

Tema 4: Retroceso, corte y negación

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

Control del retroceso

- Procedimiento sin corte: $f(X,Y)$

- Definir la relación $f(X,Y)$ de forma que:

si $X < 3$, entonces $Y = 0$;
si $3 \leq X < 6$, entonces $Y = 2$;
si $6 \leq X$, entonces $Y = 4$.

- Programa:

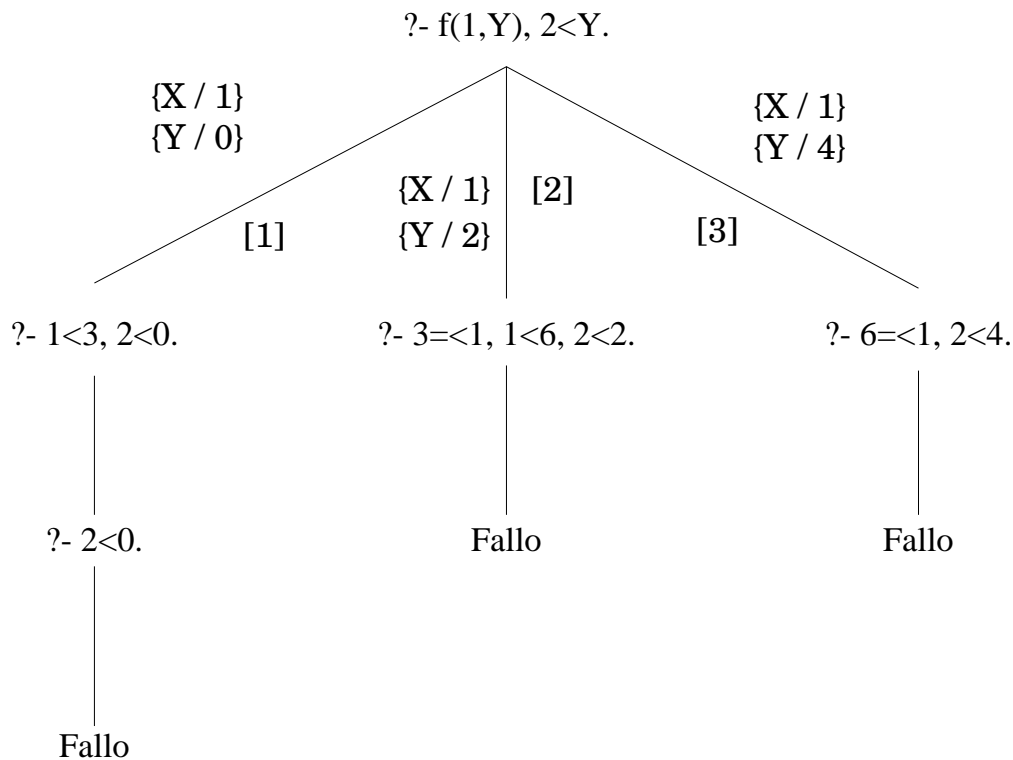
```
f(X,0) :- X < 3.  
f(X,2) :- 3 =< X, X < 6.  
f(X,4) :- 6 =< X.
```

- Traza de $?- f(1,Y), 2 < Y$.

```
?- f(1,Y), 2 < Y.  
T Call: ( 8) f(1, _G266)  
T Call: ( 9) 1 < 3  
T Exit: ( 9) 1 < 3  
T Exit: ( 8) f(1, 0)  
T Call: ( 8) 2 < 0  
T Fail: ( 8) 2 < 0  
T Redo: ( 8) f(1, _G266)  
T Redo: ( 8) f(1, _G266)  
T Fail: ( 8) f(1, _G266)  
No
```

Control del retroceso

- Arbol de resolución SLD



- Procedimiento con corte: $f_1(X,Y)$

- Definir la relación $f_1(X,Y)$ a partir de la definición de $f(X,Y)$, introduciendo un corte al final de las dos primeras cláusulas

