Lógica informática (2014–15)

Tema 13: Introducción a la programación lógica con Prolog

José A. Alonso Jiménez
Andrés Cordón Franco
María J. Hidalgo Doblado

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

Tema 13: Introducción a la programación lógica con Prolog

Tema 13: Introducción a la programación lógica con Prolog

Deducción Prolog en lógica proposicional

- Base de conocimiento y objetivo:
  - Base de conocimiento:
    - Regla 1: Si un animal es ungulado y tiene rayas negras, entonces es una cebra.
    - Regla 2: Si un animal rumia y es mamífero, entonces es ungulado.
    - Regla 3: Si un animal es mamífero y tiene pezuñas, entonces es ungulado.
    - Hecho 1: El animal es mamífero.
    - Hecho 2: El animal tiene pezuñas.
    - Hecho 3: El animal tiene rayas negras.
  - Objetivo: Demostrar a partir de la base de conocimientos que el animal es una cebra.

Deducción Prolog en lógica proposicional

- Programa:

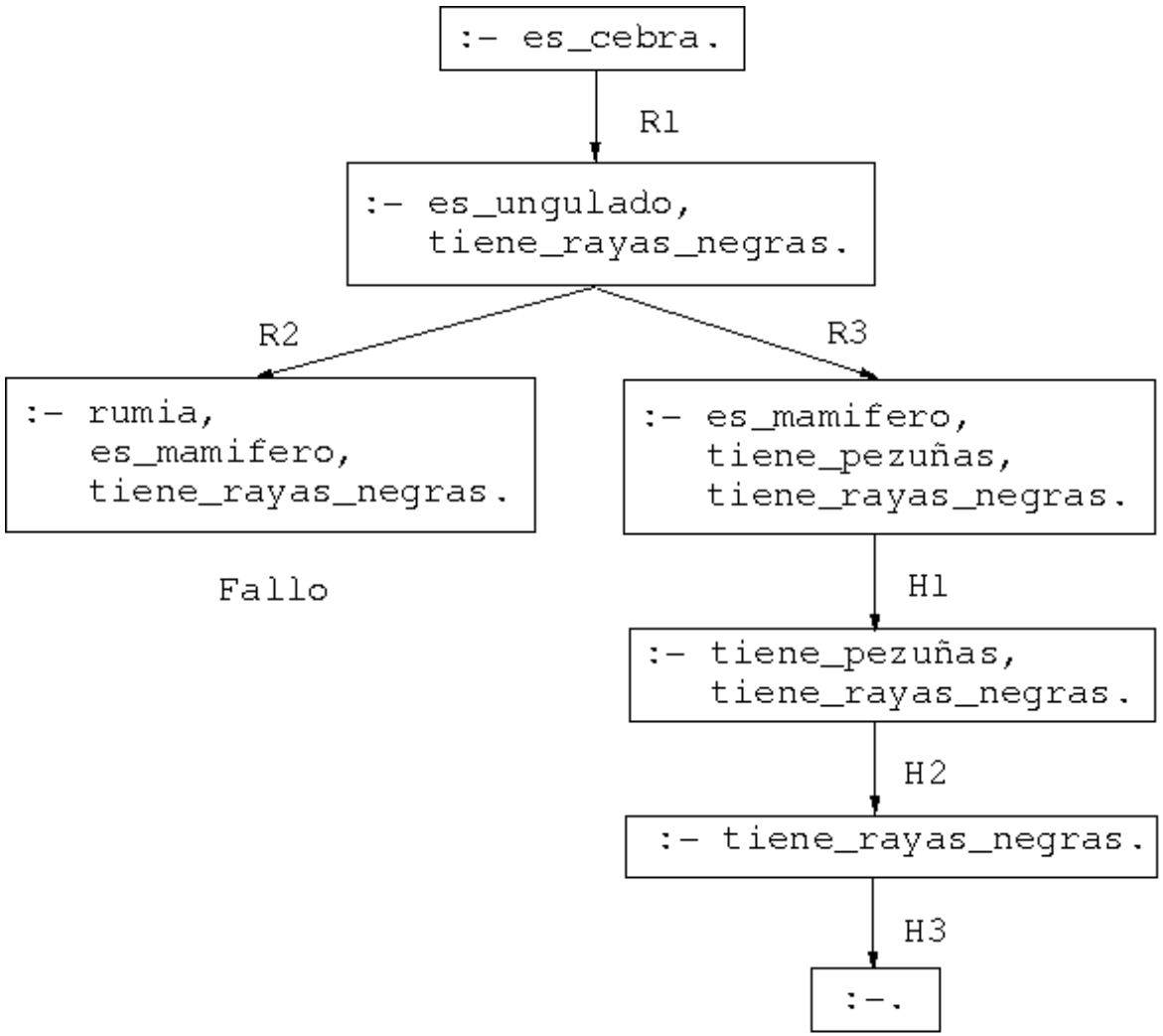
```
es_cebra    :- es_ungulado, tiene_rayas_negras. %R1
es_ungulado :- rumia, es_mamífero.              %R2
es_ungulado :- es_mamífero, tiene_pezuñas.      %R3
es_mamífero.                                    %H1
tiene_pezuñas.                                  %H2
tiene_rayas_negras.                             %H3
```

- Sesión:

```
> pl
Welcome to SWI-Prolog (Multi-threaded, Version 5.6.20)
Copyright (c) 1990-2006 University of Amsterdam.
?- [animales].
Yes
?- es_cebra.
```

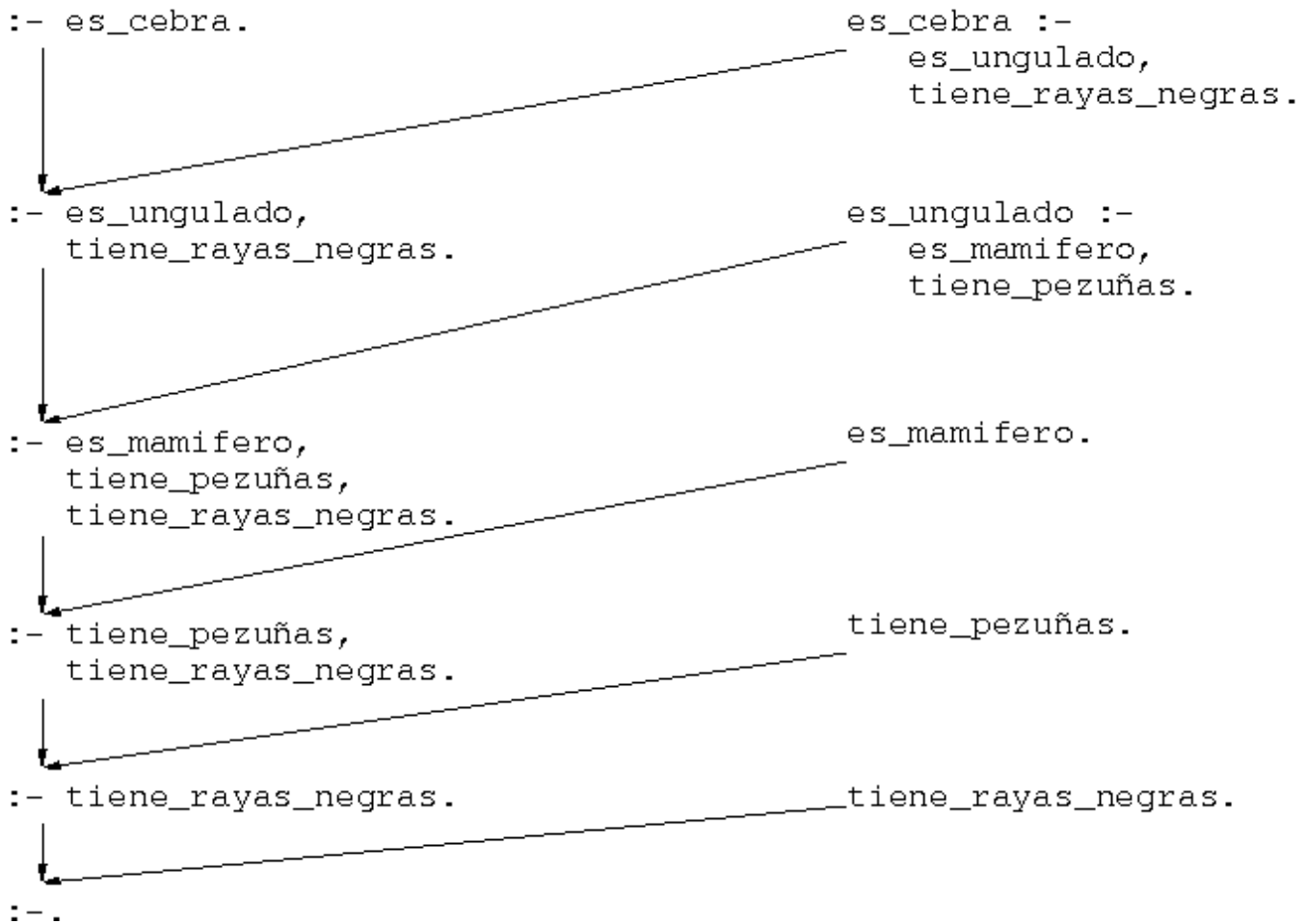
Deducción Prolog en lógica proposicional

- Árbol de deducción:



Deducción Prolog en lógica proposicional

- Demostración por resolución SLD:



Deducción Prolog en lógica relacional

- Base de conocimiento:
  - Hechos 1-4: 6 y 12 son divisibles por 2 y por 3.
  - Hecho 5: 4 es divisible por 2.
  - Regla 1: Los números divisibles por 2 y por 3 son divisibles por 6.
- Programa:

```
divide(2,6).                          % Hecho 1
divide(2,4).                          % Hecho 2
divide(2,12).                         % Hecho 3
divide(3,6).                          % Hecho 4
divide(3,12).                         % Hecho 5
divide(6,X) :- divide(2,X), divide(3,X). % Regla 1
```

