Introducción a PROLOG

Carmen Graciani Díaz
José L. Ruiz Reina

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Introducción

- **Lógica como sistema de especificación y lenguaje de programación**
- **Principios**
  - **Programas $\Longrightarrow$ Teorías (Bases de conocimiento)**
  - **Ejecución $\Longrightarrow$ Búsqueda de una prueba (o respuesta) a una consulta (u objetivo)**
  - **Programación $\Longrightarrow$ Modelización**
- **PROLOG $\equiv$ PROgramación LÓGica (PROgramming in LOGic)**
- **Pensar declarativamente**

Pequeño ejemplo: Lógica proposicional

- Base de conocimiento
  - **Regla 1: Si está contento entonces escucha música**
  - **Regla 2: Si tiene radio entonces escucha música**
  - **Regla 3: Si escucha música y tiene una guitarra entonces toca la guitarra**
  - **Hecho 1: Tiene una guitarra**
  - **Hecho 2: Está contento**
- Consulta
  - **¿ Está tocando la guitarra ?**

Pequeño ejemplo: Implementación

- **Programa**

```
escucha_musica :- esta_contento.                          % Regla 1
escucha_musica :- tiene_radio.                            % Regla 2
toca_la_guitarra :- escucha_musica, tiene_guitarra.       % Regla 3

tiene_guitarra.                                           % Hecho 1
esta_contento.                                            % Hecho 2
```

- **Ejecución**

```
Welcome to SWI-Prolog (Multi-threaded, Version 5.2.5)
...
?- [guitarra].
% guitarra compiled 0.00 sec, 1,276 bytes
Yes
?- toca_la_guitarra.
Yes
?-
```

Razonamiento

- Modus ponens: Si el cuerpo de una regla se puede deducir de la base de conocimiento entonces la cabeza se deduce de la base de conocimiento.

- Deducción
  - Recorre la base de conocimiento buscando un hecho o la cabeza de una regla que coincida con la consulta
    - Si coincide con un hecho entonces la consulta es cierta
    - Si coincide con la cabeza de una regla y el cuerpo se deduce de la base de conocimiento entonces la consulta es cierta

Otro ejemplo: Lógica proposicional

- Base de conocimiento
  - **Regla 1: Si Vicente está contento y escucha música entonces toca la guitarra**
  - **Regla 2: Si Beatriz está contenta entonces toca la guitarra**
  - **Regla 3: Si Beatriz escucha música entonces toca la guitarra**
  - **Hecho 1: Vicente está contento**
  - **Hecho 2: Beatriz escucha música**
- Consulta
  - **¿ Está tocando la guitarra Vicente?**
  - **¿ Quién está tocando la guitarra?**

Otro ejemplo: Implementación

- **Programa**

```
toca_la_guitarra(vicente) :- esta_contento(vicente),        % Regla 1
        escucha_musica(vicente).
toca_la_guitarra(beatriz) :- esta_contento(beatriz).        % Regla 2
toca_la_guitarra(beatriz) :- escucha_musica(beatriz).       % Regla 3

esta_contento(vicente).                                     % Hecho 1
escucha_musica(beatriz).                                    % Hecho 2
```

- **Ejecución**

```
?- [guitarra_arg].
% guitarra compiled 0.00 sec, 1,276 bytes
Yes
?- toca_la_guitarra(vicente).
No
?- toca_la_guitarra(X).
X = beatriz ;
No
```

Razonamiento

- **Deducción**
  - **Recorre la base de conocimiento buscando un hecho o la cabeza de una regla tal que, al sustituir la variable de la consulta, coincida**
    - **Si coincide con un hecho entonces la consulta es cierta y la respuesta es la sustitución realizada**
    - **Si coincide con la cabeza de una regla y el cuerpo se deduce de la base de conocimiento entonces la consulta es cierta y la respuesta es la sustitución realizada**

Otro ejemplo: Lógica de primer orden

- Base de conocimiento
  - **Regla 1: Si alguien está contento y escucha música entonces toca la guitarra**
  - **Hecho 1: Vicente está contento**
  - **Hecho 2: Beatriz está contenta**
  - **Hecho 3: Beatriz escucha música**
- Consulta
  - **¿ Está tocando la guitarra Vicente?**
  - **¿ Quién está tocando la guitarra?**