- Programa

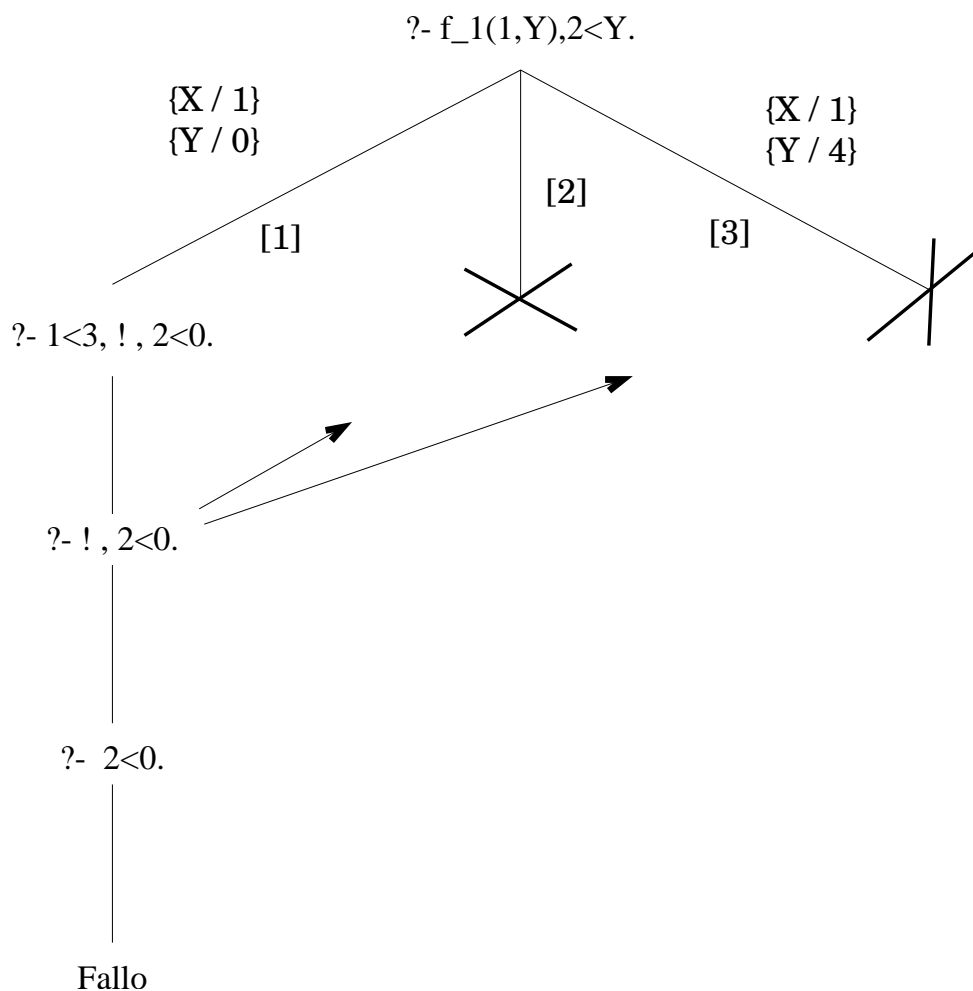
```
f_1(X,0) :- X < 3, !.
f_1(X,2) :- 3 =< X, X < 6, !.
f_1(X,4) :- 6 =< X.
```

Control del retroceso

- Traza de $?- f_1(1,Y), 2 < Y.$

```

?- f_1(1,Y), 2 < Y.
T Call: ( 8) f_1(1, _G278)
T Call: ( 9) 1 < 3
T Exit: ( 9) 1 < 3
T Exit: ( 8) f_1(1, 0)
T Call: ( 8) 2 < 0
T Fail: ( 8) 2 < 0
No
  
