---
Teoría de Modelos. Clase del 6 de mayo de 2020
---
---
Tema 3: Razonamiento por defecto en ASP.
---
([GK], páginas 101-107)
---

Llamamos razonamiento "por defecto" a aquel razonamiento que, en lenguaje

natural, contiene palabras del tipo "Normalmente", "Por lo general", "En la mayor parte de los casos", etc.

Ejemplo clásico: "Los pájaros normalmente vuelan".

Las reglas de razonamiento por defecto admiten excepciones ("los pingüinos
son pájaros que no vuelan") y añadir una tal excepción a nuestra base de conocimiento no debe causar la aparición de ninguna inconsistencia.

El razonamiento por defecto está presente en nuestra manera habitual de razonar
cuando aplicamos el sentido común y su representación formal es una tema clásico
de la Inteligencia Artificial, en general, y del estudio del *razonamiento
no monótono*, en particular.

El paradigma de la programación con conjuntos de respuestas (ASP) nos proporciona
una marco muy adecuado para la formalización de este tipo de razonamiento no
monótono. Ilustremos las ideas principales con un par de ejemplos.

---
(A) Un ejemplo familiar
---

Consideremos el siguiente razonamiento:

1. Juan es el padre de Sam y Alicia es su madre.
2. Normalmente, los padres se ocupan de sus hijos.
   Por tanto, Juan se ocupa de Sam.

Primera representación: Formalizamos 2 como regla estricta (sin defecto):

---

```
persona(sam).
persona(juan).
persona(alicia).

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

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

cuida(X,Y) :- progenitor(X,Y).  %V1 (Regla estricta)
```

Es claro que con la representación actual podemos deducir que Juan se ocupa de Sam.
Sin embargo, supongamos que nos llega información nueva y ahora conocemos que:

3. Juan es una excepción a la regla: Juan no se ocupa de sus hijos.

La nueva información da lugar a la regla:

```
-cuida(juan,X) :- padre(juan,X).
```

Problema: Al añadir la nueva regla, el programa es INCONSISTENTE.
Debemos pues representar de otra manera la regla 2 si queremos capturar el carácter no monótono de dicha regla.

## Segunda representación: Defectos en ASP

---

En general, una regla de razonamiento por defecto de la forma:

"Normalmente los elementos de la clase C tienen la propiedad P"

se representará como sigue:

```
p(X) :- c(X), not ab(d(X)), not -p(X).
```

donde d es un símbolo para nombrar el defecto y el predicado ab(d(X)) representa "X es anormal para el defecto d" (esto es, el átomo ab(d(X)) será cierto si nos consta que el defecto d no es aplicable al objeto X).

Verbigracia, el ejemplo clásico "Los pájaros normalmente vuelan" se formalizaría mediante la regla ASP

```
vuela(X) :- pajaro(X), not ab(d_vuela(X)), not -vuela(X).
```

Idea: Si X es un pájaro y no consta en la base de conocimiento que X sea anormal para el defecto vuela/1 y tampoco consta -vuela(X), aplicamos el defecto y añadimos a la base de conocimiento vuela(X).

Las reglas por defecto en ASP admiten dos tipos de excepciones:

-) Excepciones débiles:
Si E(X) es una excepción débil para el defecto d, entonces haremos el defecto no aplicable para X y el programa no podrá aplicar la regla para extraer conclusiones sobre X.
-) Excepciones fuertes:
Si E(X) es una excepción fuerte para el defecto d, entonces no solo haremos el defecto no aplicable para X, sino que también permitimos deducir lo contrario que afirma el defecto para X.

Una excepción débil E(x) para un defecto d se representará mediante el llamado *Axioma de cancelación*

```
ab(d(X)) :- c(x), not -e(X).
```

Idea: Si X pudiera ser una excepción al defecto, impedimos aplicar el defecto (Razonador cauteloso, solo aplicamos el defecto si estamos seguros de que el objeto X no es una excepción).

Una excepción fuerte para d se representará mediante el axioma de cancelación más la regla siguiente:

```
ab(d(X)) :- c(X), not -e(X).
-p(X) :- e(X).
```

Volviendo al ejemplo clásico de los pájaros, si queremos formalizar la excepción fuerte "Los pingüinos no vuelan" añadimos

```
ab(d_vuela(X)) :- pajaro(X), not -pinguino(X).
-vuela(X) :- pinguino(X).
```

Si queremos formalizar la excepción débil "Algunas avestruces no vuelan pero hay otras que sí", basta con añadir el axioma de cancelación:

```
ab(d_vuela(X)) :- pajaro(X), not -avestruz(X).
```

Retomemos ahora el ejemplo familiar:

```
persona(sam).
persona(juan).
persona(alicia).
```

```
padre(juan,sam).
madre(alicia,sam).

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

"Normalmente, los padres se ocupan de sus hijos" se representará ahora mediante la regla:

```
cuida(X,Y) :- progenitor(X,Y),
              not ab(d_cuida(X,Y)),
              not -cuida(X,Y).
```

La excepción "Juan no se ocupa de sus hijos" se representará como una excepción fuerte del defecto d_cuida.