Y otro ejemplo: Implementación

- **Programa**

```
toca_la_guitarra(X) :- esta_contento(X), escucha_musica(X). % Regla 1

esta_contento(vicente).                                     % Hecho 1
esta_contento(beatriz).                                     % Hecho 2
escucha_musica(beatriz).                                    % Hecho 3
```

- **Ejecución**

```
?- [guitarra_var].
% guitarra compiled 0.00 sec, 1,276 bytes
Yes
?- toca_la_guitarra(vicente).
No
?- toca_la_guitarra(X).
X = beatriz ;
No
```

Razonamiento

- **Deducción**
  - **Recorre la base de conocimiento buscando un hecho o la cabeza de una regla tal que, al unificar, coincidan**
    - **Si coincide con un hecho entonces la consulta es cierta y la respuesta es la sustitución realizada**
    - **Si coincide con la cabeza de una regla y el cuerpo se deduce de la base de conocimiento entonces la consulta es cierta y la respuesta es la sustitución realizada**

listing

```
?- [guitarra_var].
% guitarra_var compiled 0.00 sec, 1,068 bytes
Yes
?- listing.
toca_la_guitarra(A) :-
        esta_contento(A),
        escucha_musica(A).

esta_contento(vicente).
esta_contento(beatriz).

escucha_musica(beatriz).
Yes
?- listing(toca_la_guitarra).
toca_la_guitarra(A) :-
        esta_contento(A),
        escucha_musica(A).
Yes
```

listing

```
?- [guitarra].
% guitarra compiled 0.00 sec, 1,152 bytes
Yes
?- listing(toca_la_guitarra).
toca_la_guitarra :-
        escucha_musica,
        tiene_guitarra.

toca_la_guitarra(A) :-
        esta_contento(A),
        escucha_musica(A).
Yes
?- listing(toca_la_guitarra/1).
toca_la_guitarra(A) :-
        esta_contento(A),
        escucha_musica(A).
Yes
```

PROLOG: Sintaxis (I)

- **Términos**
  - **Átomo**
    - **Sucesión de símbolos (mayúsculas o minúsculas) números y/o guión de subrayado empezando por una minúscula:** `toca_la_guitarra`, `libro1`, `a8_4`, ...
    - **Sucesión de caracteres entre comillas simples:** `'Alvaro'`, `' '`, `'toca la guitarra'`, ...
    - **Sucesión de caracteres especiales:** `:-`, `-->`, ...
  - **Número:** `237`, `-984`, `48.9767`, `10E3`, `746E-23`, ...
    - **Los números y átomos se denominan constantes**

PROLOG: Sintaxis (II)

- **Variable: Sucesión de símbolos (mayúsculas o minúsculas) números y/o guión de subrayado empezando por una mayúscula o guión de subrayado:** `Beatriz`, `X`, `_libro1`, `_`, `...`

- **Término compuesto:** `<functor>(<argumentos>+)`, **siendo** `<functor>` **un átomo y los** `<argumentos>` **son términos.** `toca_la_guitarra(vicente)`, `h(r(p(X, Y), p(Z, Y)))`, `...`
  Aridad: número de argumentos.

Unificación

- **Dos términos unifican:**
  - **Son iguales**
  - **Contienen variables y estas pueden ser sustituidas por términos de tal forma que los términos resultantes son iguales**
- **Algoritmo de unificación:**
  - **1.- Los términos son constantes y coinciden**
  - **2.- Si uno de los términos es una variable se sustituye por el otro término (previamente las variables que aparecen se sustituyen todas por variables nuevas).**
  - **3.- Si son dos términos compuestos:**
    - **Los functores y la aridad coinciden**
    - **Cada par de argumentos (por orden, uno de cada término) unifica y las sustituciones encontradas son compatibles**
  - **4.- Dos términos unifican si y sólo si se cumple alguna de las tres condiciones anteriores.**

Unificación =

```
?- a = a.
Yes
?- a = b.
No
?- 2 = 1+1.
No
?- X = toca_la_guitarra(Y).
X = toca_la_guitarra(_G147)
Y = _G147
Yes
?- X = Y.
X = _G147
Y = _G147
Yes
```