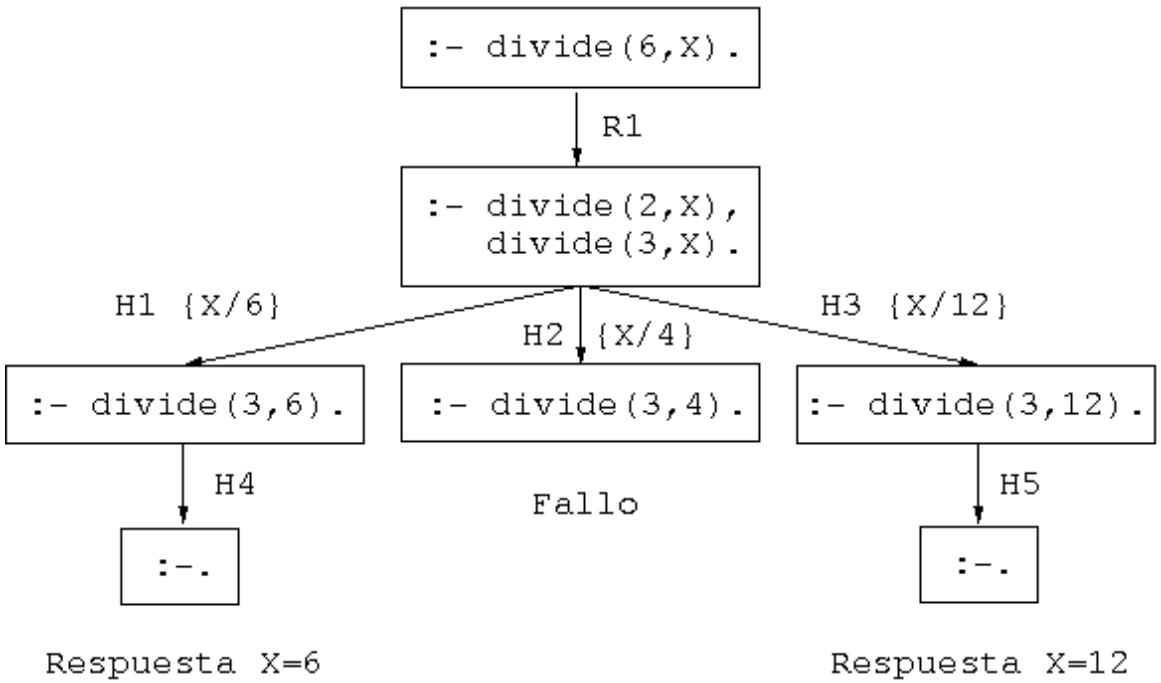
Deducción Prolog en lógica relacional

- Símbolos:
  - Constantes: 2, 3, 4, 6, 12
  - Relación binaria: `divide`
  - Variable: `X`
- Interpretaciones de la Regla 1:
  - `divide(6,X) :- divide(2,X), divide(3,X).`
  - Interpretación declarativa:
    $(\forall X)[divide(2, X) \wedge divide(3, X) \rightarrow divide(6, X)]$
  - Interpretación procedimental.
- Consulta: ¿Cuáles son los múltiplos de 6?

```
?- divide(6,X).
X = 6 ;
X = 12 ;
No
```

Deducción Prolog en lógica relacional

- Árbol de deducción:



- Comentarios:
  - Unificación.
  - Cálculo de respuestas.
  - Respuestas múltiples.

Deducción Prolog en lógica funcional

- Representación de los números naturales:

  `0`, `s(0)`, `s(s(0))`, ...

- Definición de la suma:

```
0 + Y = Y
s(X) + Y = s(X+Y)
```

- Programa

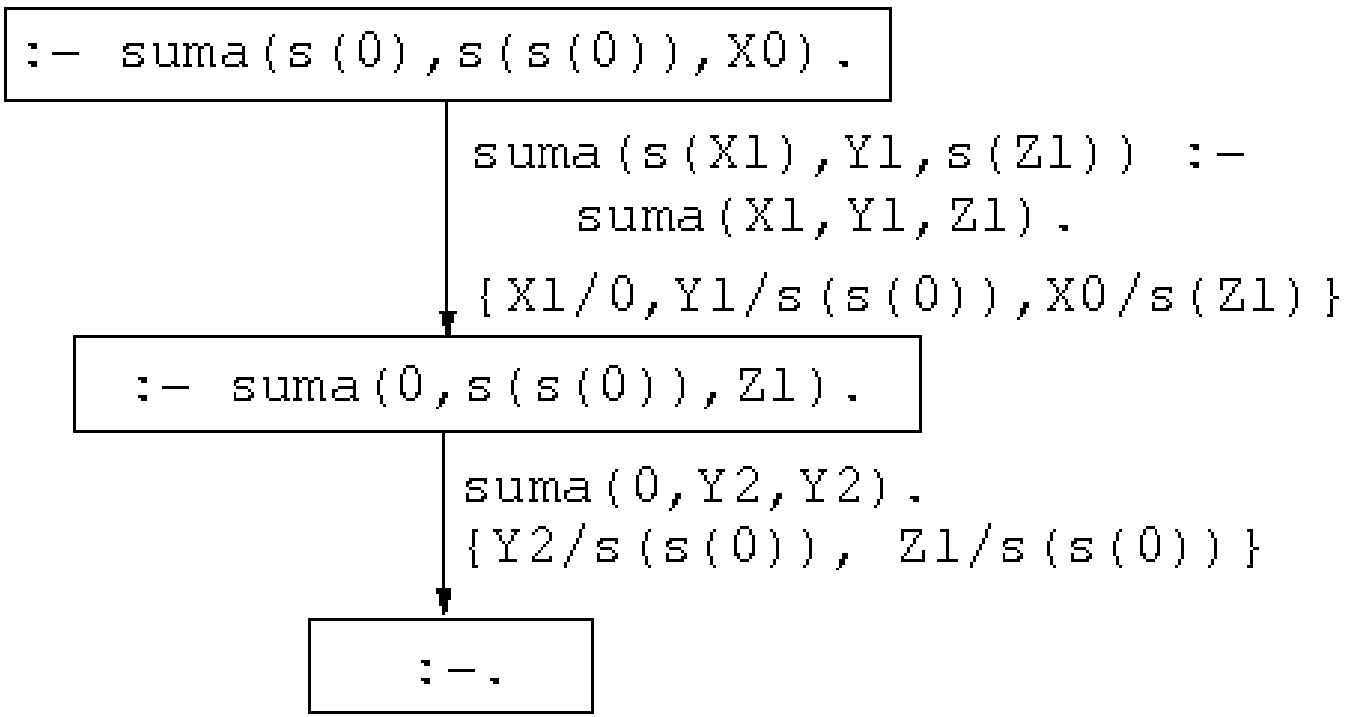
```
suma(0,Y,Y).                          % R1
suma(s(X),Y,s(Z)) :- suma(X,Y,Z).     % R2
```

- Consulta: ¿Cuál es la suma de `s(0)` y `s(s(0))`?

```
?- suma(s(0),s(s(0)),X).
X = s(s(s(0)))
Yes
```

Deducción Prolog en lógica funcional

- Árbol de deducción:



Resp.: X = X0 = s(Z1) = s(s(s(0)))

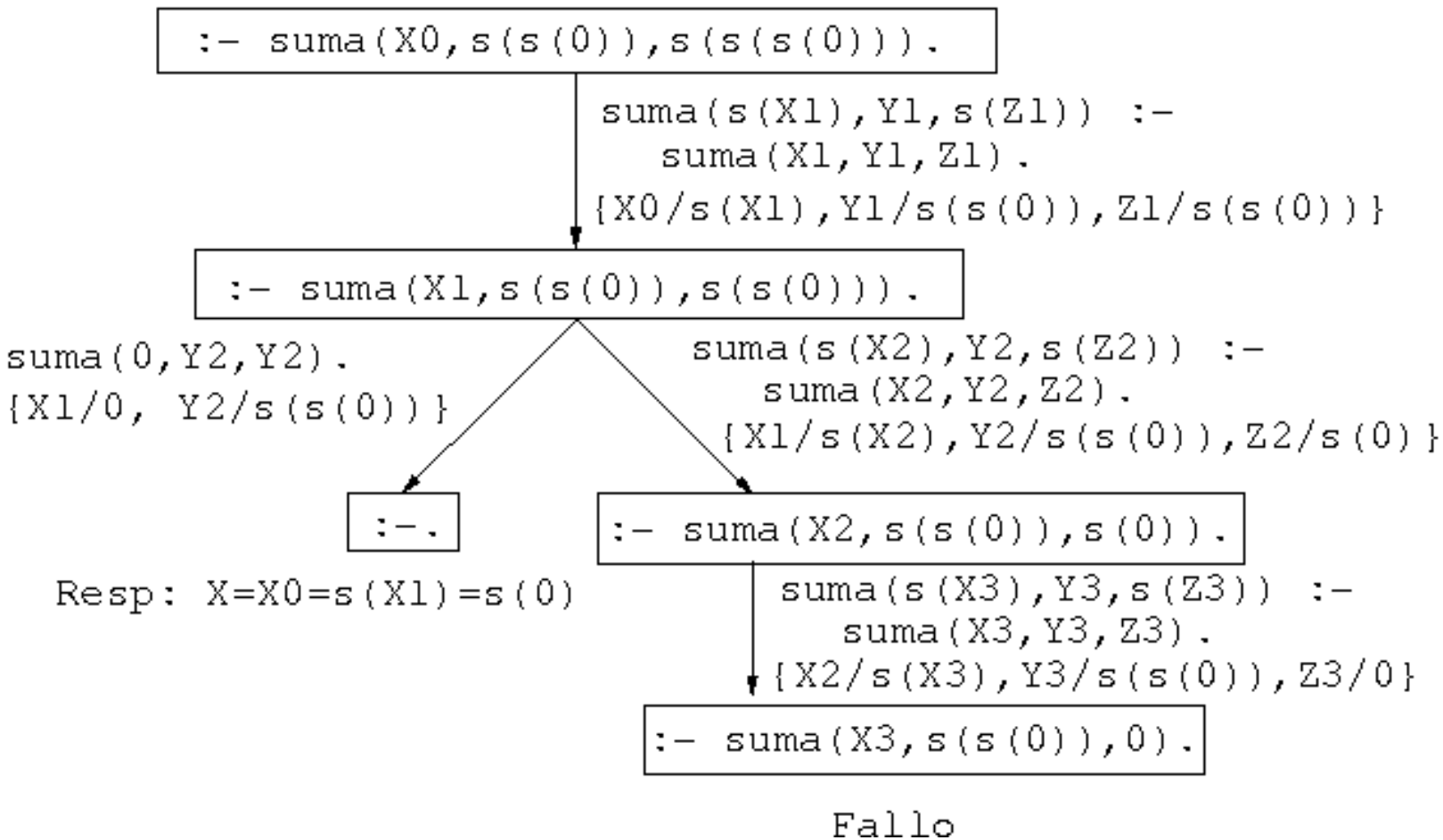
Deducción Prolog en lógica funcional

- Consulta:
  - ¿Cuál es la resta de `s(s(s(0)))` y `s(s(0))`?
  - Sesión:

```
?- suma(X,s(s(0)),s(s(s(0)))).
X = s(0) ;
No
```

