```
------------------------------------------------
Teoría de Modelos. Clase del 15 de abril de 2020
------------------------------------------------
------------------------------------------------
Modelización de Bases de Conocimiento en ASP
------------------------------------------------
([GK], páginas 77-87)
------------------------------------------------
1. Base de datos de una familia.
------------------------------------------------

-----------------------------------------
Se dispone de la siguiente información:
 Juan es el padre de Pablo.
 Alicia es madre de Pablo.

     Juan   Alicia
      \      /
        Pablo
-----------------------------------------

% Usamos los predicados persona/1 y genero/1 para clasificar los
elementos
% que componen la familia:

persona(juan).
persona(pablo).
persona(alicia).

genero(hombre).
genero(mujer).

% Las relaciones entre los objetos del dominio se modelizan mediante los
% predicados: padre/2, madre/2, progenitor/2, hijo/2; y el predicado
% genero_de/2 (sobre personas y géneros).

padre(juan,pablo).
madre(alicia,pablo).

genero_de(juan,hombre).
genero_de(alicia,mujer).
genero_de(pablo,hombre).

progenitor(X,Y) :- padre(X,Y).
progenitor(X,Y) :- madre(X,Y).

hijo(X,Y) :- progenitor(Y,X).

% clingo version 4.5.4
```

```
% Solving...
% Answer: 1

% persona(juan) persona(pablo) persona(alicia) genero(hombre)
% genero(mujer) padre(juan,pablo) madre(alicia,pablo)
% genero_de(juan,hombre) genero_de(alicia,mujer) genero_de(pablo,hombre)
% progenitor(juan,pablo) progenitor(alicia,pablo) hijo(pablo,juan)
% hijo(pablo,alicia)


------------------------------------------------------------
2. Ampliación de la base de datos con la noción de HERMANO
------------------------------------------------------------

------------------------------------------------------------------
Ampliamos la información de nuestra base de conocimiento:
  Juan y Alicia tienen otro hijo, Pedro.

          Juan           Alicia
           \_ _ _ _ _ _ _ __/
              Pablo  Pedro
------------------------------------------------------------------

% Ampliamos la representación familiar añadiendo la información
% sobre el nuevo objeto Pedro (hechos H1,H2,H3,H4).

persona(juan).
persona(pablo).
persona(alicia).

genero(hombre).
genero(mujer).

padre(juan,pablo).
madre(alicia,pablo).

genero_de(juan,hombre).
genero_de(alicia,mujer).
genero_de(pablo,hombre).

progenitor(X,Y) :- padre(X,Y).
progenitor(X,Y) :- madre(X,Y).

hijo(X,Y) :- progenitor(Y,X).

persona(pedro).                %H1
padre(juan,pedro).             %H2
madre(alicia,pedro).           %H3
genero_de(pedro,hombre).       %H4
```

```
% Ahora debemos definir el predicado hermano/2.
% Primer intento de modelar el predicado hermano(X,Y)
% (X es un hermano de la persona Y)
%  hermano(X,Y) = X es un hombre, X e Y tienen el mismo
%  padre y la misma madre.

hermano1(X,Y) :- genero_de(X,hombre),
                 padre(F,X),
                 padre(F,Y),
                 madre(M,X),
                 madre(M,Y).

% clingo version 4.5.4
% Solving...
% Answer: 1

% persona(juan) persona(pablo) persona(alicia) genero(hombre)
% genero(mujer) padre(juan,pablo) madre(alicia,pablo)
% genero_de(juan,hombre) genero_de(alicia,mujer) genero_de(pablo,hombre)
% persona(pedro) padre(juan,pedro) madre(alicia,pedro)
% genero_de(pedro,hombre) progenitor(juan,pablo) progenitor(juan,pedro)
% progenitor(alicia,pablo) progenitor(alicia,pedro) hijo(pablo,juan)
% hijo(pedro,juan) hijo(pablo,alicia) hijo(pedro,alicia)
% **hermano1(pablo,pablo)** hermano1(pablo,pedro) hermano1(pedro,pablo)
% **hermano1(pedro,pedro)**

Problema: Los hechos hermano1(pablo,pablo) y hermano1(pedro,pedro)
aparecen
en el conjunto de respuestas. No hemos incluido el conocimiento implícito

de que una persona no puede ser hermano se sí mismo. Para solucionar
esto,
necesitamos ampliar la sintaxis de CLINGO con un operador predefinido:
el operador distinto (!=). El átomo t1 != t2 será verdadero cuando los
términos t1 y t2 sean distintos.

% La representación correcta es:

hermano(X,Y) :- genero_de(X,hombre),
                padre(F,X),
                padre(F,Y),
                madre(M,X),
                madre(M,Y),
                X != Y.

% Nota: Análogamente, disponemos del operador predefinido *igual* (==).
% El átomo t1 == t2  será verdadero si los términos t1 y t2 son iguales.
```

```
% Sin embargo, para definir hermano/2 no hemos necesitado el operador
(==)
% porque, por ejemplo, para decir que X e Y tiene el mismo padre basta
con
% usar la misma variable F dentro del cuerpo de la regla
% (...,padre(F,X),padre(F,Y),...)

-----------------------------------------------------
3. Representación de información implícita negativa
-----------------------------------------------------

En el proceso de representación del conocimiento se presentan dos
factores importantes que han de tenerse siempre en cuenta:
  (*) El conocimiento implícito
  (*) El sentido común

Observa que, por el momento, la respuesta de nuestro programa a la
pregunta:
"¿Es Alicia el padre de Pedro?" sería: Desconocido.
(puesto que ni padre(alicia,pedro) ni -padre(alicia,pedro) aparecen en su

 único conjunto de respuestas).

Sin embargo, el sentido común nos dice que la respuesta debería ser
un categórico NO. Sabemos que Alicia es una mujer y si una persona
es mujer entonces no puede ser el padre de nadie. Ahora bien,
no hemos introducido ninguna regla en nuestro programa para
representar dicho conocimiento implícito. Podemos reparar esta
situación con la siguiente regla que añade información negativa
sobre el predicado padre:

-padre(X,Y) :- genero_de(X,mujer),persona(Y).  %R1

------------------------
Nota(*Reglas seguras*):
------------------------
Observa que en el cuerpo de la regla R1 anterior hemos añadido
el átomo persona(Y) y no hemos escrito simplemente:

-padre(X,Y) :- genero_de(X,mujer).   %R1'

Esto es así porque CLINGO añade una restricción sintáctica adicional
a las reglas que podemos usar en un programa lógico. Una variable V
se dirá segura en una regla R si aparece al menos una vez en el
*cuerpo* de la regla de forma positiva (es decir, sin estar
afectada de la negación por defecto *not*, aunque sí que podría
estar afectada de la negación fuerte -).

Observa que la regla R1' anterior NO es segura, porque la variable Y
```