Unificación =

```
?- padre(X, Y) = padre(hijo(pepe), maria).
X = hijo(pepe)
Y = maria
Yes
?- padre(X, maria) = padre(hijo(pepe), Y).
X = hijo(pepe)
Y = maria
Yes
?- padre(X, maria) = padre(hijo(pepe), X).
No
?- padre(X) = X.
X = padre(padre(padre(padre(padre(padre(padre(padre(padre(padre(...))))))))))
Yes
```

Unificación

- **Base de conocimiento**

```
horizontal(recta(pto(X, Y), pto(Z, Y))).
vertical(recta(pto(X, Y), pto(X, Z))).
```

- **Consulta**

```
?- vertical(recta(pto(3, 2), pto(3, 1))).
Yes
?- vertical(recta(pto(3, 2), pto(X, 1))).
X = 3
Yes
?- vertical(recta(pto(3, 2), pto(X, Y))).
X = 3
Y = _G151
Yes
?- horizontal(recta(pto(3, 2), Z)).
Z = pto(_G207, 2)
Yes
```

SLD-Resolución

- **Base de conocimiento**

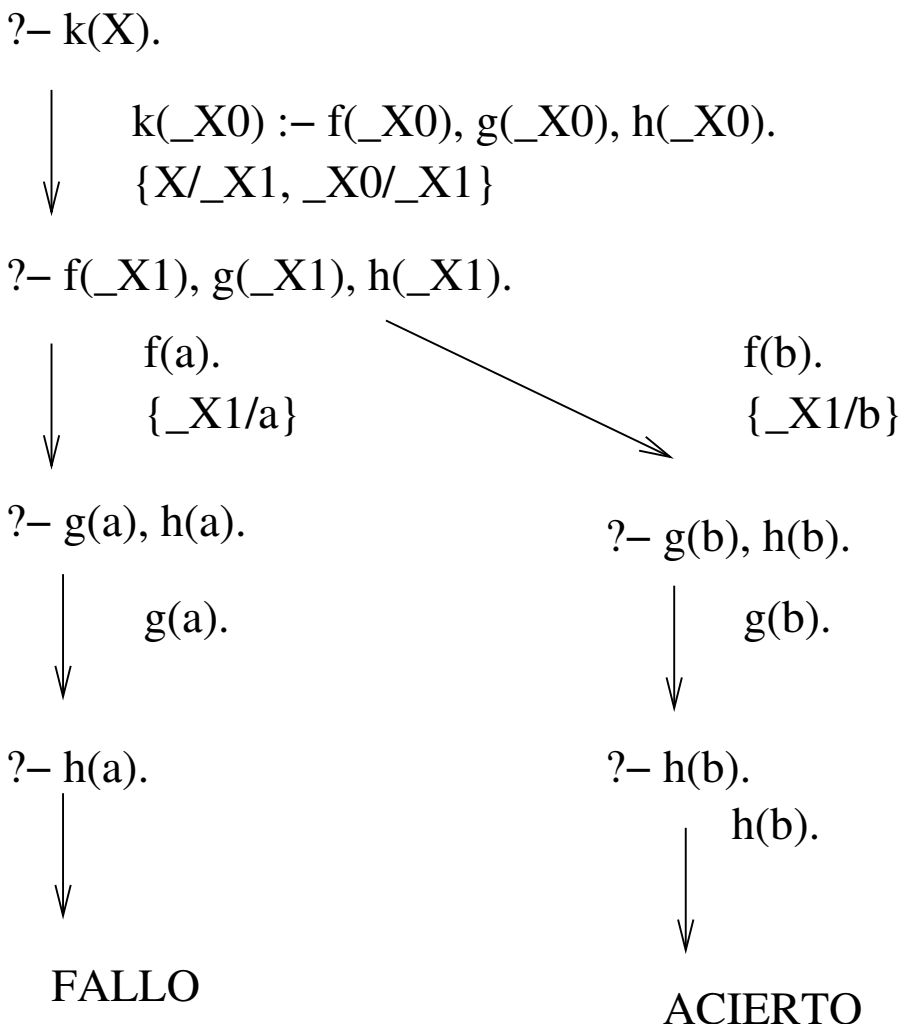
```
f(a). f(b).
g(a). g(b).
h(b).

k(X) :- f(X), g(X), h(X).
```