```



Control del retroceso

- Comentarios sobre el corte en f_1 :
 - ha mejorado la eficiencia
 - ha modificado la semántica procedimental
 - no ha modificado la semántica declarativa
- Efectos del corte:
 - Fija todas las sustituciones de variables se hayan realizado a partir de la cláusula donde ha aparecido el corte.
 - No intenta encontrar soluciones alternativas a los literales a la izquierda del corte
 - No intenta cláusulas alternativas a la del corte

Control del retroceso

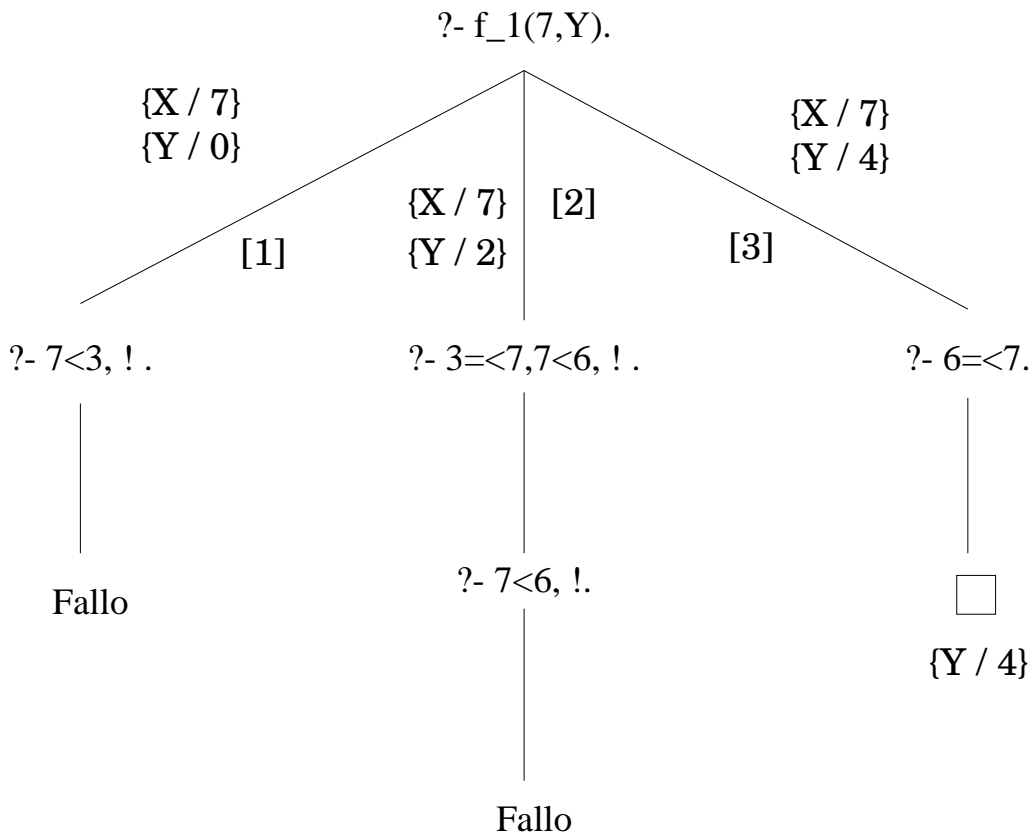
- Traza de $?- f_1(7, Y)$.

```

?- f_1(7, Y).
Call: ( 7) f_1(7, _G118) ?
Call: ( 8) 7<3 ?
Fail: ( 8) 7<3 ?
Redo: ( 7) f_1(7, _G118) ?
Call: ( 8) 3=<7 ?
Exit: ( 8) 3=<7 ?
Call: ( 8) 7<6 ?
Fail: ( 8) 7<6 ?
Redo: ( 7) f_1(7, _G118) ?
Call: ( 8) 6=<7 ?
Exit: ( 8) 6=<7 ?
Exit: ( 7) f_1(7, 4) ?