Deducción Prolog en lógica funcional

- Árbol de deducción:



Deducción Prolog en lógica funcional

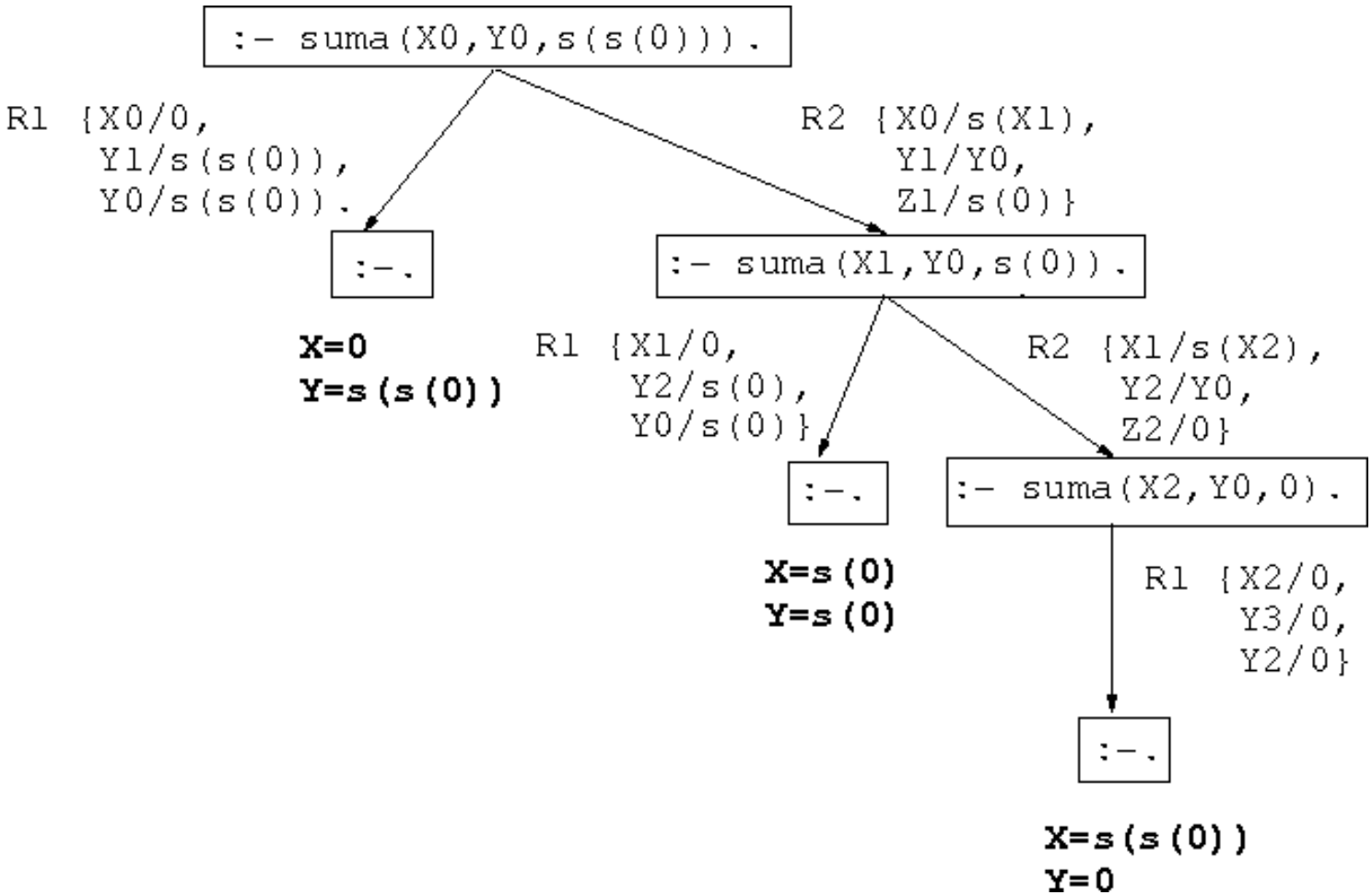
- Consulta:
  - Pregunta: ¿Cuáles son las soluciones de la ecuación

    ```
    X + Y = s(s(0))?
    ```
  - Sesión:

    ```
    ?- suma(X,Y,s(s(0))).
    X = 0          Y = s(s(0)) ;
    X = s(0)       Y = s(0) ;
    X = s(s(0))    Y = 0 ;
    No
    ```

Deducción Prolog en lógica funcional

- Árbol de deducción:



Deducción Prolog en lógica funcional

- Consulta:
  - Pregunta: resolver el sistema de ecuaciones

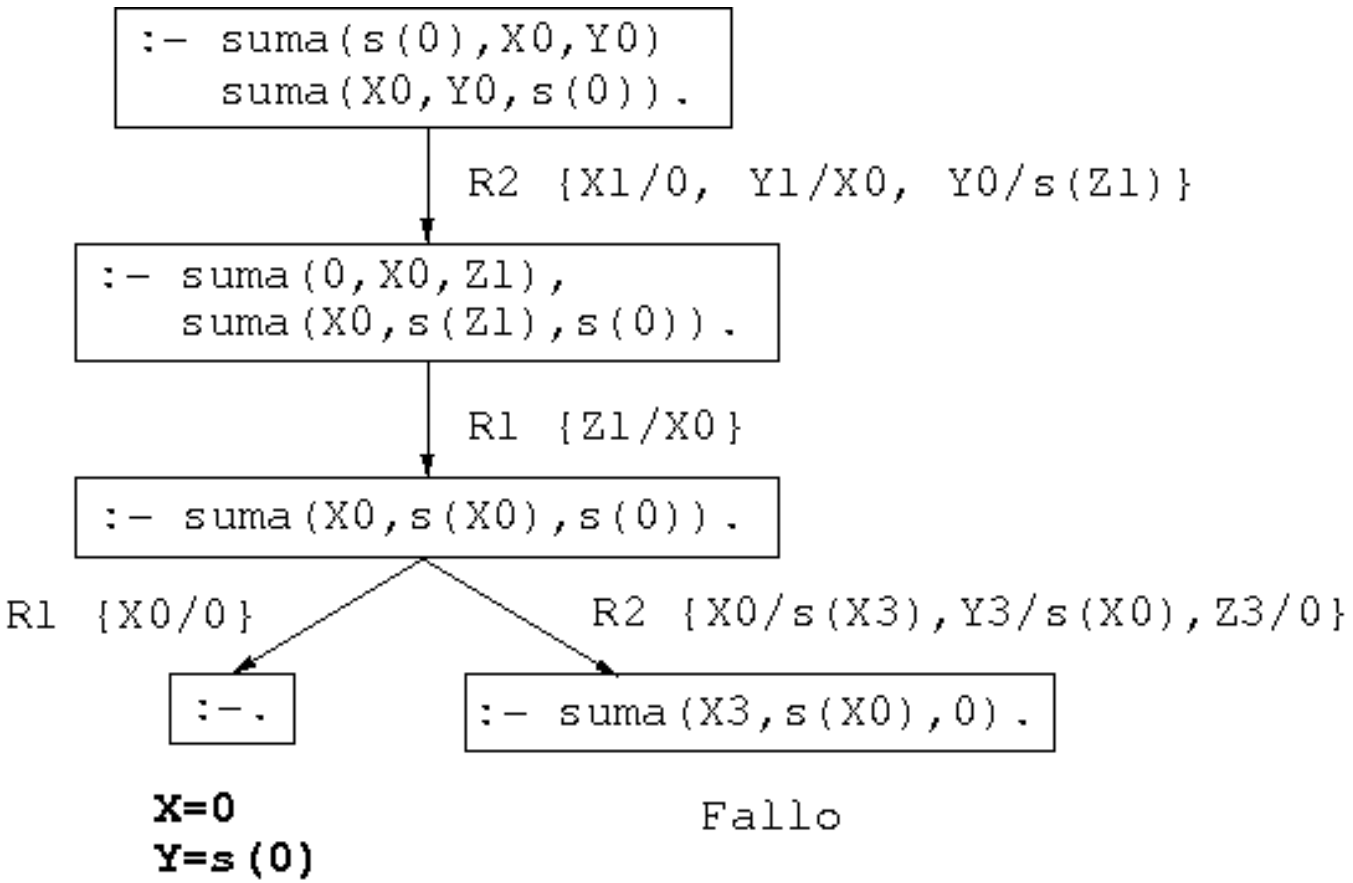
$$1 + X = Y$$
$$X + Y = 1$$

  - Sesión:

```
?- suma(s(0),X,Y), suma(X,Y,s(0)).
X = 0
Y = s(0) ;
No
```

Deducción Prolog en lógica funcional

- Árbol de deducción:



Tema 13: Introducción a la programación lógica con Prolog

Representación de listas

```
?- [X|Y] = [a,b].
X = a
Y = [b]
?- [X|Y] = [a,b,c,d].
X = a
Y = [b, c, d]
?- [X,Y|Z] = [a,b,c,d].
X = a
Y = b
Z = [c, d]
```

Definición de concatenación (append)

- *Especificación:* `conc(A,B,C)` se verifica si `C` es la lista obtenida escribiendo los elementos de la lista `B` a continuación de los elementos de la lista `A`. Por ejemplo,

```
?- conc([a,b],[b,d],C).
C =[a,b,b,d]
```

- *Definición 1:*

```
conc(A,B,C) :- A=[], C=B.
conc(A,B,C) :- A=[X|D], conc(D,B,E), C=[X|E].
```

