

Razonamiento con información incompleta

José A. Alonso

Dpto. de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

Razonamiento por defecto

- **Ejemplo de razonamiento por defecto**

El animal_1 es un pájaro
Normalmente, los pájaros vuelan.
Por tanto, el animal_1 vuela.

- **Programa P1**

- **Programa P1**

```
pajaro(animal_1).  
vuela(X) :-  
    pajaro(X),  
    normal(X).
```

- **Modelos del programa P1**

```
{pajaro(animal_1)}  
{pajaro(animal_1), vuela(animal_1)}  
{pajaro(animal_1), normal(animal_1), vuela(animal_1)}
```

- **Consecuencia**

```
P1 |=/= vuela(animal_1)
```

Razonamiento por defecto

- Programa P2 con anormal/1

- Programa P2

```
:- dynamic anormal/1.
```

```
pajaro(animal_1).
```

```
vuela(X) :-
```

```
    pajaro(X),
```

```
    not anormal(X).
```

- Sesión

```
?- vuela(animal_1).
```

```
Yes
```

- Traza

```
?- vuela(animal_1).
```

```
Call: ( 7) vuela(animal_1) ?
```

```
Call: ( 8) pajaro(animal_1) ?
```

```
Exit: ( 8) pajaro(animal_1) ?
```

```
^ Call: ( 8) not anormal(animal_1) ?
```

```
Call: ( 9) anormal(animal_1) ?
```

```
Fail: ( 9) anormal(animal_1) ?
```

```
^ Exit: ( 8) not anormal(animal_1) ?
```

```
Exit: ( 7) vuela(animal_1) ?
```

```
Yes
```

Razonamiento por defecto

- **Extensión del conocimiento**

- **Nuevo conocimiento**

- El animal_1 es un avestruz.

- Los avestruces son pájaros que no vuelan.

- **Programa extendido**

- ```
:- dynamic anormal/1.
```

- ```
pajaro(animal_1).
avestruz(animal_1).
vuela(X) :-
    pajaro(X),
    not anormal(X).
anormal(X) :- avestruz(X).
```

- **Traza**

- ```
?- vuela(animal_1).
Call: (7) vuela(animal_1) ?
Call: (8) pajaro(animal_1) ?
Exit: (8) pajaro(animal_1) ?
^ Call: (8) not anormal(animal_1) ?
Call: (9) anormal(animal_1) ?
Call: (10) avestruz(animal_1) ?
Exit: (10) avestruz(animal_1) ?
Exit: (9) anormal(animal_1) ?
^ Fail: (8) not anormal(animal_1) ?
Fail: (7) vuela(animal_1) ?
```

No

# Razonamiento por defecto

- Cancelación reglas por defectos mediante reglas específicas
  - Regla por defecto: “Normalmente, los pájaros vuelan”
  - Regla específica: “Los avestruces no vuelan”
- Razonamiento monótono y no-monótono
  - Razonamiento monótono  
 $P1 \models C$  y  $P2$  extiende a  $P1$ , entonces  $P2 \models C$ .
  - Razonamiento no-monótono  
 $P1 \models C$  y  $P2$  extiende a  $P1$ , es posible  $P2 \not\models C$ .

# Razonamiento por defecto

- Programa con reglas y reglas con excepciones (defectos)

- Programa objeto

```
defecto((vuela(X) :- pajaro(X))).
```

```
regla((pajaro(X) :- avestruz(X))).
```

```
regla((not vuela(X) :- avestruz(X))).
```

```
regla((avestruz(animal_1) :- true)).
```

```
regla((pajaro(animal_2) :- true)).
```

- Sesión

```
?- explica(vuela(X),E).
```

```
X = animal_2
```

```
E = [defecto((vuela(animal_2) :- pajaro(animal_2))),
 regla((pajaro(animal_2) :- true))] ;
```

```
No
```

```
?- explica(not vuela(X),E).
```

```
X = animal_1
```

```
E = [regla((not vuela(animal_1) :- avestruz(animal_1))),
 regla((avestruz(animal_1) :- true))] ;
```

```
No
```

```
?- explica(vuela(animal_2),_).
```

```
Yes
```

```
?- explica(vuela(animal_1),_).
```

```
No
```