```

$Y = 4$



Control del retroceso

- Procedimiento con corte $f_2(X,Y)$
 - Definir la relación $f_2(X,Y)$ suprimiendo en la definición de $f_1(X,Y)$ las comparaciones innecesarias.
 - Programa
- ```
f_2(X,0) :- X < 3, !.
f_2(X,2) :- X < 6, !.
f_2(X,4).
```

# Control del retroceso

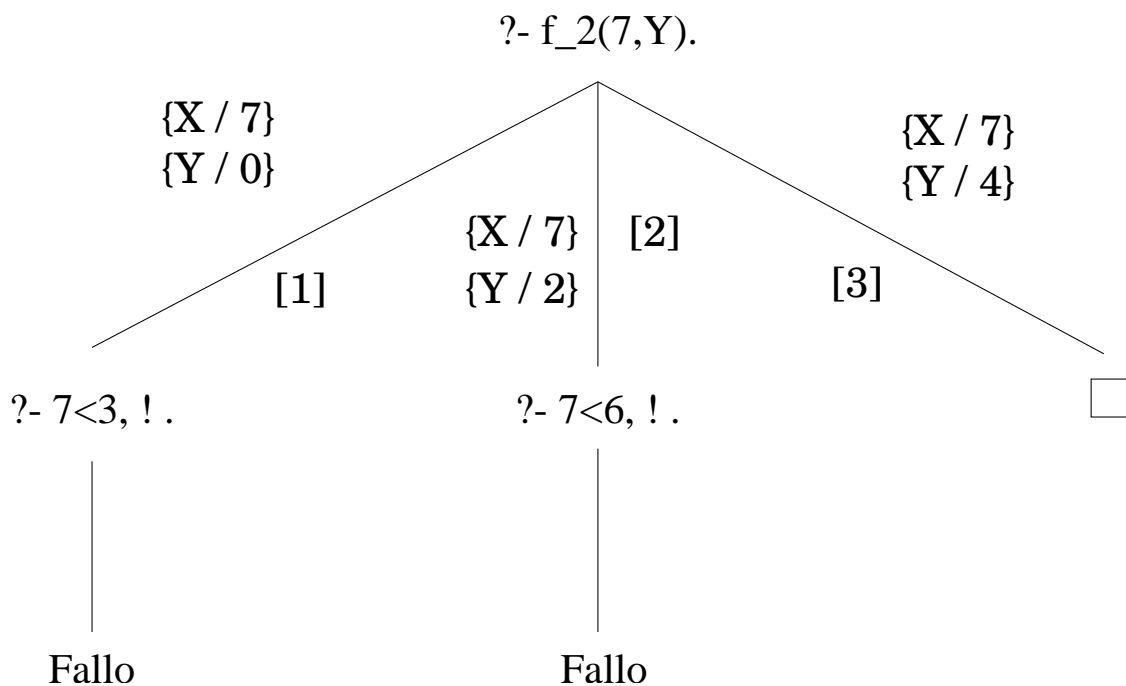
- Traza de `?- f_2(7,Y)`.

```

?- f_2(7,Y).
T Call: (8) f_2(7, _G236)
T Call: (9) 7 < 3
T Fail: (9) 7 < 3
T Redo: (8) f_2(7, _G236)
T Call: (9) 7 < 6
T Fail: (9) 7 < 6
T Redo: (8) f_2(7, _G236)
T Exit: (8) f_2(7, 4)
Y = 4 ;
No