- **Consulta**

```
?- k(X).
X = b
Yes
```

SLD-Resolución



SLD-Resolución `trace`

```
[trace]  ?- k(X).
   Call: (7) k(_G273) ?
   Call: (8) f(_G273) ?
   Exit: (8) f(a) ?
   Call: (8) g(a) ?
   Exit: (8) g(a) ?
   Call: (8) h(a) ?
   Fail: (8) h(a) ?
   Fail: (8) g(a) ?
   Redo: (8) f(_G273) ?
   Exit: (8) f(b) ?
   Call: (8) g(b) ?
   Exit: (8) g(b) ?
   Call: (8) h(b) ?
   Exit: (8) h(b) ?
   Exit: (7) k(b) ?
X = b
Yes
```

Recursión

- **Base de conocimiento**

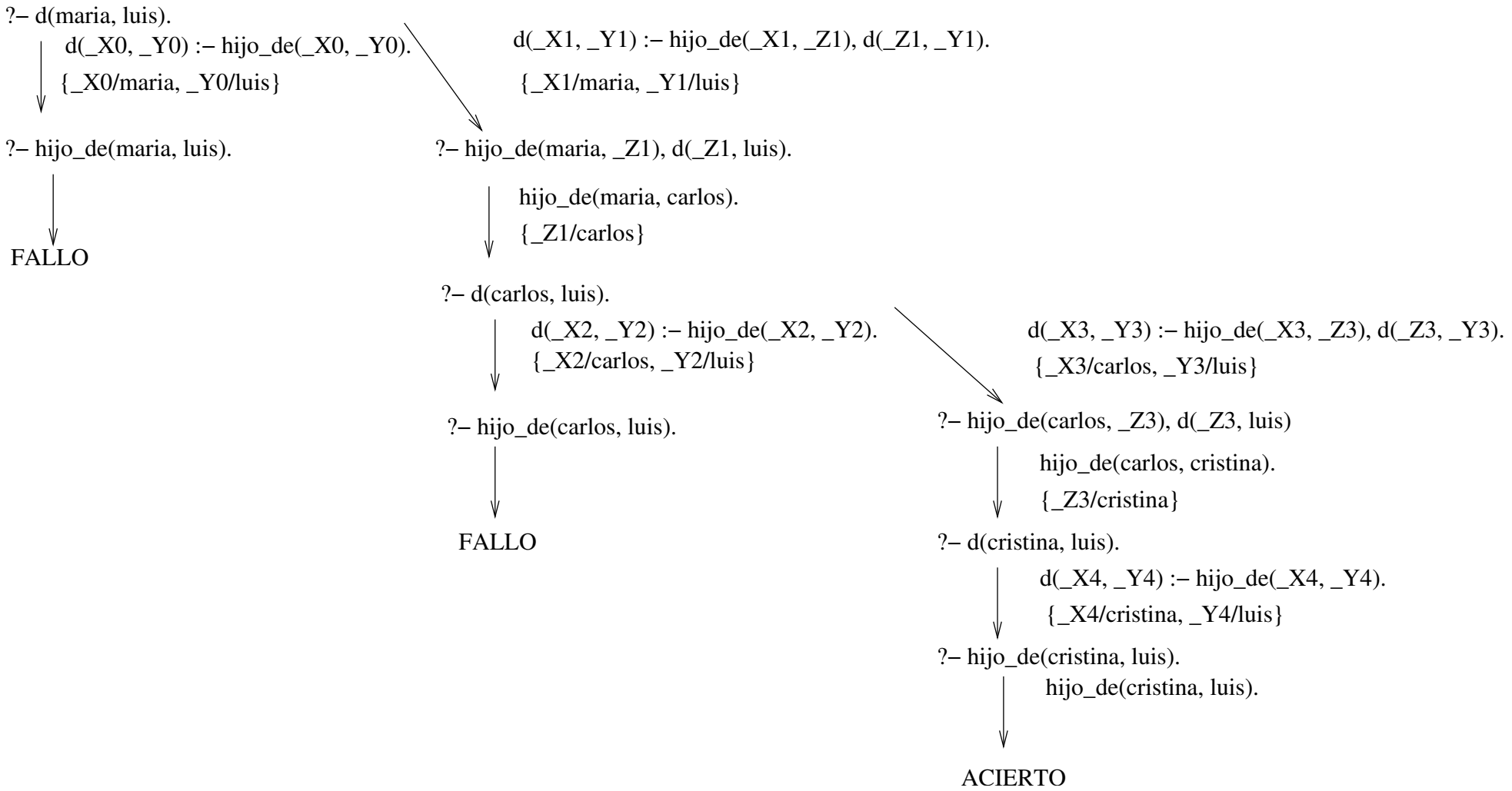
```
hijo_de(maria, carlos).
hijo_de(carlos, cristina).
hijo_de(cristina, luis).

descendiente(X, Y) :- hijo_de(X, Y).
descendiente(X, Y) :- hijo_de(X, Z), descendiente(Z, Y).
```

