior (mediante el
predicado arista(Relaci□n, Concepto1, Concepto2)).

Lo primero que se nos pide no es m□s que una variante del cierre transitivo
de esta relaci□n, contando las aristas por las que vamos pasando.
*/

% relacion_repetida(+Concepto1, +Concepto2, ?Relaci□n, ?NumRepeticiones)
relacion_repetida(X, Y, R, 1) :- arista(R, X, Y).
relacion_repetida(X, Y, R, N) :-
                     arista(R, X, Z),
                     relacion_repetida(Z, Y, R, M),
                     N is M+1.


% busq_relacion(+Concepto1, +Concepto2, ?CadenaDeRelaciones)
```

```
% Camino de X a Y directo (o de Y a X, o de X a X), las flechas van en un nico
sentido
camino(X, X, [X]).
camino(X, Y, [X, R | Cadena]) :-
                arista(R, X, Z),
                camino(Z, Y, Cadena).

% busq_relacion comprende una serie de caminos ms genrica
% que agregan todos los caminos posibles del predicado camino
% as como aquellos que se consiguen con camino de X a Z (flechas de X a Z) y
camino de Y a Z (flechas de Y a Z)
busq_relacion(X, X, [X], [X], X).
busq_relacion(X, Y, [X, R | Izda], [Y], Y) :-
                    arista(R, X, Z),
                    camino(Z, Y, Izda).
busq_relacion(X, Y, [X], [Y, R|Dcha], X) :-
                    arista(R, Y, Z),
                    camino(Z, X, Dcha).
busq_relacion(X, Y, [X, R1 | Izda], [Y, R2 |Dcha], I) :-
                    arista(R1, X, Z1),
                    arista(R2, Y, Z2),
                    busq_relacion(Z1, Z2, Izda, Dcha, I).



arista(progenitor, p1, p5).
arista(progenitor, p1, p2).
arista(progenitor, p3, p2).
arista(progenitor, p3, p4).
arista(progenitor, p5, p6).
arista(progenitor, p7, p8).

/*
arista(r1, c1, c11).
arista(r2, c11, c).
arista(r3, c2, c).
arista(r4, c2, c11).
arista(r5, c2, c1).
*/

/*
?- relacion_repetida(X, Y, R, N).
X = p1                  Y = p5   R = progenitor    N = 1 ;
X = p1                  Y = p2   R = progenitor    N = 1 ;
X = p3                  Y = p2   R = progenitor    N = 1 ;
X = p3                  Y = p4   R = progenitor    N = 1 ;
X = p5                  Y = p6   R = progenitor    N = 1 ;
X = p7                  Y = p8   R = progenitor    N = 1 ;
X = p1                  Y = p6   R = progenitor    N = 2 ;

No
```

```
?- busq_relacion(p1, Y, C, D, I).

Y = p1
C = [p1]
D = [p1]
I = p1 ;

Y = p5
C = [p1, progenitor, p5]
D = [p5]
I = p5 ;

Y = p6
C = [p1, progenitor, p5, progenitor, p6]
D = [p6]
I = p6 ;

Y = p2
C = [p1, progenitor, p2]
D = [p2]
I = p2 ;

Y = p1
C = [p1, progenitor, p5]
D = [p1, progenitor, p5]
I = p5 ;

Y = p1
C = [p1, progenitor, p5, progenitor, p6]
D = [p1, progenitor, p5, progenitor, p6]
I = p6 ;

Y = p5
C = [p1, progenitor, p5, progenitor, p6]
D = [p5, progenitor, p6]
I = p6 ;

Y = p1
C = [p1, progenitor, p2]
D = [p1, progenitor, p2]
I = p2 ;

Y = p3
C = [p1, progenitor, p2]
D = [p3, progenitor, p2]
I = p2 ;

No
*/
```