```

- Arbol de resolución SLD



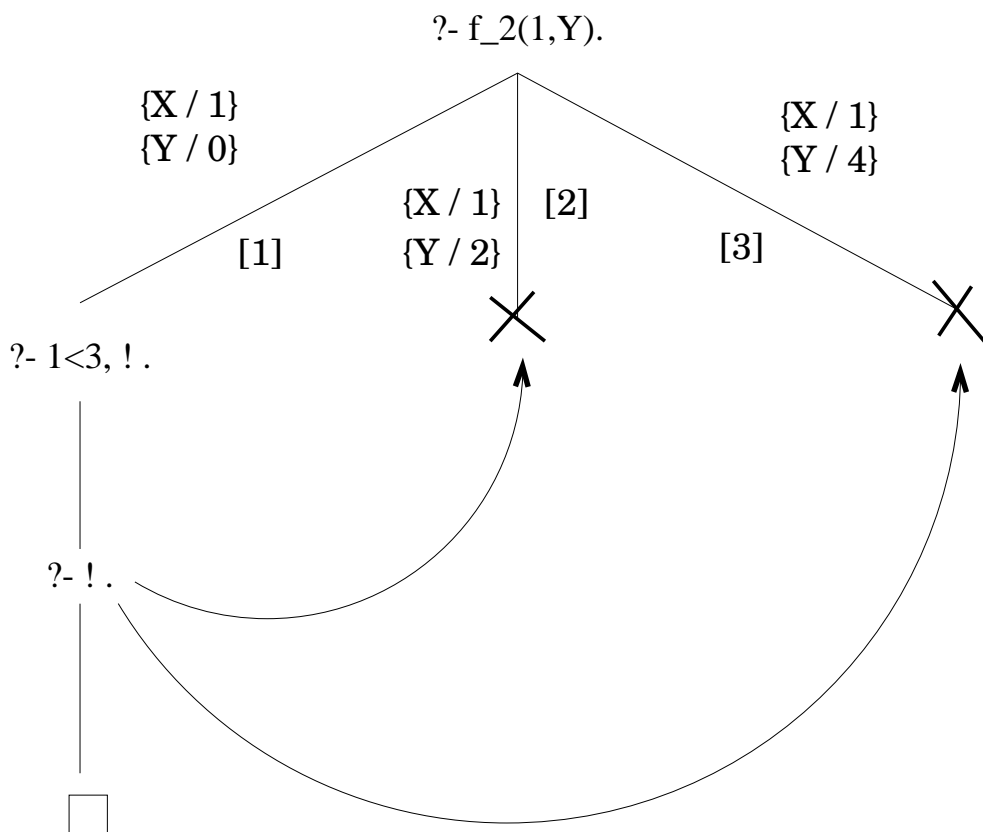


# Control del retroceso

- Traza de `?- f_2(1,Y)`.

```
?- f_2(1,Y).
 Call: (7) f_2(1, _G118) ?
 Call: (8) 1<3 ?
 Exit: (8) 1<3 ?
 Exit: (7) f_2(1, 0) ?
Y = 0 ;
No
```

- Arbol de resolución SLD



# Control del retroceso

- Procedimiento con eliminación de cortes
  - Definir la relación  $f\_3(X,Y)$  a partir de la definición de  $f\_2(X,Y)$ , suprimiendo los cortes