```
ab(d_cuida(juan,X)) :- persona(X),not -padre(juan,X).
-cuida(juan,X) :- padre(juan,X).
```

Ahora el programa sí es consistente y es capaz de deducir que, por una parte, Juan no cuida de Sam y, por otra, Alicia sí cuida de Sam.

El lector atento habrá observado que en el ejemplo anterior el axioma de cancelación para la excepción fuerte "Juan no cuida de sus hijos" no es esencial. Ahora bien, hay otras situaciones donde añadir el axioma de cancelación a las excepciones fuertes sí es importante. Veamos un ejemplo de ello.

Supongamos que nos llegan noticias de la existencia de un país de nombre Distopia donde los progenitores no cuidan de sus hijos. Debemos pues añadir una nueva excepción fuerte para el defecto d_cuida.

```
ab(d_cuida(X,Y)) :- progenitor(X,Y),not -nacido_en(X,distopia).
-cuida(X,Y) :- progenitor(X,Y),nacido_en(X,distopia).
```

Ampliemos la base familiar con una nueva familia. Julia y Luis son los progenitores de Ana y Teresa.

```
persona(teresa).
persona(ana).
persona(luis).
persona(julia).
```

```
padre(luis,ana).
padre(luis,teresa).
madre(julia,ana).
madre(julia,teresa).

Tenemos información sobre la nacionalidad de algunos.

pais(distopia).
pais(francia).
pais(italia).

nacido_en(alicia,francia).
nacido_en(sam,italia).
nacido_en(luis,distopia).

Más la regla de conocimiento implícito

-nacido_en(X,P) :- nacido_en(X,W),P!=W,pais(P).

Si ejecutamos el programa obtenemos

% clingo version 5.4.0
% Reading from clase6mayo.lp
% Solving...
% Answer: 1
% cuida(alicia,sam) -cuida(luis,ana) -cuida(luis,teresa) -cuida(juan,sam)
% SATISFIABLE

La respuesta es la esperada. Deducimos que Juan no cuida a Sam por
la primera excepción y que Luis no cuida ni a Ana ni a Teresa por
la segunda excepción (Luis nació en Distopia). Deducimos que Alicia
sí cuida de Sam (Alicia nació en Francia y, por tanto, no en Distopia).
Mientras que si Julia cuida de sus hijas o no es desconocido:
no podemos descartar que Julia sea de Distopia
(no conocemos su nacionalidad) y, por tanto, el axioma de cancelación
para
la segunda excepción añade ab(d_cuida(julia,ana)) y ab(d_cuida(teresa))
y hace no aplicable la regla de razonamiento por defecto.

Si no añadimos el axioma de cancelación para la segunda excepción,
el conjunto de respuestas varía y, de manera errónea si pretendemos
representar un razonador cauteloso, el programa deduciría que Julia
sí cuida de sus hijas, aun sin tener la certeza de que Julia
sea de un país distinto a Distopia. En este tipo de situaciones,
es importante añadir el axioma de cancelación para las excepciones
fuertes.

-------------------------
```

(B) Estudiantes temerosos
---

Consideramos ahora un segundo ejemplo, que contiene tanto excepciones fuertes como débiles.

1. Por regla general, los estudiantes temen a las Mates.
2. Bea no teme a las Mates.
3. Los estudiantes del Dpto CCIA no temen a las Mates.
4. Algunos estudiantes de Bilogía temen a las Mates y otros no.

Regla de razonamiento por defecto:

```
teme(X,mates) :- estudiante(X),
                 not ab(d_mates(X)),
                 not -teme(X,mates).
```

Excepción fuerte: "Bea no teme a las Mates"

```
-teme(bea,mates).
ab(d_mates(bea)).
```

Excepción fuerte: "Los estudiantes del Dpto CCIA no temen a las Mates"

```
-teme(X,mates) :- pertenece(X,ccia).
ab(d_mates(X)) :- estudiante(X), not -pertenece(X,ccia).
```

Excepción débil: "Algunos estudiantes de Bilogía temen a las Mates y otros no"

```
ab(d_mates(X)) :- estudiante(X),not -pertenece(X,bio).
```

Añadimos info sobre algunos estudiantes.

```
estudiante(bea).
estudiante(renata).
estudiante(dario).
estudiante(ursula).

dpto(ccia).
dpto(bio).
dpto(filosofia).

pertenece(bea,filosofia).
pertenece(renata,ccia).
pertenece(dario,bio).
pertenece(ursula,filosofia).
```

```
-pertenece(X,D) :- pertenece(X,D1),
                   dpto(D),D1!=D.
```

Observemos que las respustas son las esperadas para un razonador cauteloso.

```
% clingo version 5.4.0
% Reading from clase6mayo.lp
% Solving...
% Answer: 1
% teme(ursula,mates) -teme(bea,mates) -teme(renata,mates)
% SATISFIABLE

----------------------------------------
FIN DE LA CLASE
----------------------------------------
```