- **Consulta**

```
?- descendiente(maria, luis).
Yes
```

Recursión


?– d(maria, luis).
d(_X0, _Y0) :– hijo_de(_X0, _Y0).
{_X0/maria, _Y0/luis}
?– hijo_de(maria, luis).
FALLO
d(_X1, _Y1) :– hijo_de(_X1, _Z1), d(_Z1, _Y1).
{_X1/maria, _Y1/luis}
?– hijo_de(maria, _Z1), d(_Z1, luis).
hijo_de(maria, carlos).
{_Z1/carlos}
?– d(carlos, luis).
d(_X2, _Y2) :– hijo_de(_X2, _Y2).
{_X2/carlos, _Y2/luis}
?– hijo_de(carlos, luis).
FALLO
d(_X3, _Y3) :– hijo_de(_X3, _Z3), d(_Z3, _Y3).
{_X3/carlos, _Y3/luis}
?– hijo_de(carlos, _Z3), d(_Z3, luis)
hijo_de(carlos, cristina).
{_Z3/cristina}
?– d(cristina, luis).
d(_X4, _Y4) :– hijo_de(_X4, _Y4).
{_X4/cristina, _Y4/luis}
?– hijo_de(cristina, luis).
hijo_de(cristina, luis).
ACIERTO

Recursión

```
[trace]  ?- descendiente(maria, luis).
   Call: (7) descendiente(maria, luis) ?
   Call: (8) hijo_de(maria, luis) ?
   Fail: (8) hijo_de(maria, luis) ?
   Redo: (7) descendiente(maria, luis) ?
   Call: (8) hijo_de(maria, _G313) ?
   Exit: (8) hijo_de(maria, carlos) ?
   Call: (8) descendiente(carlos, luis) ?
   Call: (9) hijo_de(carlos, luis) ?
   Fail: (9) hijo_de(carlos, luis) ?
   Redo: (8) descendiente(carlos, luis) ?
   Call: (9) hijo_de(carlos, _G313) ?
   Exit: (9) hijo_de(carlos, cristina) ?
   Call: (9) descendiente(cristina, luis) ?
   Call: (10) hijo_de(cristina, luis) ?
   Exit: (10) hijo_de(cristina, luis) ?
   Exit: (9) descendiente(cristina, luis) ?
   Exit: (8) descendiente(carlos, luis) ?
   Exit: (7) descendiente(maria, luis) ?
Yes
```

Recursión

- **Base de conocimiento**

```
hijo_de(maria, carlos).
hijo_de(carlos, cristina).
hijo_de(cristina, luis).

descendiente(X, Y) :- descendiente(Z, Y), hijo_de(X, Z).
descendiente(X, Y) :- hijo_de(X, Y).
```

- **Consulta**

```
?- descendiente(maria, luis).
ERROR: Out of local stack
   Exception: (27,930) descendiente(_G191, luis) ? a
% Execution Aborted
```

Recursión

```
?- trace.
Yes
[trace]  ?- descendiente(maria, luis).
   Call: (7) descendiente(maria, luis) ?
   Call: (8) descendiente(_G312, luis) ?
   Call: (9) descendiente(_G312, luis) ?
   Call: (10) descendiente(_G312, luis) ?
   Call: (11) descendiente(_G312, luis) ?
   Call: (12) descendiente(_G312, luis) ?
   Call: (13) descendiente(_G312, luis) ?
   Call: (14) descendiente(_G312, luis) ?
   Call: (15) descendiente(_G312, luis) ?
   Call: (16) descendiente(_G312, luis) ?
   Call: (17) descendiente(_G312, luis) ?
   Call: (18) descendiente(_G312, luis) ?
   Call: (19) descendiente(_G312, luis) ?
   Call: (20) descendiente(_G312, luis) ?
   Call: (21) descendiente(_G312, luis) ?
   Call: (22) descendiente(_G312, luis) ?
   Call: (23) descendiente(_G312, luis) ?
   Call: (24) descendiente(_G312, luis) ?
```

