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

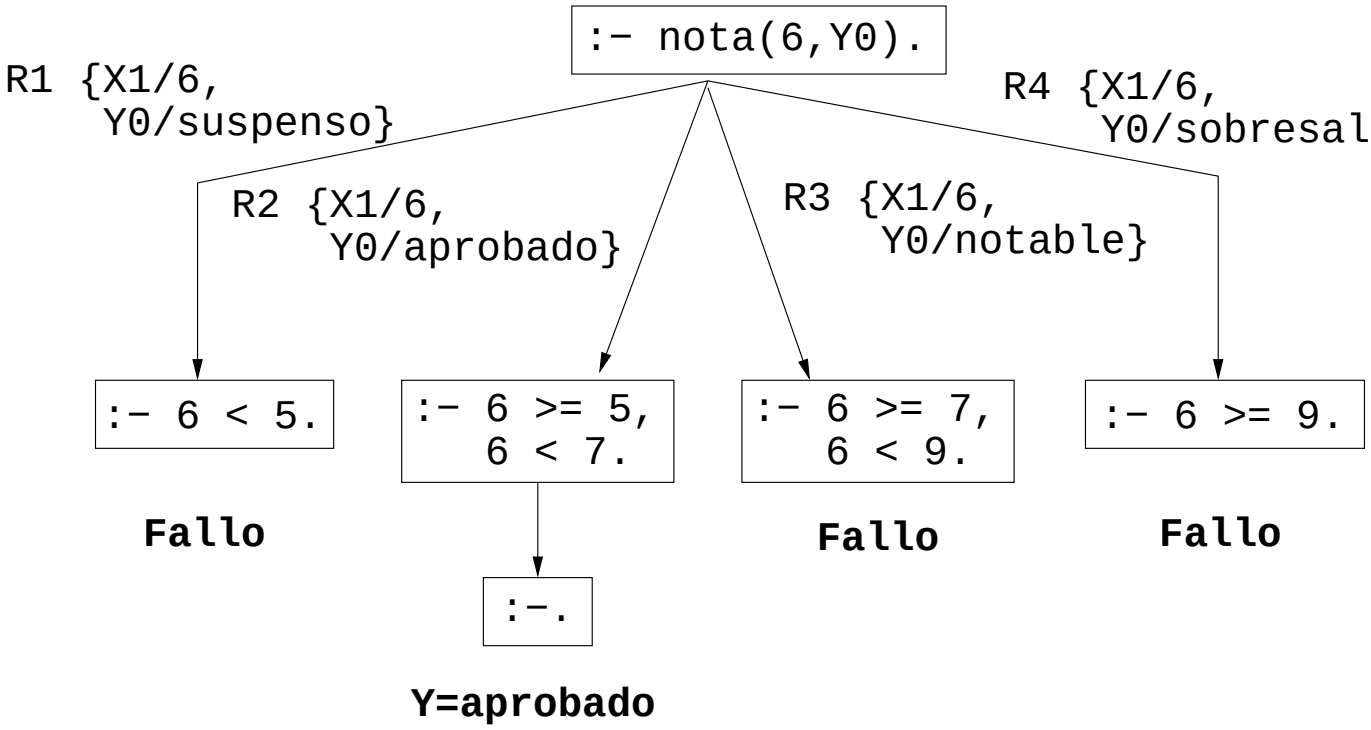
- **Clasificación de notas**
  - **Programa sin corte**