- *Definición 2:*

```
conc([],B,B).
conc([X|D],B,[X|E]) :- conc(D,B,E).
```

Definición de concatenación (append)

- *Especificación:* `conc(A,B,C)` se verifica si `C` es la lista obtenida escribiendo los elementos de la lista `B` a continuación de los elementos de la lista `A`. Por ejemplo,

```
?- conc([a,b],[b,d],C).
C =[a,b,b,d]
```

- *Definición 1:*

```
conc(A,B,C) :- A=[], C=B.
conc(A,B,C) :- A=[X|D], conc(D,B,E), C=[X|E].
```

- *Definición 2:*

```
conc([],B,B).
conc([X|D],B,[X|E]) :- conc(D,B,E).
```

Definición de concatenación (append)

- *Especificación:* `conc(A,B,C)` se verifica si `C` es la lista obtenida escribiendo los elementos de la lista `B` a continuación de los elementos de la lista `A`. Por ejemplo,

```
?- conc([a,b],[b,d],C).
C =[a,b,b,d]
```

- *Definición 1:*

```
conc(A,B,C) :- A=[], C=B.
conc(A,B,C) :- A=[X|D], conc(D,B,E), C=[X|E].
```

- *Definición 2:*

```
conc([],B,B).
conc([X|D],B,[X|E]) :- conc(D,B,E).
```

Consultas con la relación de concatenación

- Analogía entre la definición de `conc` y la de `suma`,
- ¿Cuál es el resultado de concatenar las listas `[a,b]` y `[c,d,e]`?

```
?- conc([a,b],[c,d,e],L).
L = [a, b, c, d, e]
```

- ¿Qué lista hay que añadirle a la lista `[a,b]` para obtener `[a,b,c,d]`?

```
?- conc([a,b],L,[a,b,c,d]).
L = [c, d]
```

- ¿Qué dos listas hay que concatenar para obtener `[a,b]`?

```
?- conc(L,M,[a,b]).
L = []      M = [a, b] ;
L = [a]     M = [b] ;
L = [a, b]  M = [] ;
No
```

Consultas con la relación de concatenación

- Analogía entre la definición de `conc` y la de `suma`,
- ¿Cuál es el resultado de concatenar las listas `[a,b]` y `[c,d,e]`?

```
?- conc([a,b],[c,d,e],L).
L = [a, b, c, d, e]
```

- ¿Qué lista hay que añadirle a la lista `[a,b]` para obtener `[a,b,c,d]`?

```
?- conc([a,b],L,[a,b,c,d]).
L = [c, d]
```

- ¿Qué dos listas hay que concatenar para obtener `[a,b]`?

```
?- conc(L,M,[a,b]).
L = []      M = [a, b] ;
L = [a]     M = [b] ;
L = [a, b]  M = [] ;
No
```

Consultas con la relación de concatenación

- Analogía entre la definición de `conc` y la de `suma`,
- ¿Cuál es el resultado de concatenar las listas `[a,b]` y `[c,d,e]`?

```
?- conc([a,b],[c,d,e],L).
L = [a, b, c, d, e]
```

- ¿Qué lista hay que añadirle a la lista `[a,b]` para obtener `[a,b,c,d]`?

```
?- conc([a,b],L,[a,b,c,d]).
L = [c, d]
```

- ¿Qué dos listas hay que concatenar para obtener `[a,b]`?

```
?- conc(L,M,[a,b]).
L = []      M = [a, b] ;
L = [a]     M = [b] ;
L = [a, b]  M = [] ;
No
```

Consultas con la relación de concatenación

- Analogía entre la definición de `conc` y la de `suma`,
- ¿Cuál es el resultado de concatenar las listas `[a,b]` y `[c,d,e]`?

```
?- conc([a,b],[c,d,e],L).
L = [a, b, c, d, e]
```

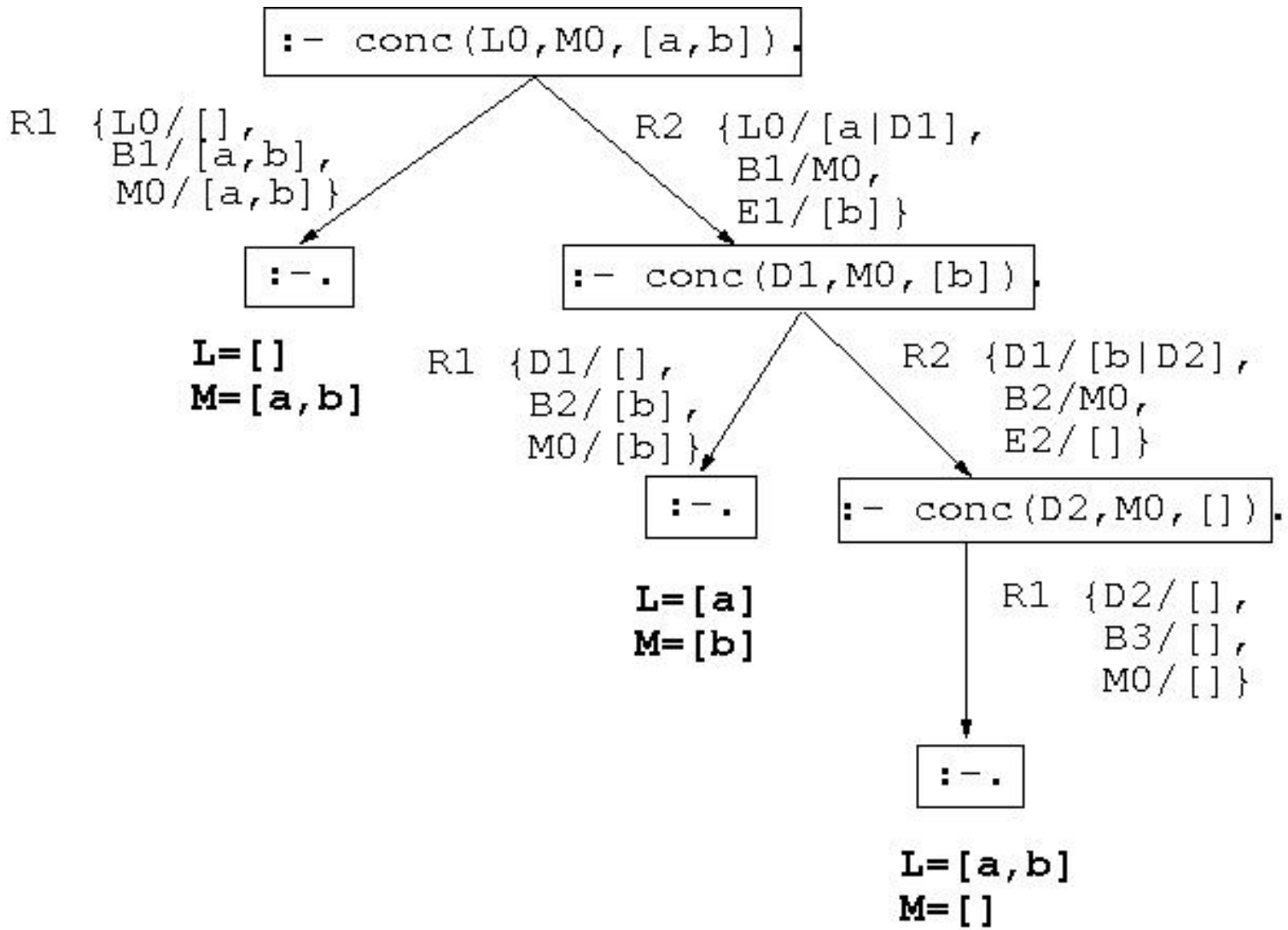
- ¿Qué lista hay que añadirle a la lista `[a,b]` para obtener `[a,b,c,d]`?

```
?- conc([a,b],L,[a,b,c,d]).
L = [c, d]
```

- ¿Qué dos listas hay que concatenar para obtener `[a,b]`?

```
?- conc(L,M,[a,b]).
L = []        M = [a, b] ;
L = [a]       M = [b] ;
L = [a, b]    M = [] ;
No
```

Árbol de deducción de `?- conc(L,M,[a,b]).`



Definición de la relación de pertenencia (`member`)

- *Especificación:* `pertenece(X,L)` se verifica si `X` es un elemento de la lista `L`.
- *Definición 1:*

```
pertenece(X,[X|L]).
pertenece(X,[Y|L]) :- pertenece(X,L).
```

- *Definición 2:*

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

Definición de la relación de pertenencia (`member`)

- *Especificación:* `pertenece(X,L)` se verifica si `X` es un elemento de la lista `L`.
- *Definición 1:*

```
pertenece(X,[X|L]).
pertenece(X,[Y|L]) :- pertenece(X,L).
```

- *Definición 2:*

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

Definición de la relación de pertenencia (`member`)

- *Especificación:* `pertenece(X,L)` se verifica si `X` es un elemento de la lista `L`.
- *Definición 1:*

```
pertenece(X,[X|L]).
pertenece(X,[Y|L]) :- pertenece(X,L).
```

- *Definición 2:*

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

Consultas con la relación de pertenencia

```
?- pertenece(b,[a,b,c]).
Yes
?- pertenece(d,[a,b,c]).
No
?- pertenece(X,[a,b,a]).
X = a ;
X = b ;
X = a ;
No
?- pertenece(a,L).
L = [a|_G233] ;
L = [_G232, a|_G236] ;
L = [_G232, _G235, a|_G239]
Yes
```

Consultas con la relación de pertenencia

```
?- pertenece(b,[a,b,c]).
Yes
?- pertenece(d,[a,b,c]).
No
?- pertenece(X,[a,b,a]).
X = a ;
X = b ;
X = a ;
No
?- pertenece(a,L).
L = [a|_G233] ;
L = [_G232, a|_G236] ;
L = [_G232, _G235, a|_G239]
Yes
```

Consultas con la relación de pertenencia

```
?- pertenece(b,[a,b,c]).
Yes
?- pertenece(d,[a,b,c]).
No
?- pertenece(X,[a,b,a]).
X = a ;
X = b ;
X = a ;
No
?- pertenece(a,L).
L = [a|_G233] ;
L = [_G232, a|_G236] ;
L = [_G232, _G235, a|_G239]
Yes
```