no ocurre en el cuerpo de R1'. Es por ello que nos vemos obligados a añadir el átomo persona(Y) y ahora la regla sí es segura.

Otro ejemplo de regla NO segura sería:

```
-hermano(X,Y) :- not hermano(X,Y).
```

Las variables X,Y aparecen en el cuerpo de la regla, pero están afectadas de la negación por defecto *not*. Podemos hacer la regla segura añadiendo:

```
-hermano(X,Y) :- not hermano(X,Y),
                 persona(X),persona(Y).
```

Consideramos ahora una segunda pregunta a nuestro programa:
"¿Puede ser Pedro el padre de Pablo?"
De nuevo, la respuesta del programa sería "Desconocido"
(porque ni -padre(pedro,pablo) ni padre(pedro,pablo) aparecen
en el conjunto de respuestas).
Sin embargo, aplicando el sentido común la respuesta debería ser un claro NO: sabemos que Juan es el padre de Pablo y una persona no puede tener más de un padre. Podemos reparar esta situación mediante la siguiente regla que de nuevo añade información negativa sobre el predicado padre:

```
-padre(X,Y) :- padre(Z,Y),
               X != Z,
               persona(X).
```

Nota: Es necesario añadir el átomo persona(X) en el cuerpo de la regla para que la regla sea segura. La ocurrencia de X en el átomo X!=Z no cuenta para hacer la regla segura por estar afectada del operador distinto (!=).