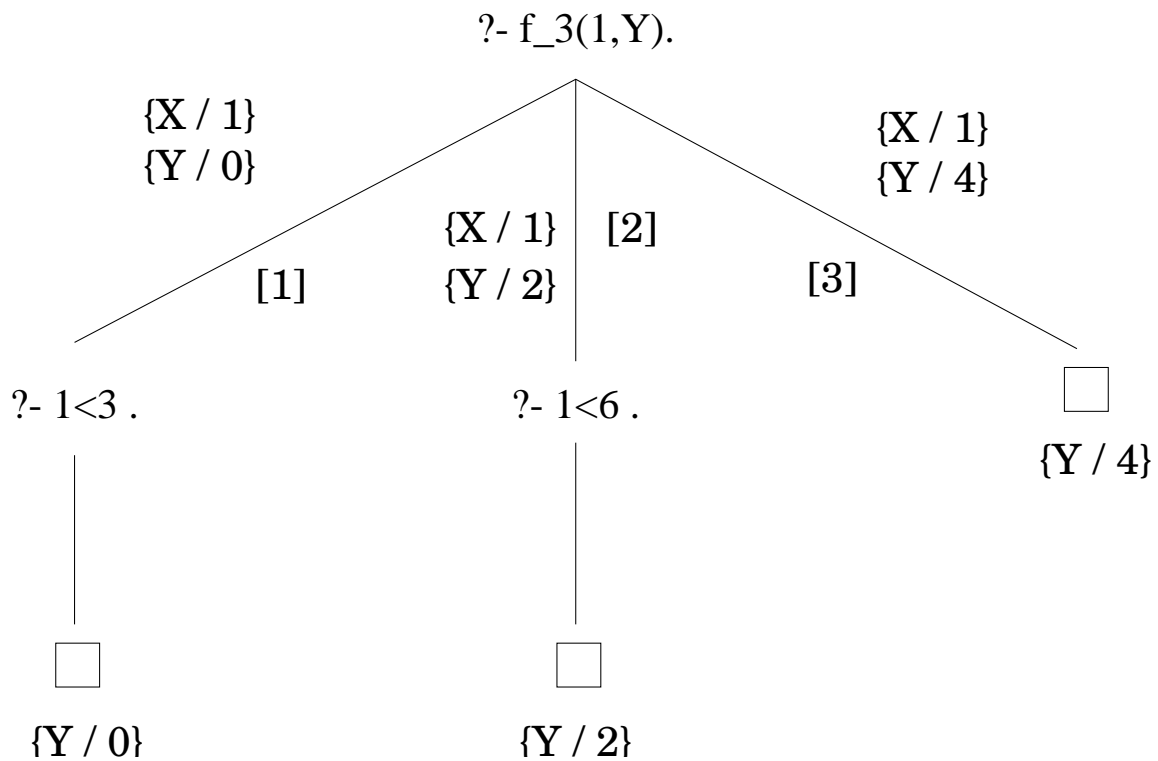
- Programa

$f\_3(X,0) :- X < 3.$   
 $f\_3(X,2) :- X < 6.$   
 $f\_3(X,4).$

- Respuestas a  $f\_3(1,Y)$

$?- f\_3(1,Y).$   
 $Y = 0 ; Y = 2 ; Y = 4 ;$   
 No

- Arbol de resolución SLD



# Ejemplos con corte

- Cálculo del máximo

- Programa sin corte

```
maximo_1(X,Y,X) :- Y =< X.
maximo_1(X,Y,Y) :- X =< Y.
```

- Programa con corte

```
maximo_2(X,Y,X) :- Y =< X, !.
maximo_2(X,Y,Y).
```

- Sesión

```
?- maximo_1(3,5,X).
X = 5 ;
No
?- maximo_2(3,5,X).
X = 5 ;
No
?- maximo_1(3,2,2).
No
?- maximo_2(3,2,2).
Yes
```

- Comentario: eficiencia vs. semántica

# Ejemplos con corte

- Test de pertenencia

- Programa sin corte

```
pertenece_1(X, [X|_]).
pertenece_1(X, [_|L]) :- pertenece_1(X,L).
```

- Programa con corte

```
pertenece_2(X, [X|_]) :- !.
pertenece_2(X, [_|L]) :- pertenece_2(X,L).
```

- Sesión

```
?- pertenece_1(a, [a,b,a]).
Yes
?- pertenece_1(c, [a,b,a]).
No
?- pertenece_1(X, [a,b,a]).
X = a;
X = b;
X = a;
No
?- pertenece_2(a, [a,b,a]).
Yes
?- pertenece_2(c, [a,b,a]).
No
?- pertenece_2(X, [a,b,a]).
X = a; No
```

- Predicados member y memberchk

# Ejemplos con corte

- **Agregación sin repeticiones**

- agregar( $X, L, L1$ ) se verifica si  $X$  es un elemento de  $L$  y  $L1$  es  $L$ ; o, en caso contrario,  $L1$  es la lista obtenida añadiéndole  $X$  a  $L$

- **Ejemplos**

?- agregar( $a, [b, c], L$ ).  
 $L = [a, b, c]$

?- agregar( $b, [b, c], L$ ).  
 $L = [b, c]$

- **Programa**

```
agregar(X, L, L) :-
 memberchk(X, L), !.
agregar($X, L, [X|L]$).
```