# Razonamiento por defecto

- Metaprograma para explicaciones

```
explica(A,E) :-
 explica(A, [],E).

explica(true,E,E) :- !.
explica((A,B),E0,E) :- !,
 explica(A,E0,E1),
 explica(B,E1,E).
explica(A,E0,E) :-
 prueba(A,E0,E).
explica(A,E0,[defecto((A:-B))|E]) :-
 defecto((A:-B)),
 explica(B,E0,E),
 not contradiccion(A,E).

prueba(true,E,E) :- !.
prueba((A,B),E0,E) :- !,
 prueba(A,E0,E1),
 prueba(B,E1,E).
prueba(A,E0,[regla((A:-B))|E]) :-
 regla((A:-B)),
 prueba(B,E0,E).

contradiccion(not A,E) :- !,
 prueba(A,E,_E1).
contradiccion(A,E) :-
 prueba(not A,E,_E1).
```

# Razonamiento por defecto

- Explicaciones de hechos contradictorios

- Programa objeto

```
defecto((not vuela(X) :- mamifero(X))).
defecto((vuela(X) :- vampiro(X))).
defecto((not vuela(X) :- muerto(X))).

regla((mamifero(X) :- vampiro(X))).
regla((vampiro(dracula) :- true)).
regla((muerto(dracula) :- true)).
```

- Sesión

```
:- [p5].
```

```
?- explica(vuela(dracula),E).
E = [defecto((vuela(dracula) :- vampiro(dracula))),
 regla((vampiro(dracula) :- true))] ;
No
```

```
?- explica(not vuela(dracula),E).
E = [defecto((not vuela(dracula) :- mamifero(dracula))),
 regla((mamifero(dracula) :- vampiro(dracula))),
 regla((vampiro(dracula) :- true))] ;
E = [defecto((not vuela(dracula) :- muerto(dracula))),
 regla((muerto(dracula) :- true))] ;
No
```



# Razonamiento por defecto

- Cancelación entre defectos mediante nombres

- Programa objeto

```
defecto(mamiferos_no_vuelan(X),
 (not vuela(X) :- mamifero(X))).
```

```
defecto(vampiros_vuelan(X),
 (vuela(X) :- vampiro(X))).
```

```
defecto(muertos_no_vuelan(X),
 (not vuela(X) :- muerto(X))).
```

```
regla((mamifero(X) :- vampiro(X))).
```

```
regla((vampiro(dracula) :- true)).
```

```
regla((muerto(dracula) :- true)).
```

```
regla((not mamiferos_no_vuelan(X) :- vampiro(X))).
```

```
regla((not vampiros_vuelan(X) :- muerto(X))).
```

- Modificación de explica

```
explica(A,E0,[defecto(Nombre)|E]) :-
 defecto(Nombre,(A:-B)),
 explica(B,E0,E),
 not contradiccion(Nombre,E),
 not contradiccion(A,E).
```

- Sesión

```
?- explica(vuela(dracula),E).
```

No

```
?- explica(not vuela(dracula),E).
```

```
E = [defecto(muertos_no_vuelan(dracula)),
 regla((muerto(dracula) :- true))]
```

Yes

# Razonamiento abductivo

- **Problema de la abducción**

Dados        P un programa lógico y  
              O una observación (un hecho básico  
                  en el lenguaje de P)

Encontrar E una explicación (una lista de hechos atómicos  
en el lenguaje de P cuyos predicados no son  
cabezas de cláusulas de P) tal que  
 $P \cup E \vdash O$ )

- **Abducción para programas definidos**

- **Programa objeto**

```
 europeo(X) <- español(X).
 español(X) <- andaluz(X).
 europeo(X) <- italiano(X).
```

- **Sesión**

```
?- abduce(europeo(juan),E).
E = [andaluz(juan)] ;
E = [italiano(juan)] ;
No
```