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

  - **Sesión**

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

  - **Árbol de deducción**

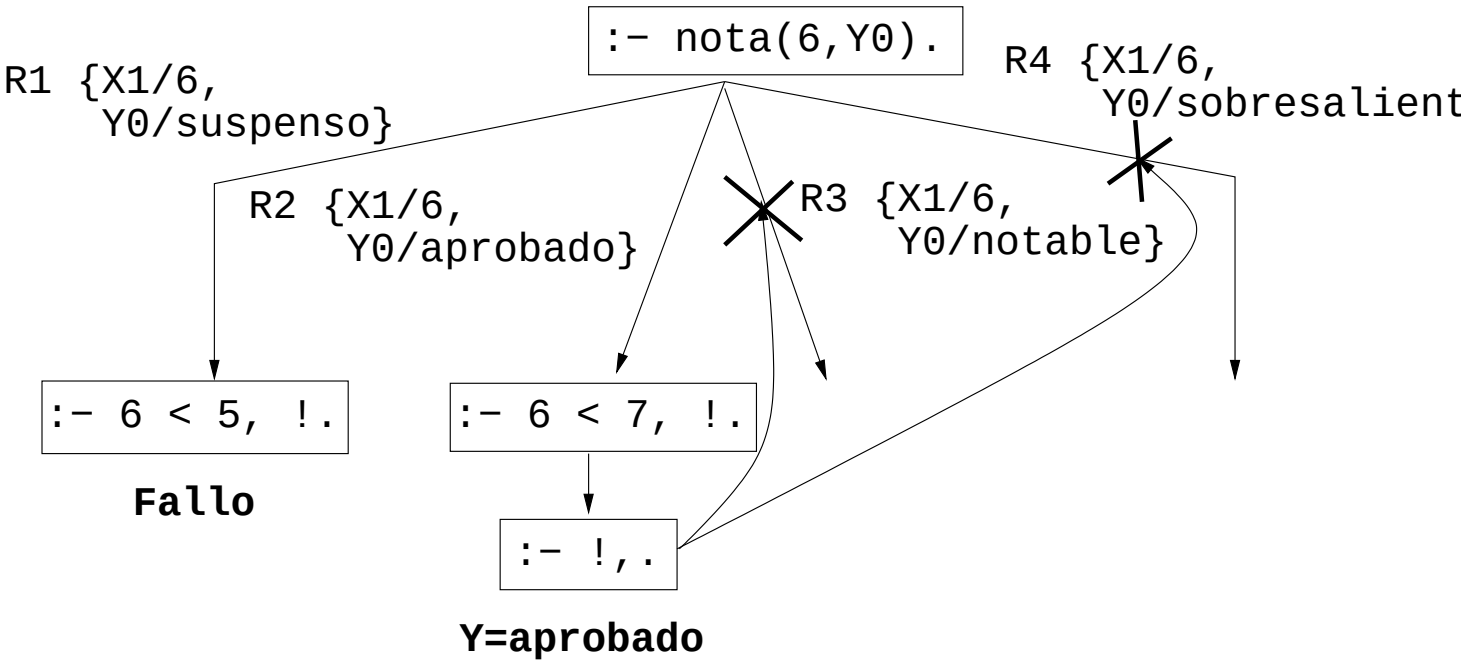


Control del retroceso

- **Clasificación de notas**
  - **Programa con corte**

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

  - **Árbol de deducción**



  - **Cálculo incorrecto:**

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

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]).
```

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