---

4. Representación de información implícita. Razonamiento por defecto con *NOT*

Introducimos una nueva persona en la base de conocimiento.

```
persona(roberto).
genero_de(roberto,hombre).
```

Y consideramos la pregunta: "¿Es Juan el padre de Roberto?"
De nuevo, la respuesta de nuestro programa sería: "Desconocido".
Sin embargo, aplicando el sentido común, en nuestra vida diaria

trabajaríamos con la hipótesis de que Juan NO es el padre de Roberto. En efecto, razonaríamos así: "Si Juan fuese el padre de Roberto, tendría constancia de ello. Puesto que no tengo constancia de ello en mi base de conocimiento, por el momento intentaré sacar conclusiones suponiendo que NO lo es. Si más adelante fuese necesario, me retractaría de esta decisión." Este tipo de razonamiento es típico del *RAZONAMIENTO NO MONÓTONO*. En dicho campo, suele nombrarse como "Hipótesis de Mundo Cerrado" (CWA, por sus siglas en inglés).

Afortunadamente, es fácilmente representable en ASP usando el operador *not*. Representamos de forma explícita que si no se tiene certeza de que X sea padre de Y, entonces no lo es.

```
-padre(X,Y) :- persona(X), persona(Y),
               not padre(X,Y).
```

Ahora nuestro programa es capaz de deducir que Juan no es el padre de Roberto. Además, si más adelante tuviésemos conocimiento de que, después de todo, Juan tiene un tercer hijo, que es Roberto; podríamos añadir el hecho

```
padre(juan,roberto).
```

a nuestra base de conocimiento sin generar ninguna contradicción.

---

5. Segundo ejemplo. Definición de HUÉRFANO.

Cambiamos ahora de base de conocimiento y consideremos un nuevo programa ASP.

```
% Consideremos las siguientes personas:

persona(maria).
persona(roberto).
persona(miguel).
persona(rafael).
persona(cati).
persona(patricia).

% Algunos son niños:

ninno(maria).
ninno(roberto).

% Relaciones de parentesco:
```

```
padre(miguel,maria).
padre(rafael,roberto).
madre(cati,maria).
madre(patricia,roberto).

% Algunos han muerto

muerto(rafael).
muerto(patricia).
muerto(cati).

% Representamos la completitud de la información hasta el momento
mediante la hipótesis de mundo cerrado:

-ninno(X) :- persona(X),
             not ninno(X).

-padre(X,Y) :- persona(X),
               ninno(Y),
               not padre(X,Y).

-madre(X,Y) :- persona(X),
               ninno(Y),
               not madre(X,Y).

-muerto(X) :- persona(X),
              not muerto(X).

% Objetivo: modelizar la noción de huérfano.
% Nota: Consideramos que X es huérfano si es menor de edad
% y han fallecido tanto su padre como su madre.

padresMuertos(P) :- padre(F,P),
                    madre(M,P),
                    muerto(F),
                    muerto(M).

huerfano(P) :- ninno(P),
               padresMuertos(P).

-huerfano(P) :- ninno(P),
                not huerfano(P).
```

Nuestro programa es capaz de inferir que Roberto es huérfano
y que María no lo es.

```
-----------------------------------------
6. Tercer ejemplo. Definiciones recursivas
-----------------------------------------


% Consideramos la siguiente familia:

%                      Sam              Susan
%                            \        /
%                             \      /
%   Kathy       Mike            Rich       Patty
%         \      /                      |      /
%          \    /                       |     /
%             Mary                       Bob
%                 \                     /
%                  \                   /
%                   \                 /
%                        Bill

% Modelizamos la información de la familia:

persona(bill).
persona(mary).
persona(bob).
persona(kathy).
persona(mike).
persona(rich).
persona(patty).
persona(sam).
persona(susan).


padre(bob,bill).
padre(rich,bob).
padre(mike,mary).
padre(sam,rich).
madre(mary,bill).
madre(patty,bob).
madre(kathy,mary).
madre(susan,rich).

% La información sobre los padres es completa:

-padre(P,H) :- persona(P), persona(H),
               not padre(P,H).

-madre(M,H) :- persona(M), persona(H),
               not madre(M,H).
```

```
% Objetivo: modelizar la noción de antepasado.
Para ello, es necesario una definición recursiva.

progenitor(X,Y) :- padre(X,Y).
progenitor(X,Y) :- madre(X,Y).

antepasado(X,Y) :- progenitor(X,Y).
antepasado(X,Y) :- progenitor(Z,Y),
                   antepasado(X,Z).
-antepasado(X,Y) :- persona(X), persona(Y),
                    not antepasado(X,Y).
-------------------------------------------------------------
FIN DE LA CLASE
-------------------------------------------------------------
```