# Razonamiento abductivo

- Programa

```
:- op(1200,xfx,<-).
```

```
abduce(0,E) :-
 abduce(0,[],E).
```

```
abduce(true,E,E) :- !.
abduce((A,B),E0,E) :- !,
 abduce(A,E0,E1),
 abduce(B,E1,E).
```

```
abduce(A,E0,E) :-
 (A <- B),
 abduce(B,E0,E).
```

```
abduce(A,E,E) :-
 member(A,E).
```

```
abduce(A,E,[A|E]) :-
 not member(A,E),
 abducible(A).
```

```
abducible(A) :-
 not (A <- _B).
```

# Razonamiento abductivo

- Problemas al aplicar abducción a programas indefinidos

- Programa objeto

```
vuela(X) <- pajaro(X), not anormal(X).
anormal(X) <- avestruz(X).
pajaro(X) <- avestruz(X).
pajaro(X) <- palomo(X).
```

- Sesión

```
?- abduce(vuela(animal_1),E).
E = [not anormal(animal_1), avestruz(animal_1)] ;
E = [not anormal(animal_1), palomo(animal_1)] ;
No
```

- Problemas:

- \* Explicación contradictoria
- \* Explicación con predicado no abducible

# Razonamiento abductivo

- Abducción para programas generales

- Metaprograma abductivo

```
:- op(1200,xfx,<-).
```

```
abduce(0,E) :-
 abduce(0,[],E).
```

```
abduce(true,E,E) :- !.
abduce((A,B),E0,E) :- !,
 abduce(A,E0,E1),
 abduce(B,E1,E).
```

```
abduce(A,E0,E) :-
 (A <- B),
 abduce(B,E0,E).
```

```
abduce(A,E,E) :-
 member(A,E).
```

```
abduce(A,E,[A|E]) :-
 not member(A,E),
 abducible(A),
 not abduce_not(A,E,E).
```

```
abduce(not(A),E0,E) :-
 not member(A,E0),
 abduce_not(A,E0,E).
```