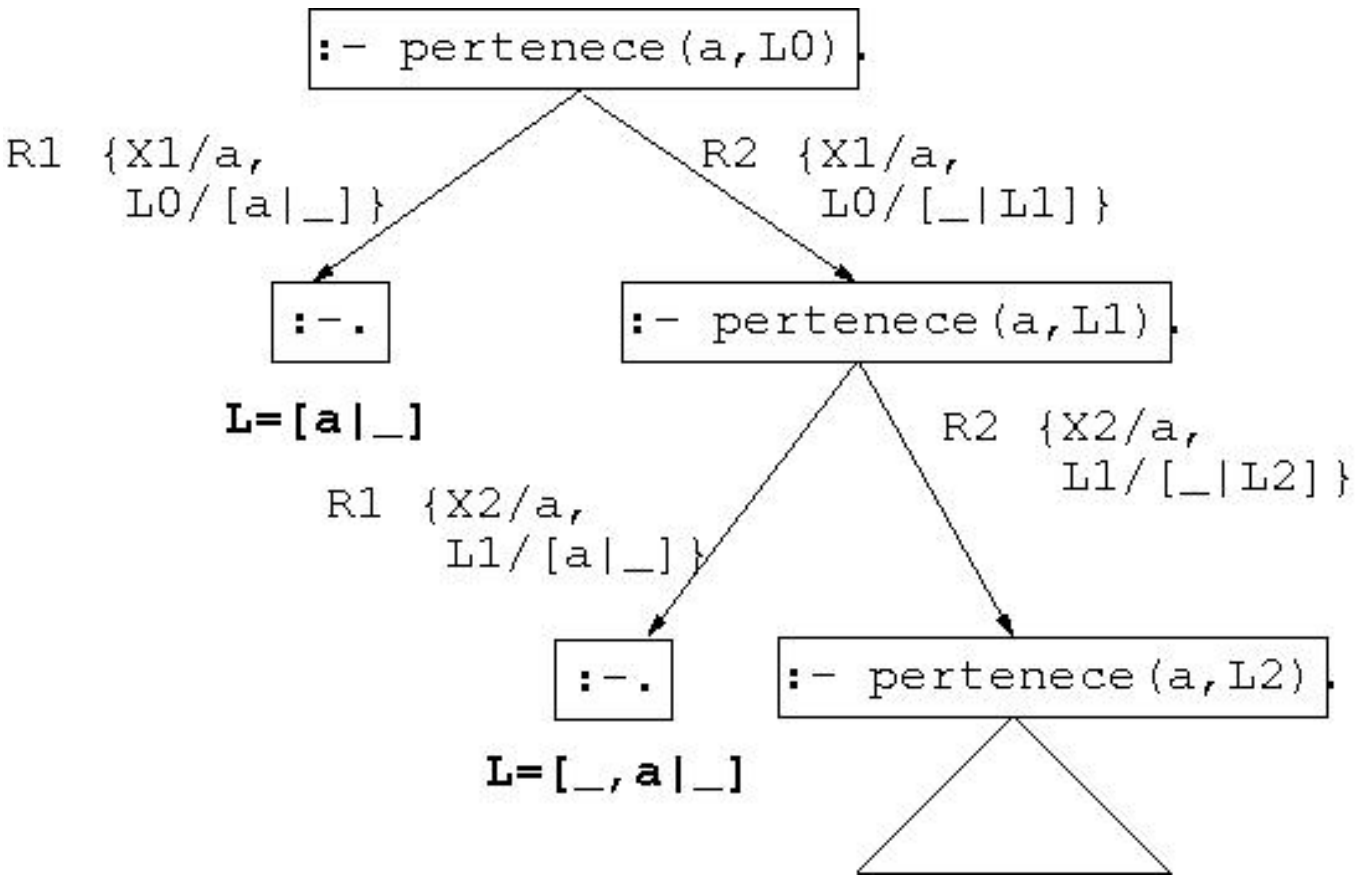
Consultas con la relación de pertenencia

```
?- pertenece(b,[a,b,c]).
Yes
?- pertenece(d,[a,b,c]).
No
?- pertenece(X,[a,b,a]).
X = a ;
X = b ;
X = a ;
No
?- pertenece(a,L).
L = [a|_G233] ;
L = [_G232, a|_G236] ;
L = [_G232, _G235, a|_G239]
Yes
```

Consultas con la relación de pertenencia

```
?- pertenece(b,[a,b,c]).
Yes
?- pertenece(d,[a,b,c]).
No
?- pertenece(X,[a,b,a]).
X = a ;
X = b ;
X = a ;
No
?- pertenece(a,L).
L = [a|_G233] ;
L = [_G232, a|_G236] ;
L = [_G232, _G235, a|_G239]
Yes
```

Árbol de deducción de `?- pertenece(a,L).`



Tema 13: Introducción a la programación lógica con Prolog

Ejemplos de operadores aritméticos

- Ejemplos de notación infija y prefija en expresiones aritméticas:

```
?- +(X,Y) = a+b.
X = a
Y = b
?- +(X,Y) = a+b+c.
X = a+b
Y = c
?- +(X,Y) = a+(b+c).
X = a
Y = b+c
?- a+b+c = (a+b)+c.
Yes
?- a+b+c = a+(b+c).
No
```

Ejemplos de operadores aritméticos

- Ejemplos de notación infija y prefija en expresiones aritméticas:

```
?- +(X,Y) = a+b.
X = a
Y = b
?- +(X,Y) = a+b+c.
X = a+b
Y = c
?- +(X,Y) = a+(b+c).
X = a
Y = b+c
?- a+b+c = (a+b)+c.
Yes
?- a+b+c = a+(b+c).
No
```

Ejemplos de operadores aritméticos

- Ejemplos de notación infija y prefija en expresiones aritméticas:

```
?- +(X,Y) = a+b.
X = a
Y = b
?- +(X,Y) = a+b+c.
X = a+b
Y = c
?- +(X,Y) = a+(b+c).
X = a
Y = b+c
?- a+b+c = (a+b)+c.
Yes
?- a+b+c = a+(b+c).
No
```

Ejemplos de operadores aritméticos

- Ejemplos de notación infija y prefija en expresiones aritméticas:

```
?- +(X,Y) = a+b.
X = a
Y = b
?- +(X,Y) = a+b+c.
X = a+b
Y = c
?- +(X,Y) = a+(b+c).
X = a
Y = b+c
?- a+b+c = (a+b)+c.
Yes
?- a+b+c = a+(b+c).
No
```

Ejemplos de operadores aritméticos

- Ejemplos de notación infija y prefija en expresiones aritméticas:

```
?- +(X,Y) = a+b.
X = a
Y = b
?- +(X,Y) = a+b+c.
X = a+b
Y = c
?- +(X,Y) = a+(b+c).
X = a
Y = b+c
?- a+b+c = (a+b)+c.
Yes
?- a+b+c = a+(b+c).
No
```

Ejemplos de operadores aritméticos

- Ejemplos de notación infija y prefija en expresiones aritméticas:

```
?- +(X,Y) = a+b.
X = a
Y = b
?- +(X,Y) = a+b+c.
X = a+b
Y = c
?- +(X,Y) = a+(b+c).
X = a
Y = b+c
?- a+b+c = (a+b)+c.
Yes
?- a+b+c = a+(b+c).
No
```

Ejemplos de asociatividad y precedencia

- Ejemplos de asociatividad:

```
?- X^Y = a^b^c.
X = a        Y = b^c
?- a^b^c = a^(b^c).
Yes
```

- Ejemplo de precedencia

```
?- X+Y = a+b*c.
X = a        Y = b*c
?- X*Y = a+b*c.
No
?- X*Y = (a+b)*c.
X = a+b      Y = c
?- a+b*c = (a+b)*c.
No
```

Ejemplos de asociatividad y precedencia

- Ejemplos de asociatividad:

```
?- X^Y = a^b^c.
X = a        Y = b^c
?- a^b^c = a^(b^c).
Yes
```

- Ejemplo de precedencia

```
?- X+Y = a+b*c.
X = a        Y = b*c
?- X*Y = a+b*c.
No
?- X*Y = (a+b)*c.
X = a+b      Y = c
?- a+b*c = (a+b)*c.
No
```

Ejemplos de asociatividad y precedencia

- Ejemplos de asociatividad:

```
?- X^Y = a^b^c.
X = a        Y = b^c
?- a^b^c = a^(b^c).
Yes
```

- Ejemplo de precedencia

```
?- X+Y = a+b*c.
X = a        Y = b*c
?- X*Y = a+b*c.
No
?- X*Y = (a+b)*c.
X = a+b      Y = c
?- a+b*c = (a+b)*c.
No
```

Ejemplos de asociatividad y precedencia

- Ejemplos de asociatividad:

```
?- X^Y = a^b^c.
X = a        Y = b^c
?- a^b^c = a^(b^c).
Yes
```

- Ejemplo de precedencia

```
?- X+Y = a+b*c.
X = a        Y = b*c
?- X*Y = a+b*c.
No
?- X*Y = (a+b)*c.
X = a+b      Y = c
?- a+b*c = (a+b)*c.
No
```

Ejemplos de asociatividad y precedencia

- Ejemplos de asociatividad:

```
?- X^Y = a^b^c.
X = a        Y = b^c
?- a^b^c = a^(b^c).
Yes
```

- Ejemplo de precedencia

```
?- X+Y = a+b*c.
X = a        Y = b*c
?- X*Y = a+b*c.
No
?- X*Y = (a+b)*c.
X = a+b      Y = c
?- a+b*c = (a+b)*c.
No
```

Ejemplos de asociatividad y precedencia

- Ejemplos de asociatividad:

```
?- X^Y = a^b^c.
X = a        Y = b^c
?- a^b^c = a^(b^c).
Yes
```

- Ejemplo de precedencia

```
?- X+Y = a+b*c.
X = a        Y = b*c
?- X*Y = a+b*c.
No
?- X*Y = (a+b)*c.
X = a+b      Y = c
?- a+b*c = (a+b)*c.
No
```

Ejemplos de asociatividad y precedencia

- Ejemplos de asociatividad:

```
?- X^Y = a^b^c.
X = a        Y = b^c
?- a^b^c = a^(b^c).
Yes
```

- Ejemplo de precedencia

```
?- X+Y = a+b*c.
X = a        Y = b*c
?- X*Y = a+b*c.
No
?- X*Y = (a+b)*c.
X = a+b      Y = c
?- a+b*c = (a+b)*c.
No
```