# Ejemplos con corte

- Clasificación en categorías

- Programa sin corte

```
nota_1(X,suspenso) :- X < 5.
nota_1(X,aprobado) :- X >= 5, X < 7.
nota_1(X,notable) :- X >= 7, X < 9.
nota_1(X,sobresaliente) :- X >= 9.
```

- Traza

```
?- nota_1(7.3,Y).
 Call: (7) nota_1(7.3, _G114) ?
 Call: (8) 7.3<5 ?
 Fail: (8) 7.3<5 ?
 Redo: (7) nota_1(7.3, _G114) ?
 Call: (8) 7.3>=5 ?
 Exit: (8) 7.3>=5 ?
 Call: (8) 7.3<7 ?
 Fail: (8) 7.3<7 ?
 Redo: (7) nota_1(7.3, _G114) ?
 Call: (8) 7.3>=7 ?
 Exit: (8) 7.3>=7 ?
 Call: (8) 7.3<9 ?
 Exit: (8) 7.3<9 ?
 Exit: (7) nota_1(7.3, notable) ?
Y = notable ;
 Redo: (7) nota_1(7.3, _G114) ?
 Call: (8) 7.3>=9 ?
 Fail: (8) 7.3>=9 ?
 Fail: (7) nota_1(7.3, _G114) ?
No
```

# Ejemplos con corte

- Programa con corte

```
nota_2(X,suspense) :- X < 5, !.
nota_2(X,aprobado) :- X < 7, !.
nota_2(X,notable) :- X < 9, !.
nota_2(X,sobresaliente).
```

- Traza

```
?- nota_2(7.3,Y).
 Call: (7) nota_2(7.3, _G126) ?
 Call: (8) 7.3<5 ?
 Fail: (8) 7.3<5 ?
 Redo: (7) nota_2(7.3, _G126) ?
 Call: (8) 7.3<7 ?
 Fail: (8) 7.3<7 ?
 Redo: (7) nota_2(7.3, _G126) ?
 Call: (8) 7.3<9 ?
 Exit: (8) 7.3<9 ?
 Exit: (7) nota_2(7.3, notable) ?
Y = notable ;
No

?- nota(6,sobresaliente).
Yes
```

# Negación como fallo

- Introducción de la negación como fallo

- Programa 1

q1(a) :- q1(b), !, q1(c).

q1(a) :- q1(d).

q1(d).

- Traza

?- q1(a).

Call: ( 7) q1(a) ?

Call: ( 8) q1(b) ?

Fail: ( 8) q1(b) ?

Redo: ( 7) q1(a) ?

Call: ( 8) q1(d) ?

Exit: ( 8) q1(d) ?

Exit: ( 7) q1(a) ?

Yes



# Negación como fallo

- Programa 2

```
q2(a) :- q2(b), !, q2(c).
q2(a) :- q2(d).
q2(d).
q2(b).
```

- Traza

```
?- q2(a).
 Call: (7) q2(a) ?
 Call: (8) q2(b) ?
 Exit: (8) q2(b) ?
 Call: (8) q2(c) ?
 Fail: (8) q2(c) ?
 Fail: (7) q2(a) ?
```

No

- Comentario: No monotonía

# Negación como fallo

- Programa 3: Def. de negación

```
q3(a) :- q3(b), q3(c).
q3(a) :- no(q3(b)), q3(d).
q3(d).
```

```
no(P) :- P, !, fail.
no(P).
```

- Traza

```
?- q3(a).
 Call: (7) q3(a) ?
 Call: (8) q3(b) ?
 Fail: (8) q3(b) ?
 Redo: (7) q3(a) ?
 Call: (8) no(q3(b)) ?
 Call: (9) q3(b) ?
 Fail: (9) q3(b) ?
 Redo: (8) no(q3(b)) ?
 Exit: (8) no(q3(b)) ?
 Call: (8) q3(d) ?
 Exit: (8) q3(d) ?
 Exit: (7) q3(a) ?
```