# Razonamiento abductivo

```
abduce_not((A,B),E0,E) :- !,
 abduce_not(A,E0,E);
 abduce_not(B,E0,E).
abduce_not(A,E0,E) :-
 setof(B,(A <- B),L),
 abduce_not_1(L,E0,E).
abduce_not(A,E,E) :-
 member(not(A),E).
abduce_not(A,E,[not(A)|E]) :-
 not member(not(A),E),
 abducible(A),
 not abduce(A,E,E).
abduce_not(not(A),E0,E) :-
 not member(not(A),E0),
 abduce(A,E0,E).

abduce_not_1([],E,E).
abduce_not_1([B|R],E0,E) :-
 abduce_not(B,E0,E1),
 abduce_not_1(R,E1,E).

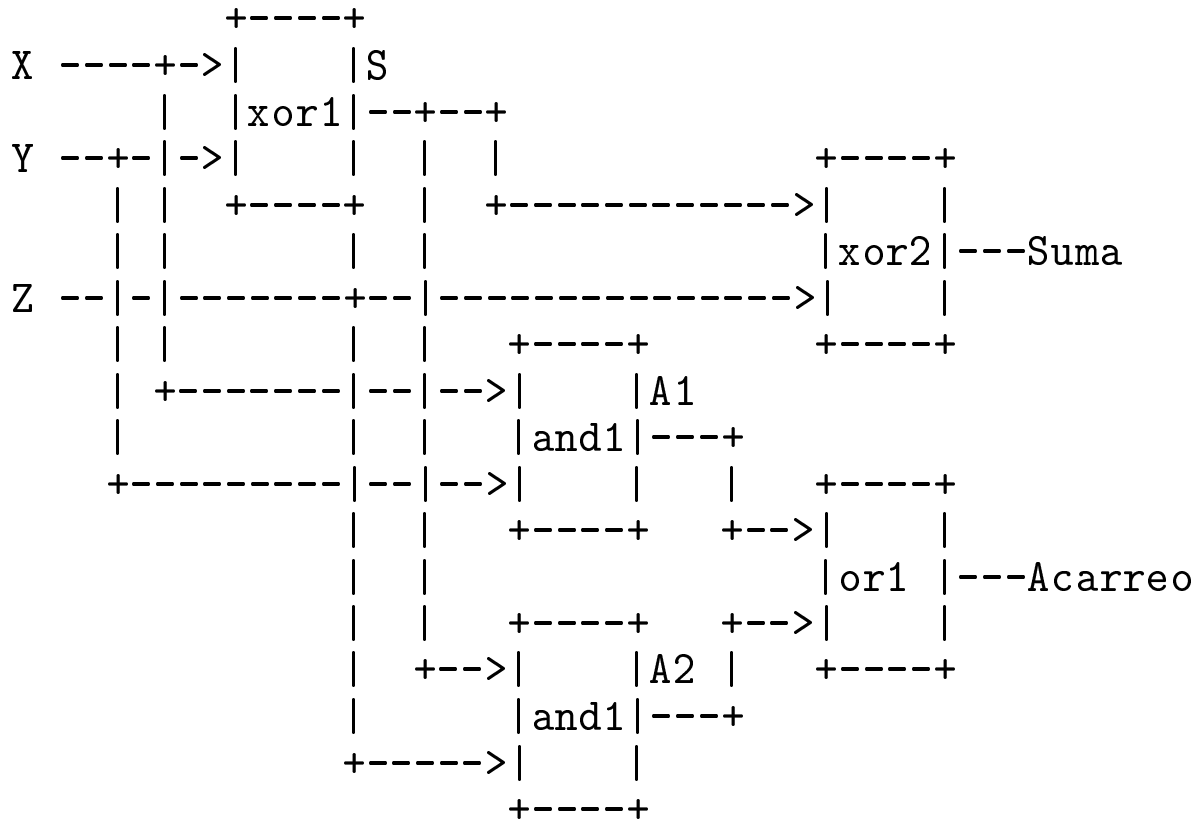
abducible(A) :-
 A \= not(_X),
 not (A <- _B).
```

- Sesión con el programa objeto anterior

```
?- abduce(vuela(animal_1),E).
E = [not avestruz(animal_1), palomo(animal_1)]
Yes
```

# Diagnóstico mediante abducción

- Representación de un sumador



# Diagnóstico mediante abducción

## ● Definición del sumador

```
sumador(X,Y,Z,Acarreo,Suma) :-
 xor(X,Y,S) ,
 xor(Z,S,Suma) ,
 and(X,Y,A1) ,
 and(Z,S,A2) ,
 or(A1,A2,Acarreo) .
```

```
and(1,1,1). and(1,0,0).
and(0,1,0). and(0,0,0).
```

```
or(1,1,1). or(1,0,1).
or(0,1,1). or(0,0,0).
```

```
xor(1,0,1). xor(0,1,1).
xor(1,1,0). xor(0,0,0).
```

```
tabla :-
 format('X Y Z A S~n'),
 tabla2.
```

```
tabla2 :-
 member(X,[0,1]),
 member(Y,[0,1]),
 member(Z,[0,1]),
 sumador(X,Y,Z,A,S),
 format('~a ~a ~a ~a ~a~n',[X,Y,Z,A,S]),
 fail.
```

```
tabla2.
```



# Diagnóstico mediante abducción

- Sesión con el sumador

```
?- tabla.
X Y Z A S
0 0 0 0 0
0 0 1 0 1
0 1 0 0 1
0 1 1 1 0
1 0 0 0 1
1 0 1 1 0
1 1 0 1 0
1 1 1 1 1
Yes
```

- Modelo de fallo del sumador

```
sumador(X,Y,Z,Acarreo,Suma) <-
 xorg(xor1,X,Y,S),
 xorg(xor2,Z,S,Suma),
 andg(and1,X,Y,A1),
 andg(and2,Z,S,A2),
 org(or1,A1,A2,Acarreo).

xorg(_N,X,Y,Z) <- xor(X,Y,Z).
xorg(N,1,1,1) <- fallo(N=f1).
xorg(N,0,0,1) <- fallo(N=f1).
xorg(N,1,0,0) <- fallo(N=f0).
xorg(N,0,1,0) <- fallo(N=f0).
```