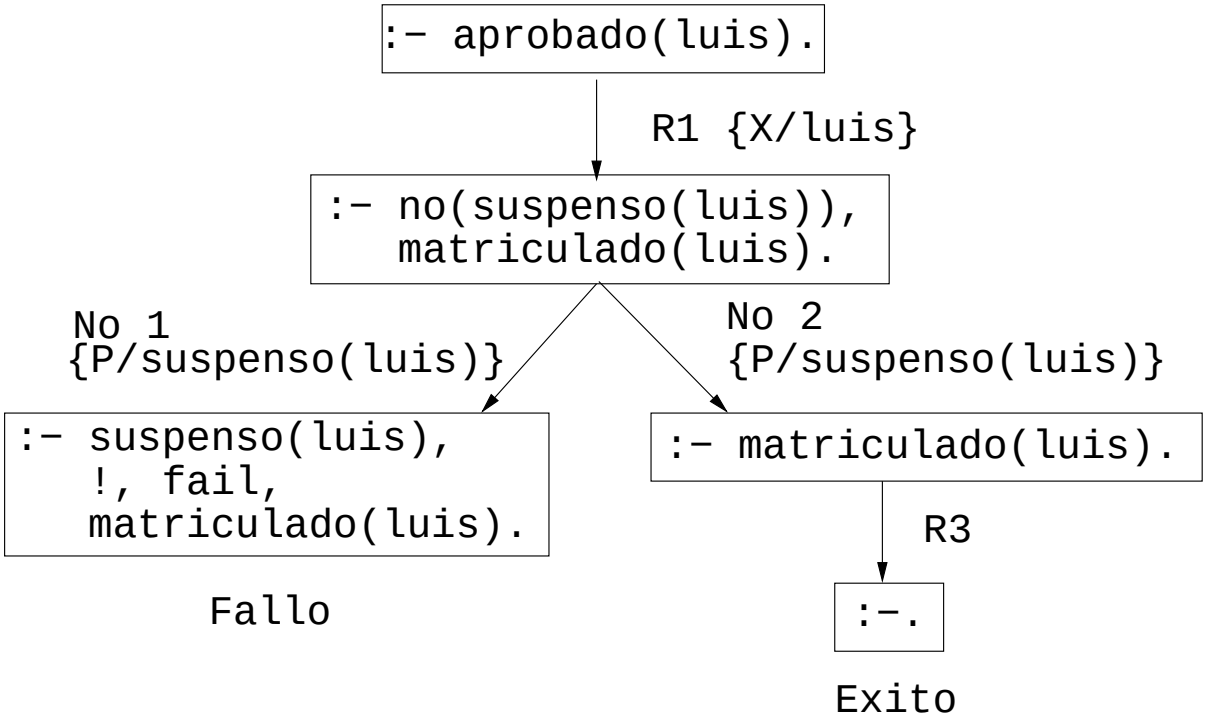
- **Problemas con negación como fallo**
  - **Programa 1**

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

  - **Cálculo**

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

  - **Árbol de deducción**

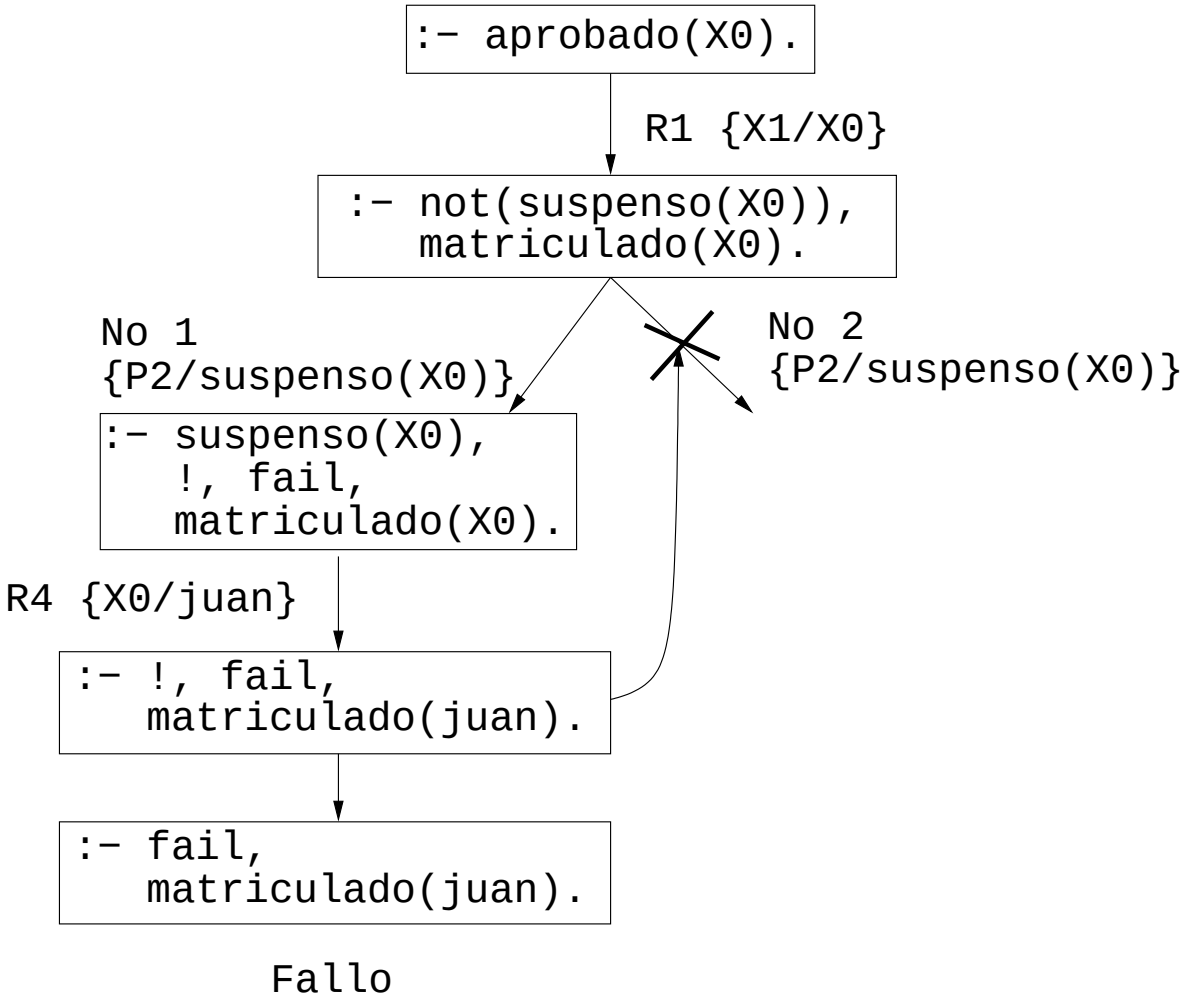


Negación como fallo

- **Cálculo**

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

- **Árbol de deducción**



Negación como fallo

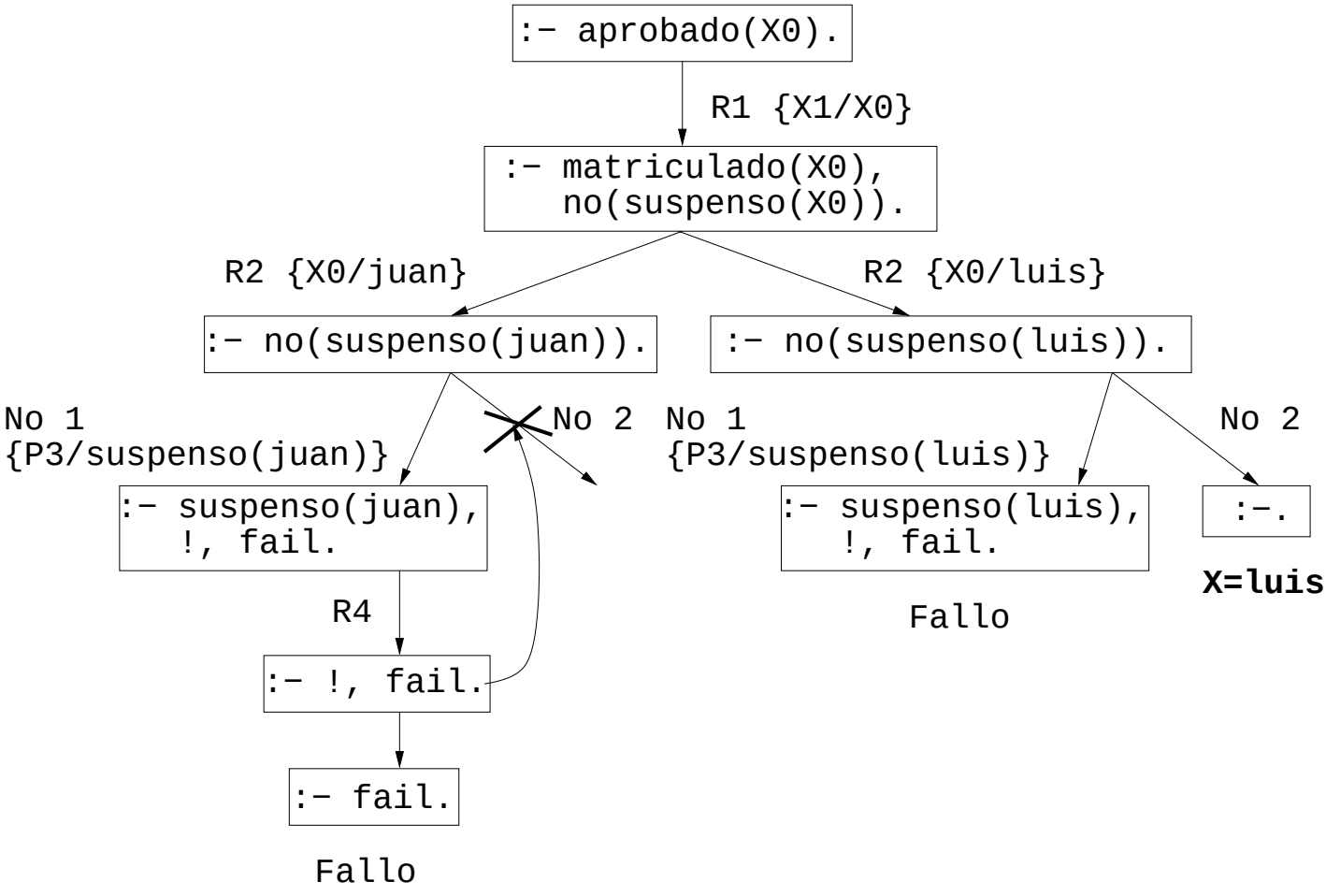
- **Programa 2**

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