Yes

# Negación como fallo

- Programa 4

```
q4(a) :- q4(b), q4(c).
q4(a) :- no(q4(b)), q4(d).
q4(d).
q4(b).
```

```
no(P) :- P, !, fail.
no(P).
```

- Traza

```
?- q4(a).
 Call: (7) q4(a) ?
 Call: (8) q4(b) ?
 Exit: (8) q4(b) ?
 Call: (8) q4(c) ?
 Fail: (8) q4(c) ?
 Redo: (7) q4(a) ?
 Call: (8) no(q4(b)) ?
 Call: (9) q4(b) ?
 Exit: (9) q4(b) ?
 Call: (9) fail ?
 Fail: (9) fail ?
 Fail: (8) no(q4(b)) ?
 Fail: (7) q4(a) ?
```

No

- Comentarios:

- Eficiencia y claridad
- Metapredicado primitivo not

# Negación como fallo

- Problemas con negación como fallo

- Programa 1

```
aprobado(X) :- not(suspenso(X)), matriculado(X).
matriculado(juan).
matriculado(luis).
suspenso(juan).
```

- Traza

```
?- aprobado(luis).
Call: (8) aprobado(luis) ?
Call: (9) not(suspenso(luis)) ?
Call: (10) suspenso(luis) ?
Fail: (10) suspenso(luis) ?
Redo: (9) not(suspenso(luis)) ?
Exit: (9) not(suspenso(luis)) ?
Call: (9) matriculado(luis) ?
Exit: (9) matriculado(luis) ?
Exit: (8) aprobado(luis) ?
```

Yes

```
?- aprobado(X).
Call: (8) aprobado(_G112) ?
Call: (9) not(suspenso(_G112)) ?
Call: (10) suspenso(_G112) ?
Exit: (10) suspenso(juan) ?
Call: (10) fail ?
Fail: (10) fail ?
Fail: (9) not(suspenso(_G112)) ?
Fail: (8) aprobado(_G112) ?
```

No

# Negación como fallo

- Programa 2

```
aprobado(X) :- matriculado(X), not(suspenso(X)).
matriculado(juan).
matriculado(luis).
suspenso(juan).
```

- Traza

```
?- aprobado(X).
 Call: (8) aprobado(_G112) ?
 Call: (9) matriculado(_G112) ?
 Exit: (9) matriculado(juan) ?
 Call: (9) not(suspenso(juan)) ?
 Call: (10) suspenso(juan) ?
 Exit: (10) suspenso(juan) ?
 Call: (10) fail ?
 Fail: (10) fail ?
 Fail: (9) not(suspenso(juan)) ?
 Redo: (9) matriculado(_G112) ?
 Exit: (9) matriculado(luis) ?
 Call: (9) not(suspenso(luis)) ?
 Call: (10) suspenso(luis) ?
 Fail: (10) suspenso(luis) ?
 Redo: (9) not(suspenso(luis)) ?
 Exit: (9) not(suspenso(luis)) ?
 Exit: (8) aprobado(luis) ?
```

X = luis

Yes

- Consejo: Asegurar que not se llame con átomos básicos

## Bibliografía

- Bratko, I. *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
  - Cap. 5: “Controlling backtracking”
- Clocksin, W.F. y Mellish, C.S. *Programming in Prolog (Fourth Edition)* (Springer Verlag, 1994)
  - Cap. 4: “Backtracking and the cut”
- Flach, P. *Simply Logical (Intelligent Reasoning by Example)* (John Wiley, 1994)
  - Cap. 3: “Logic programming and Prolog”.
- Sterling,, L. y Shapiro, E. *The Art of Prolog (2nd editition)* (The MIT Press, 1994)
  - Cap. 11: “Cuts and negation”
- Van Le, T. *Techniques of Prolog Programming* (John Wiley, 1993)
  - Cap. 4: “Control and side–effect features of Prolog”