Listas

- **Términos**
  - **Lista vacía:** `[]`
  - **Lista compuesta:** `[<término>+{| <lista>}]`
    `<términos>`**: sucesión de los primeros elementos de la lista**
    `<lista>`**: lista con los restantes elementos**
    `[a, b, c] = [a, b, c| []] = [a, b| [c]] = [a|[b, c]]`
- **Algunas relaciones que trabajan con listas**
  - `append(L1, L2, L3) :-` **La lista** `L3` **unifica con la concatenación de las listas** `L1` **y** `L2`
  - `member(E, L) :-` `E` **unifica con alguno de los elementos de la lista** `L`
  - `reverse(L1, L2) :-` **La inversa de la lista** `L1` **unifica con la lista** `L2`

Listas

```
listing(append/3).
lists:append([], A, A).
lists:append([A|B], C, [A|D]) :-
        append(B, C, D).
Yes
?- listing(member).
lists:member(A, [A|B]).
lists:member(A, [B|C]) :-
        member(A, C).
Yes
?- listing(reverse).
lists:reverse(A, B) :-
        reverse(A, [], B, B).

lists:reverse([], A, A, []).
lists:reverse([A|B], C, D, [E|F]) :-
        reverse(B, [A|C], D, F).
Yes
?-
```

Aritmética `is`

```
?- 2 is 1 + 1.
Yes
?- 2 is 6 / 3.
Yes
?- X is mod(40, 13).
X = 1
Yes
?- X is 9 ** 2.
X = 81
Yes
?- 12 is 6 * X.
ERROR: Arguments are not sufficiently instantiated
```

- **Comparaciones:** `<`(menor que), `=<`(menor o igual que), `=:=` (igualdad numérica), `=\=` (desigualdad numérica), `>=` (mayor o igual que), `>`(mayor que)
  Son todos infijos
- **Unificación:** `=`, `\=`

Aritmética

- **Base de conocimiento**

```
sumar_3_y_duplicar(X, Y) :- Y is (X + 3) * 2.
```

- **Consulta**

```
?- sumar_3_y_duplicar(2, Y).
Y = 10
Yes
?- sumar_3_y_duplicar(Y, 2).
ERROR: Arguments are not sufficiently instantiated
^  Exception: (8) 2 is (_G147+3)*2 ?
```

Acumuladores

- **Base de conocimiento**

```
longitud([], 0).
longitud([E|L], N) :- longitud(L, LN), N is LN + 1.

longitud_bis(L, N) :- longitud_ac(L, 0, N).

longitud_ac([], A, A).
longitud_ac([E| L], A, N) :- LA is A + 1, longitud_ac(L, LA, N).
```

- **Consultas**

```
?- longitud([a, b, c], N).
N = 3
Yes
?- longitud_bis([a, b, c], N).
N = 3
Yes
```

Acumuladores

```
?- trace, longitud([a, b, c], N).
   Call: (8) longitud([a, b, c], _G157) ?
   Call: (9) longitud([b, c], _G244) ?
   Call: (10) longitud([c], _G244) ?
   Call: (11) longitud([], _G244) ?
   Exit: (11) longitud([], 0) ?
^  Call: (11) _G249 is 0+1 ?
^  Exit: (11) 1 is 0+1 ?
   Exit: (10) longitud([c], 1) ?
^  Call: (10) _G252 is 1+1 ?
^  Exit: (10) 2 is 1+1 ?
   Exit: (9) longitud([b, c], 2) ?
^  Call: (9) _G157 is 2+1 ?
^  Exit: (9) 3 is 2+1 ?
   Exit: (8) longitud([a, b, c], 3) ?
N = 3
Yes
```