# Diagnóstico mediante abducción

```
andg(_N,X,Y,Z) <- and(X,Y,Z).
andg(N,0,0,1) <- fallo(N=f1).
andg(N,1,0,1) <- fallo(N=f1).
andg(N,0,1,1) <- fallo(N=f1).
andg(N,1,1,0) <- fallo(N=f0).
```

```
org(_N,X,Y,Z) <- or(X,Y,Z).
org(N,0,0,1) <- fallo(N=f1).
org(N,1,0,0) <- fallo(N=f0).
org(N,0,1,0) <- fallo(N=f0).
org(N,1,1,0) <- fallo(N=f0).
```

## ● Diagnóstico mediante abducción

```
diagnostico(Observacion,Diagnostico) :-
 abduce(Observacion,Diagnostico).
```

```
:- abolish(abducible,2).
abducible(fallo(_X)).
```

## ● Sesión de diagnóstico

```
?- diagnostico(sumador(0,0,1,1,0),D).
D = [fallo(or1 = f1), fallo(xor2 = f0)] ;
D = [fallo(and2 = f1), fallo(xor2 = f0)] ;
D = [fallo(and1 = f1), fallo(xor2 = f0)] ;
D = [fallo(and2 = f1), fallo(and1 = f1), fallo(xor2 = f0)]
D = [fallo(xor1 = f1)] ;
D = [fallo(or1 = f1), fallo(and2 = f0), fallo(xor1 = f1)]
D = [fallo(and1 = f1), fallo(xor1 = f1)] ;
D = [fallo(and2 = f0), fallo(and1 = f1), fallo(xor1 = f1)]
No
```

# Diagnóstico mediante abducción

- Diagnóstico mínimo

```
diagnostico_minimo(O,D) :-
 diagnostico(O,D),
 not((diagnostico(O,D1),
 subconjunto_propio(D1,D))).
```

```
subconjunto_propio([],Ys) :-
 Ys \= [].
```

```
subconjunto_propio([X|Xs],Ys) :-
 select(Ys,X,Ys1),
 subconjunto_propio(Xs,Ys1).
```

- Diagnóstico mínimo del sumador

```
?- diagnostico_minimo(sumador(0,0,1,1,0),D).
D = [fallo(or1 = f1), fallo(xor2 = f0)] ;
D = [fallo(and2 = f1), fallo(xor2 = f0)] ;
D = [fallo(and1 = f1), fallo(xor2 = f0)] ;
D = [fallo(xor1 = f1)] ;
No
```

# Defectos mediante abducción

- Traducción del programa objeto

```
no_vuela(X) <- mamifero(X), not mamifero_volador(X).
vuela(X) <- vampiro(X), not vampiro_no_volador(X).
no_vuela(X) <- muerto(X), not muerto_volador(X).
```

```
mamifero(X) <- vampiro(X).
vampiro(dracula) <- true.
muerto(dracula) <- true.
```

```
mamifero_volador(X) <- vampiro(X).
vampiro_no_volador(X) <- muerto(X).
```

```
:- abolish(abducible,1).
abducible(mamifero_volador(_)).
abducible(vampiro_no_volador(_)).
abducible(muerto_volador(_)).
```

- Sesión

```
?- abduce(vuela(X),E).
No
```

```
?- abduce(no_vuela(X),E).
X = dracula
E = [not muerto_volador(dracula)] ;
No
```

## Bibliografía

- Flach, P. “Simply Logical (Intelligent Reasoning by Example)” (John Wiley, 1994)
  - Cap. 8: “Reasoning incomplete information”
- Rich, E. y Knight, K. “Inteligencia artificial (segunda edición)” (McGraw–Hill Interamericana, 1994).
  - Cap. 7: “Razonamiento simbólico bajo incertidumbre”