Definición de operadores

- Definición (`ejemplo_operadores.pl`)

```
:-op(800,xfx,estudian).
:-op(400,xfx,y).

juan y ana estudian lógica.
```

- Consultas

```
?- [ejemplo_operadores].
?- Quienes estudian lógica.
Quienes = juan y ana
?- juan y Otro estudian Algo.
Otro = ana
Algo = lógica
```

Definición de operadores

- Definición (`ejemplo_operadores.pl`)

```
:-op(800,xfx,estudian).
:-op(400,xfx,y).

juan y ana estudian lógica.
```

- Consultas

```
?- [ejemplo_operadores].
?- Quienes estudian lógica.
Quienes = juan y ana
?- juan y Otro estudian Algo.
Otro = ana
Algo = lógica
```

Definición de operadores

- Definición (`ejemplo_operadores.pl`)

```
:-op(800,xfx,estudian).
:-op(400,xfx,y).

juan y ana estudian lógica.
```

- Consultas

```
?- [ejemplo_operadores].
?- Quienes estudian lógica.
```

```
Quienes = juan y ana
?- juan y Otro estudian Algo.
```

```
Otro = ana
Algo = lógica
```

Tema 13: Introducción a la programación lógica con Prolog

Ejemplo de nota sin corte

- `nota(X,Y)` se verifica si `Y` es la calificación correspondiente a la nota `X`; es decir, `Y` es `suspenso` si `X` es menor que 5, `Y` es `aprobado` si `X` es mayor o igual que 5 pero menor que 7, `Y` es `notable` si `X` es mayor que 7 pero menor que 9 e `Y` es sobresaliente si `X` es mayor que 9. Por ejemplo,

```
?- nota(6,Y).
Y = aprobado;
No
```

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

Ejemplo de nota sin corte

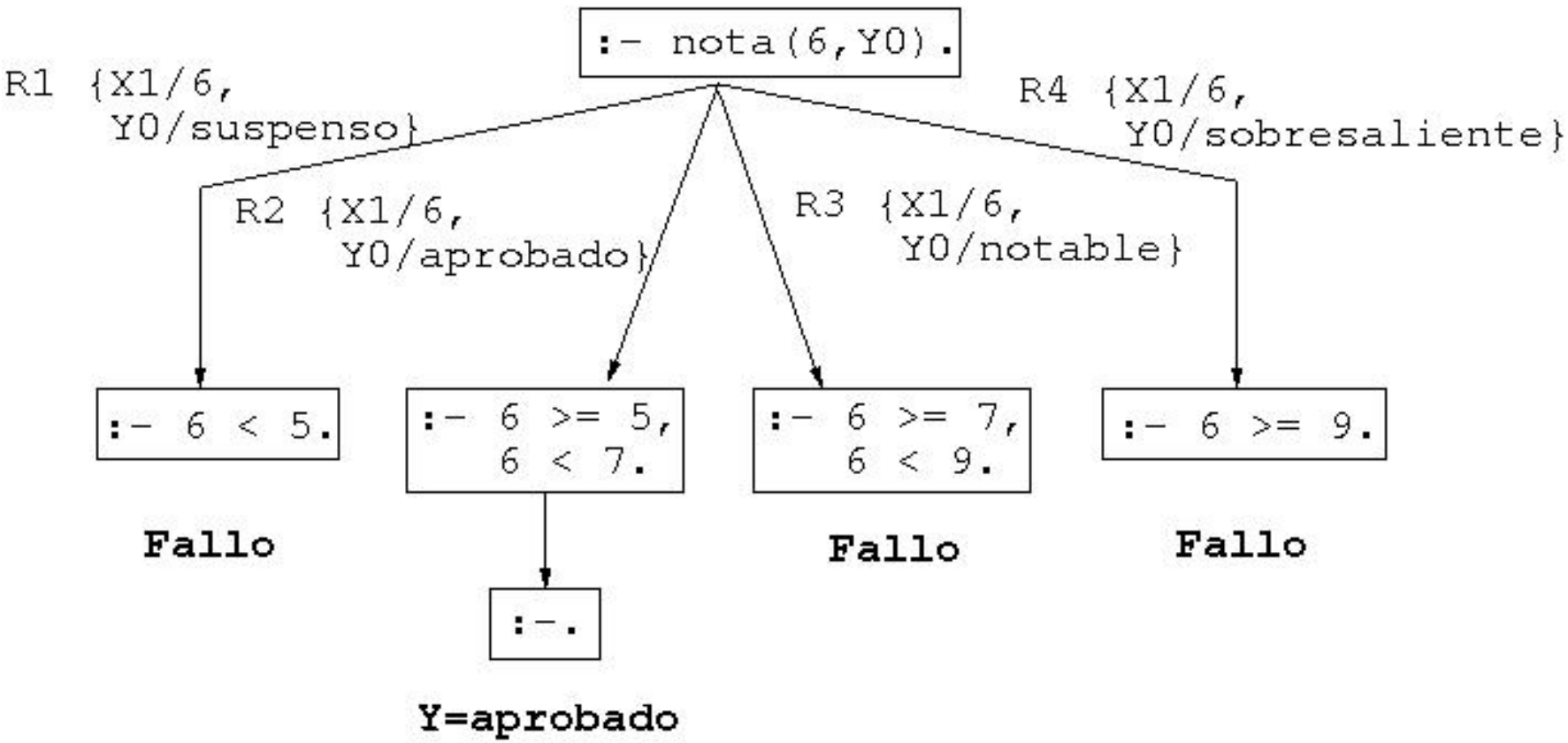
- `nota(X,Y)` se verifica si `Y` es la calificación correspondiente a la nota `X`; es decir, `Y` es `suspenso` si `X` es menor que 5, `Y` es `aprobado` si `X` es mayor o igual que 5 pero menor que 7, `Y` es `notable` si `X` es mayor que 7 pero menor que 9 e `Y` es sobresaliente si `X` es mayor que 9. Por ejemplo,

```
?- nota(6,Y).
Y = aprobado;
No
```