Acumuladores

```
?- trace, longitud_bis([a, b, c], N).
   Call: (8) longitud_bis([a, b, c], _G283) ?
   Call: (9) longitud_ac([a, b, c], 0, _G283) ?
^  Call: (10) _G369 is 0+1 ?
^  Exit: (10) 1 is 0+1 ?
   Call: (10) longitud_ac([b, c], 1, _G283) ?
^  Call: (11) _G372 is 1+1 ?
^  Exit: (11) 2 is 1+1 ?
   Call: (11) longitud_ac([c], 2, _G283) ?
^  Call: (12) _G375 is 2+1 ?
^  Exit: (12) 3 is 2+1 ?
   Call: (12) longitud_ac([], 3, _G283) ?
   Exit: (12) longitud_ac([], 3, 3) ?
   Exit: (11) longitud_ac([c], 2, 3) ?
   Exit: (10) longitud_ac([b, c], 1, 3) ?
   Exit: (9) longitud_ac([a, b, c], 0, 3) ?
   Exit: (8) longitud_bis([a, b, c], 3) ?
N = 3
Yes
```

Corte

- **Base de conocimiento**

```
s(X, Y) :- q(X, Y).
s(0, 0).
q(X, Y) :- i(X), j(Y).
i(1). i(2).
j(1). j(3).
```

- **Consulta**

```
?- s(X, Y).
X = 1
Y = 1 ;
X = 1
Y = 3 ;
X = 2
Y = 1 ;
X = 2
Y = 3 ;
X = 0
Y = 0 ;
No
```

Corte

- **Base de conocimiento**

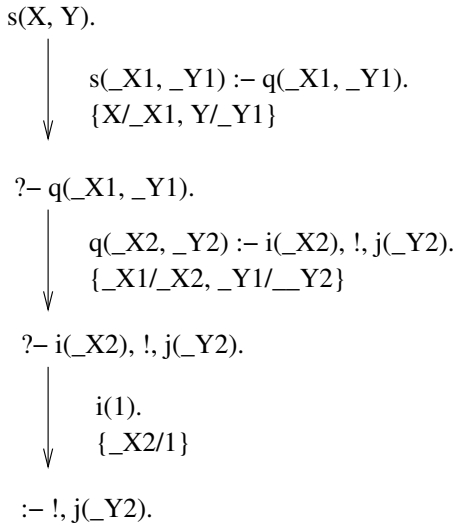
```
s(X, Y) :- q(X, Y).
s(0, 0).
q(X, Y) :- i(X), !, j(Y).
i(1). i(2).
j(1). j(3).
```

- **Consulta**

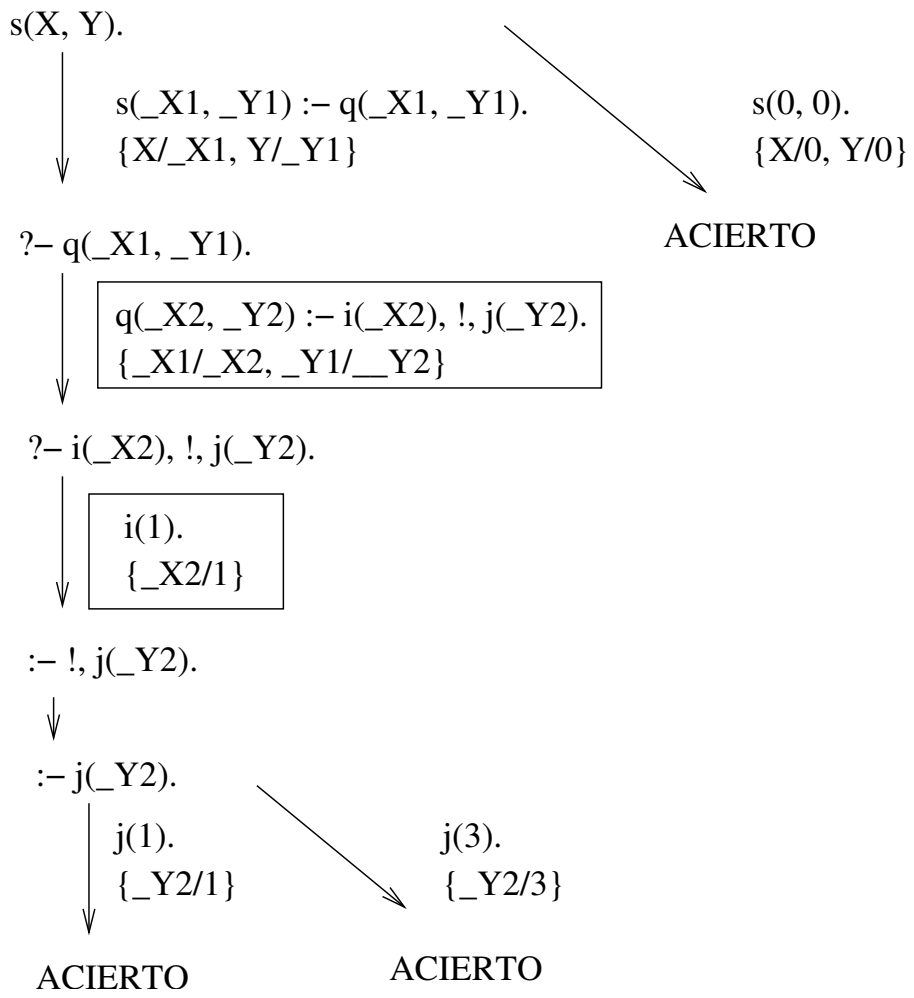
```
?- s(X, Y).
X = 1
Y = 1 ;
X = 1
Y = 3 ;
X = 0
Y = 0 ;
No
```