- **Cálculo**

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

- **Árbol de deducción**



- **Consejo:** `not` **aplicado a átomos básicos**

El condicional

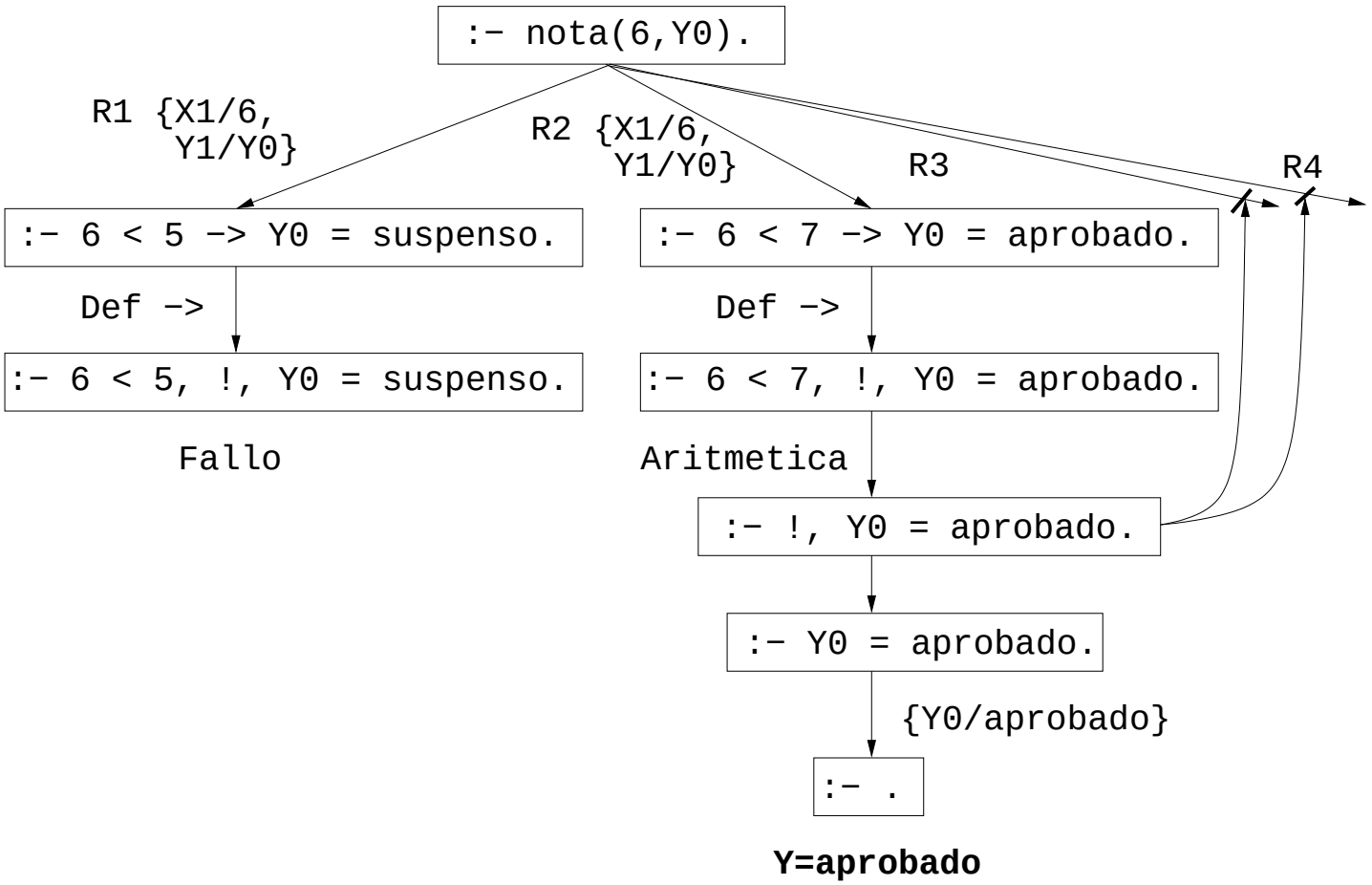
- **Definición del condicional (->) y de `true`**

```
P -> Q :- P, !, Q.                    % Def. ->
true.
```

- **Notas con el condicional**
  - **Definición:**

```
nota(X,Y) :-
   X < 5 -> Y = suspenso ;            % R1
   X < 7 -> Y = aprobado ;            % R2
   X < 9 -> Y = notable ;             % R3
   true  -> Y = sobresaliente.        % R4
```

  - **Árbol de deducción**



Bibliografía

- Alonso, J.A. y Borrego, J. *Deducción automática (Vol. 1: Construcción lógica de sistemas lógicos)* (Ed. Kronos, 2002) (`www.cs.us.es/~jalonso/libros/da1-02.pdf`)
  - Cap. 2: Introducción a la programación lógica con Prolog, pp. 21–29
- Bratko, I. *Prolog Programming for Artificial Intelligence (3 ed.)* (Addison–Wesley, 2001)
  - 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 edititon)* (The MIT Press, 1994)
  - Cap. 11: "Cuts and negation"