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

Deducción en el ejemplo sin corte

- Árbol de deducción de `?- nota(6,Y).`

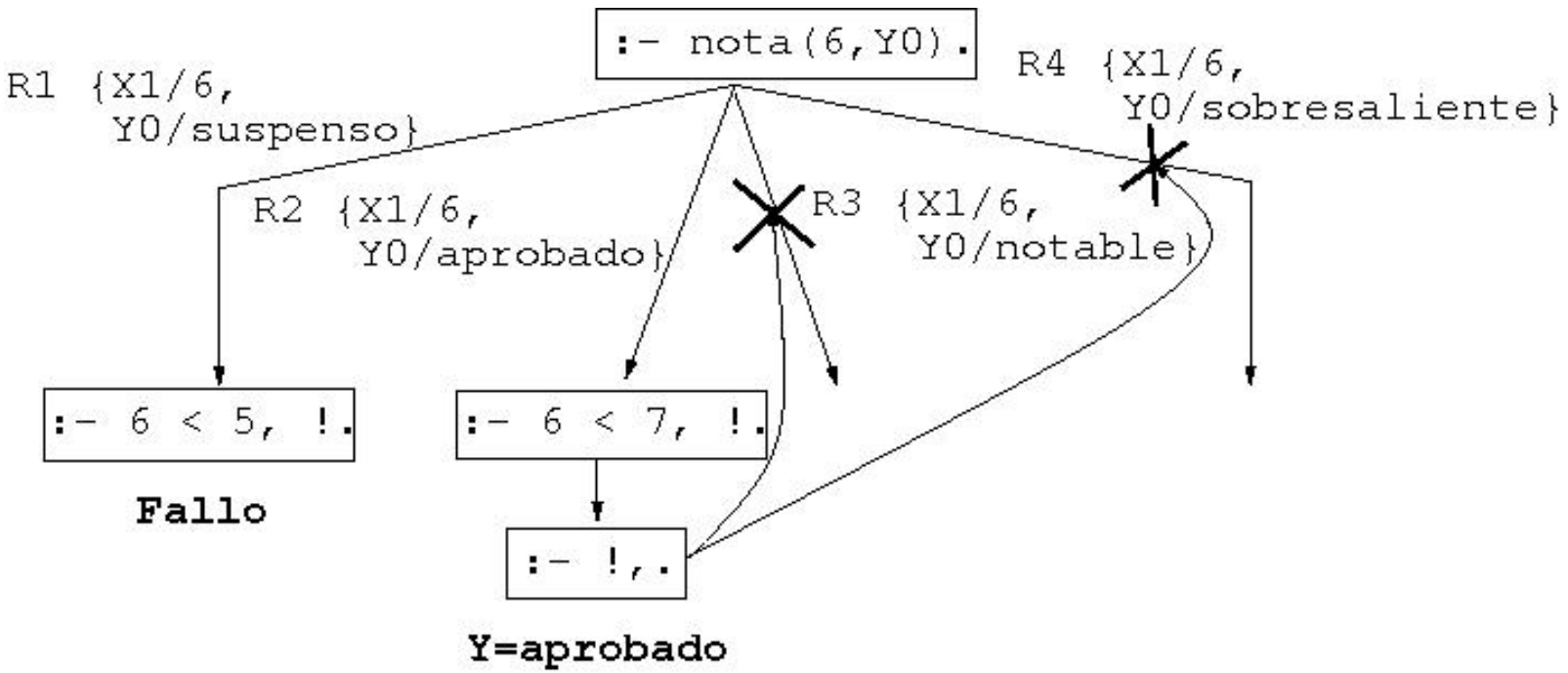


Ejemplo de `nota` con cortes

- Definición de `nota` con cortes

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

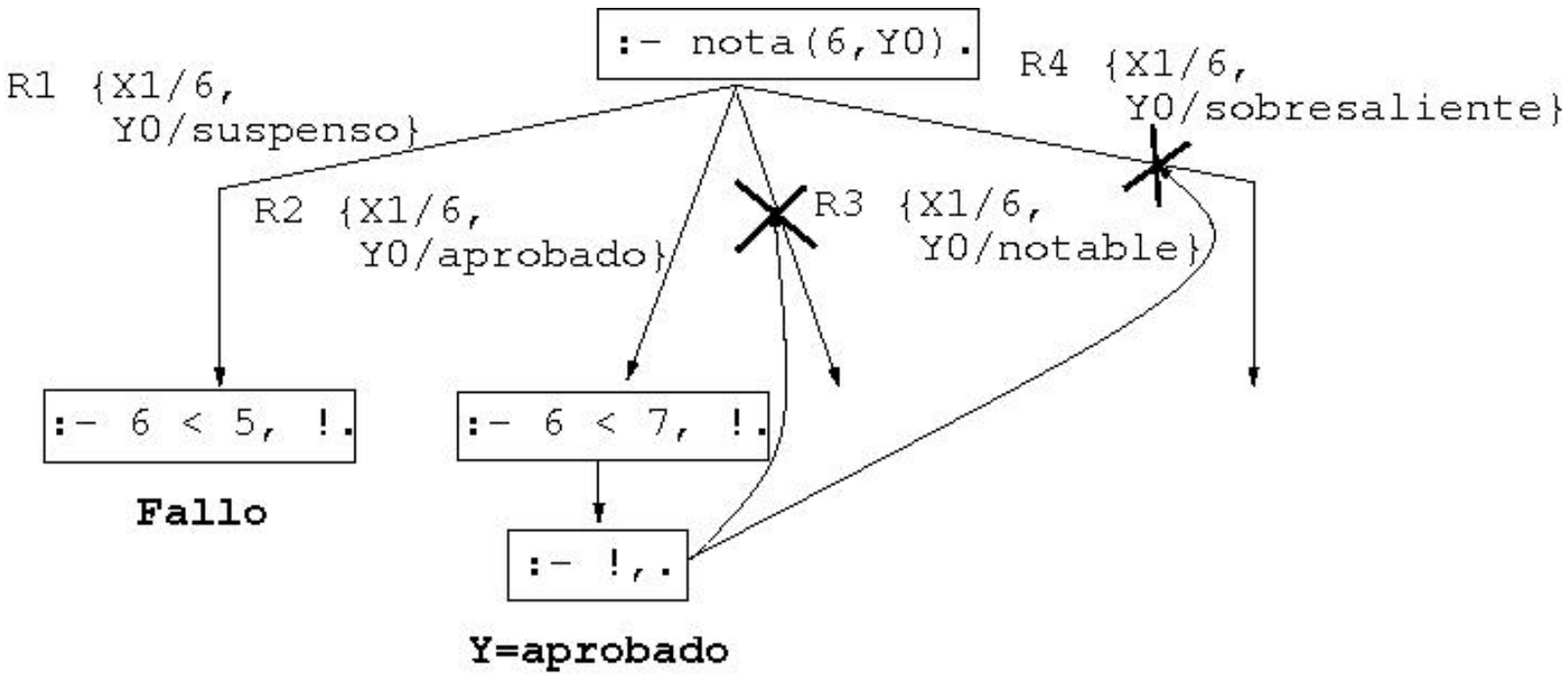
Deducción en el ejemplo con cortes



- ¿Un 6 es un sobresaliente?

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

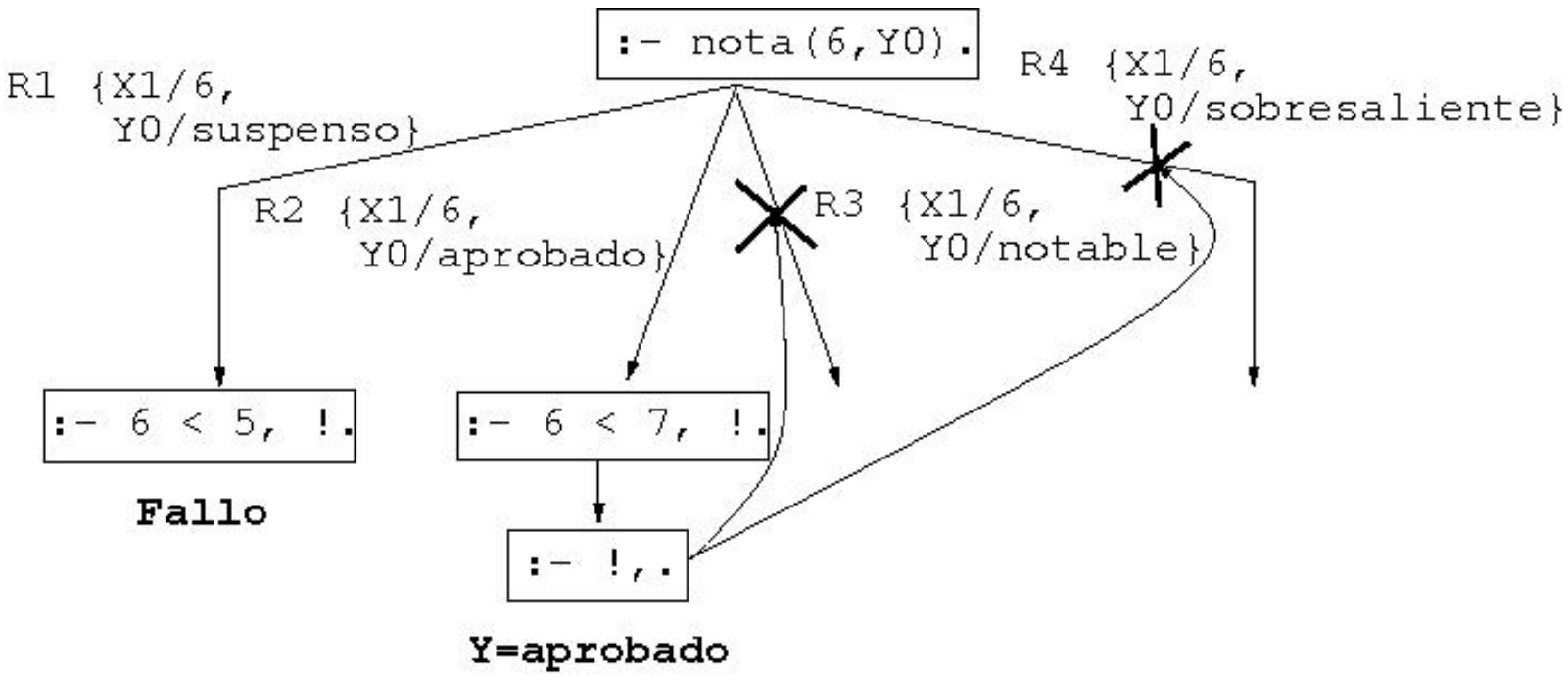
Deducción en el ejemplo con cortes



- ¿Un 6 es un sobresaliente?

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

Deducción en el ejemplo con cortes



- ¿Un 6 es un sobresaliente?

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

Tema 13: Introducción a la programación lógica con Prolog

Definición de la negación como fallo

- Definición de la negación como fallo `not`:

```
no(P) :- P, !, fail.          % No 1
no(P).                        % No 2
```

Definición de la negación como fallo

- Definición de la negación como fallo `not`:

```
no(P) :- P, !, fail.          % No 1
no(P).                        % No 2
```

Programa con negación

- Programa:

```
aprobado(X) :-                    % R1
   no(suspenso(X)),
   matriculado(X).
matriculado(juan).                % R2
matriculado(luis).                % R3
suspenso(juan).                   % R4
```

- Consultas:

```
?- aprobado(luis).
Yes
```

```
?- aprobado(X).
No
```

Programa con negación

- Programa:

```
aprobado(X) :-                    % R1
   no(suspenso(X)),
   matriculado(X).
matriculado(juan).                % R2
matriculado(luis).                % R3
suspenso(juan).                   % R4
```

- Consultas:

```
?- aprobado(luis).
Yes
```

```
?- aprobado(X).
No
```

Programa con negación

- Programa:

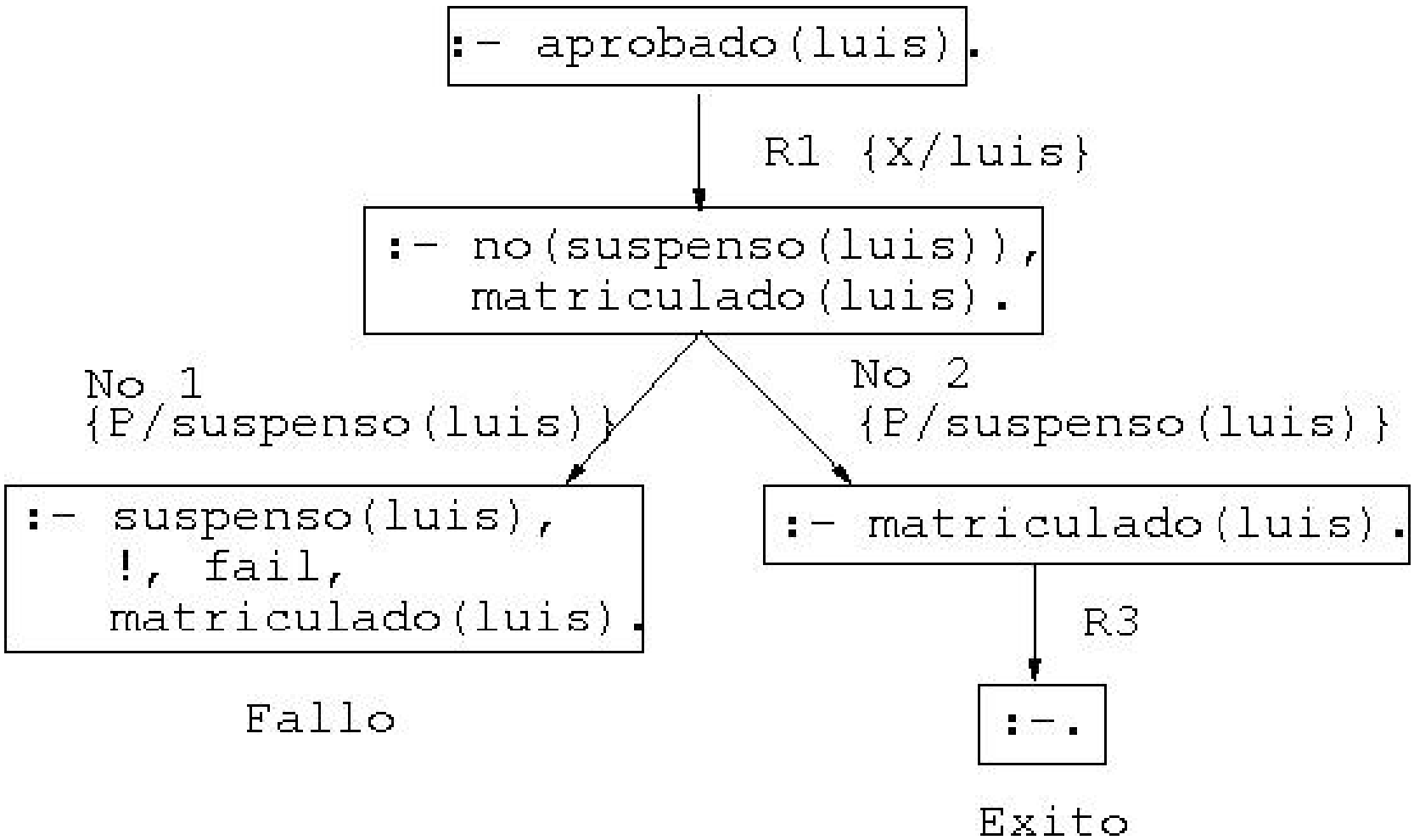
```
aprobado(X) :-                    % R1
   no(suspenso(X)),
   matriculado(X).
matriculado(juan).                % R2
matriculado(luis).                % R3
suspenso(juan).                   % R4
```