Corte

- **Resolución (control de la ejecución)**
  - **Si a lo largo de la resolución no se alcanza el lugar del corte, se procede como hasta ahora**
  - **Si se alcanza el corte, se marca el predicado que define la regla, las condiciones de la regla anteriores al corte y la sustitución obtenida hasta el momento. A lo largo del resto de la resolución no se permite una nueva resolución de dicho predicado, ni de las condiciones, ni de las sustituciones obtenidas**



Corte


s(X, Y).
s(_X1, _Y1) :– q(_X1, _Y1).
{X/_X1, Y/_Y1}
s(0, 0).
{X/0, Y/0}
ACIERTO
?– q(_X1, _Y1).
q(_X2, _Y2) :– i(_X2), !, j(_Y2).
{_X1/_X2, _Y1/__Y2}
?– i(_X2), !, j(_Y2).
i(1).
{_X2/1}
:– !, j(_Y2).
:– j(_Y2).
j(1).
{_Y2/1}
j(3).
{_Y2/3}
ACIERTO
ACIERTO

Corte

- **Base de conocimiento**

```
max(X, Y, Y) :- X =< Y.
max(X, Y, X) :- X > Y.
```

- **Consulta**

```
?- max(2, 5, X).
X = 5
Yes
?- max(2, 5, 2).
No
```

Corte

- **Base de conocimiento**

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

- **Consulta**

```
?- max(2, 5, X).
X = 5
Yes
?- max(2, 5, 2).
Yes
```

Algunos predefinidos

- `display(E)`: **muestra el término PROLOG**

```
?- display([a, b]).
.(a, .(b, []))
Yes
?- display(1+1).
+(1, 1)
Yes
```

- `write(E)`: **escribe el término por pantalla**

```
?- write([a,b]).
[a, b]
Yes
?- write(1+1).
1+1
Yes
```

Algunos predefinidos

- `tab(N)`: **escribe N espacios en blanco**
- `nl`: **escribe un salto de línea**
- `read(E)`: **pide al usuario que escriba un término, falla si éste no unifica con el argumento**

```
?- read(X).
|    a.
X = a
Yes
?- read(p(X)).
|    a.
No
```

Algunos predefinidos

- `fail`: **Siempre falla**
- `assert(C)`, `retract(C)`: **permiten añadir o eliminar cláusulas dinámicamente de la base de conocimiento. (una regla o un hecho)**
- `retractall(S)`: **elimina todas las cláusulas que definan la relación** `S`
- `not(E)`, `\+ E`: **Falla si se puede resolver la consulta** `?- E.`, **tiene éxito si esta consulta falla**

Algunos predefinidos

- `findall(E, O, L)`, `bagof(E, O, L)`, `setof(E, O, L)`

```
?- findall(X, s(X, _), L), write(L).
[0, 0, 2, 2, 0]
X = _G147
L = [0, 0, 2, 2, 0]
Yes
?- bagof(X, s(X, Y), L).
X = _G147
Y = 3
L = [0, 2] ;
X = _G147
Y = 0
L = [0, 2, 0] ;
No
?- setof(X, s(X, Y), L).
X = _G147
Y = 3
L = [0, 2] ;
X = _G147
Y = 0
L = [0, 2] ;
No
```