- Consultas:

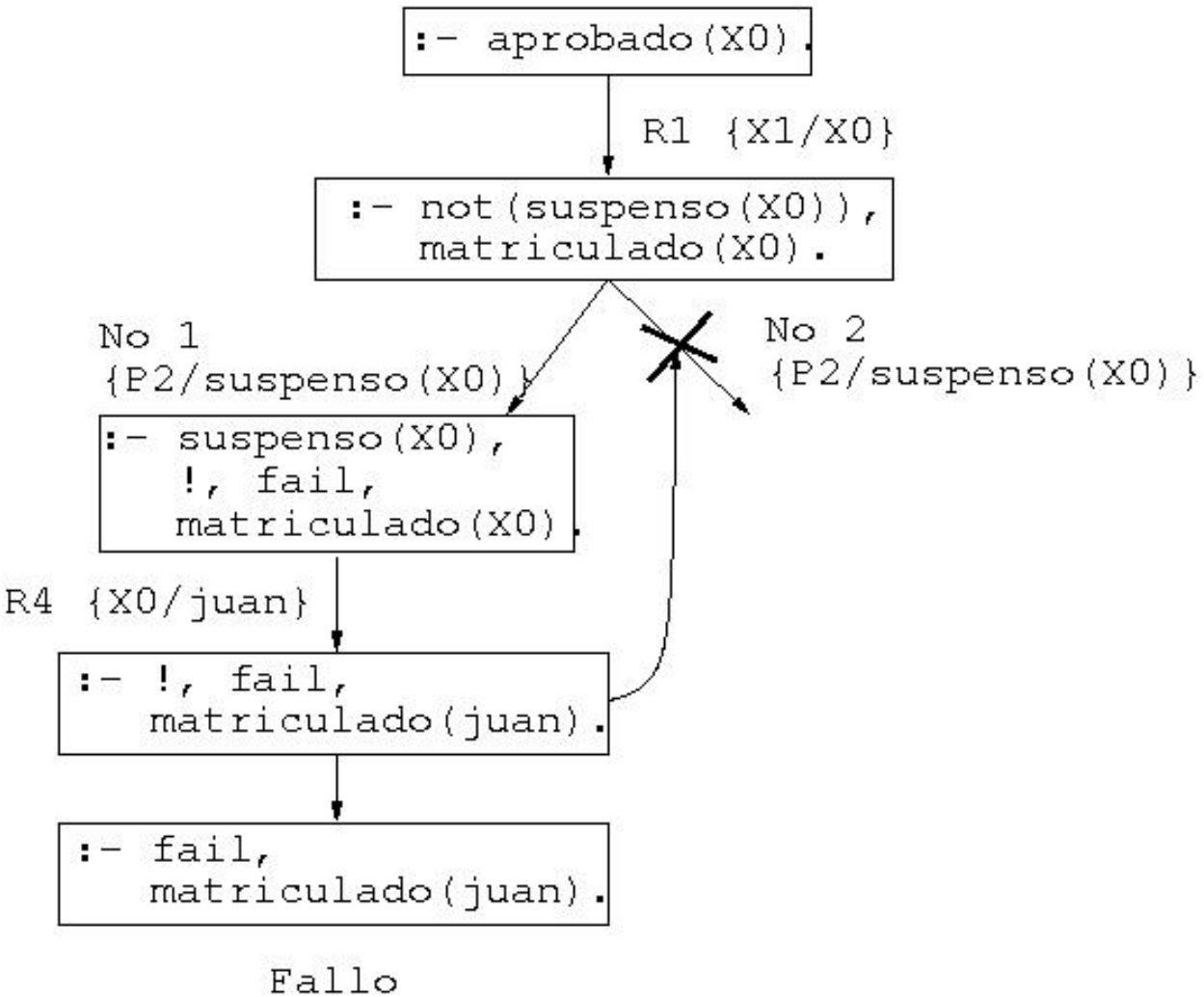
```
?- aprobado(luis).
Yes
```

```
?- aprobado(X).
No
```

Árbol de deducción de ?- aprobado(luis).



Árbol de deducción de ?- aprobado(X).



Modificación del orden de los literales

- Programa:

```
aprobado(X) :-              % R1
   matriculado(X),
   no(suspenso(X)).
matriculado(juan).          % R2
matriculado(luis).          % R3
suspenso(juan).             % R4
```

- Consulta:

```
?- aprobado(X).
X = luis
Yes
```

Modificación del orden de los literales

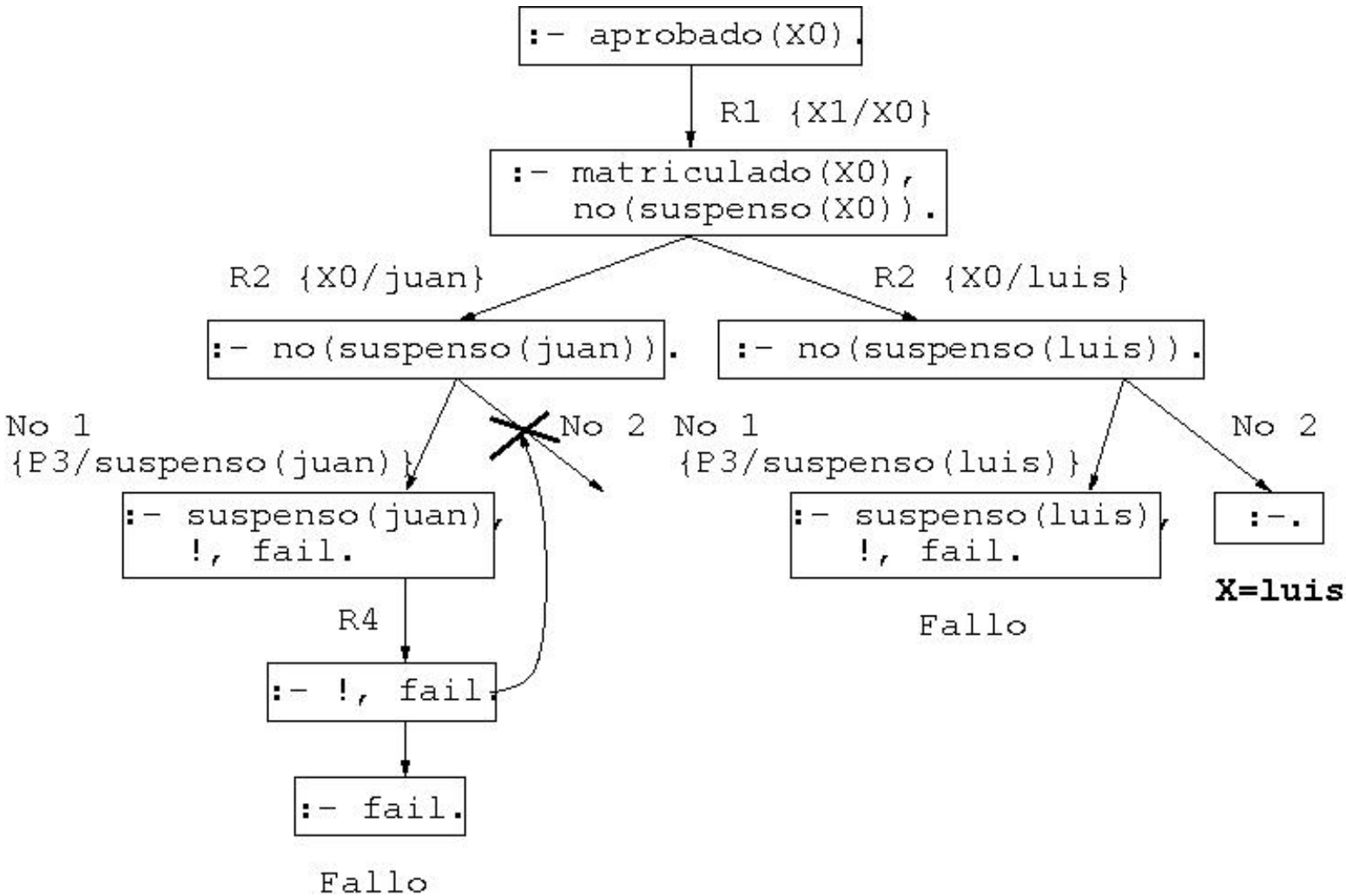
- Programa:

```
aprobado(X) :-              % R1
   matriculado(X),
   no(suspenso(X)).
matriculado(juan).          % R2
matriculado(luis).          % R3
suspenso(juan).             % R4
```

- Consulta:

```
?- aprobado(X).
X = luis
Yes
```

Árbol de deducción de ?- aprobado(X).



Bibliografía

1. J.A. Alonso (2006) *Introducción a la programación lógica con Prolog.*

   Cap. 0 (Introducción).

2. I. Bratko (1990) *Prolog Programming for Artificial Intelligence (2nd ed.)*

   Cap. 1 (An overview of Prolog) y Cap. 2 (Syntax and meaning of Prolog programs).