Tema 4: Programación lógica y Prolog

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Resolución SLD

- **Ejemplo de resolución SLD:**
  - **Programa:**

```
nieto(X,Z) :- hijo(X,Y), hijo(Y,Z).
hijo(X,Y) :- madre(Y,X).
hijo(X,Y) :- padre(Y,X).
padre(a,b).
madre(b,c).
```

  - **Pregunta:**

```
:- nieto(c,X).
```

  - **Resolución SLD:**

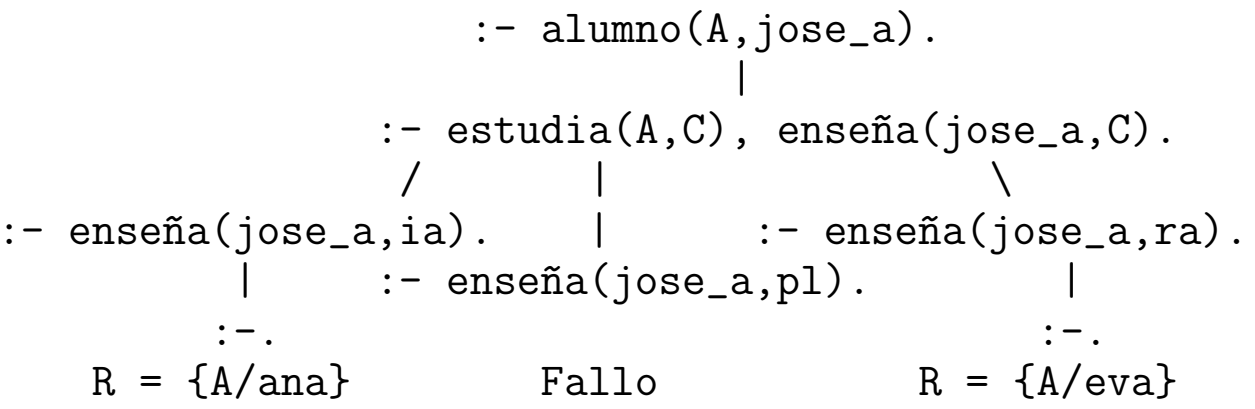
```
:- nieto(c,X0).
|    nieto(X1,Z1) :- hijo(X1,Y1), hijo(Y1,Z1).
|    {X1/c, Z1/X0}
:- hijo(c,Y1), hijo(Y1, X0).
|    hijo(X2,Y2) :- madre(Y2,X2).
|    {X2/c, Y2/Y1}
:- madre(Y1,c), hijo(Y1,X0).
|    madre(b,c).
|    {Y1/b}
:- hijo(c,X0).
|    hijo(X3,Y3) :- padre(Y3,X3).
|    {X3/b, Y3/X0}
:- padre(X0,b).
|    padre(a,b).
|    {X0/a}
:-.
```

  - **Respuesta:** {X/a}

Resolución SLD

- **Siglas SLD:**
  - **S: regla de Selección**
  - **L: resolución Lineal**
  - **D: cláusulas Definidas**
- **Reglas de selección de Prolog:**
  - **Selección de objetivos: De izquierda a derecha**
  - **Selección de reglas: Como en el programa**
- **Arbol SLD:**

```
alumno(A,P) :- estudia(A,C), enseña(P,C).
estudia(ana,ia).
estudia(ana,pl).
estudia(eva,ra).
enseña(jose_a,ia).
enseña(jose_a,ra).
enseña(rafael,pl).
```



Resolución SLD

- **Sesión**

```
?- alumno(A,jose_a).
A = ana ;
A = eva ;
No

?- trace.
Yes

?- alumno(A,jose_a).
   Call:  (  7) alumno(_G178, jose_a) ?
   Call:  (  8) estudia(_G178, _L130) ?
   Exit:  (  8) estudia(ana, ia) ?
   Call:  (  8) enseña(jose_a, ia) ?
   Exit:  (  8) enseña(jose_a, ia) ?
   Exit:  (  7) alumno(ana, jose_a) ?
A = ana ;
   Redo:  (  8) enseña(jose_a, ia) ?
   Fail:  (  8) enseña(jose_a, ia) ?
   Redo:  (  8) estudia(_G178, _L130) ?
   Exit:  (  8) estudia(ana, pl) ?
   Call:  (  8) enseña(jose_a, pl) ?
   Fail:  (  8) enseña(jose_a, pl) ?
   Redo:  (  8) estudia(_G178, _L130) ?
   Exit:  (  8) estudia(eva, ra) ?
   Call:  (  8) enseña(jose_a, ra) ?
   Exit:  (  8) enseña(jose_a, ra) ?
   Exit:  (  7) alumno(eva, jose_a) ?
A = eva ;
No
```

Resolución SLD

- **Notas:**
  - **Ramas de éxito y de fallo**
  - **Búsqueda en profundidad**
- **Ramas infinitas en Prolog por predicados simétricos:**
  - **Programa** `ej-1.pl`

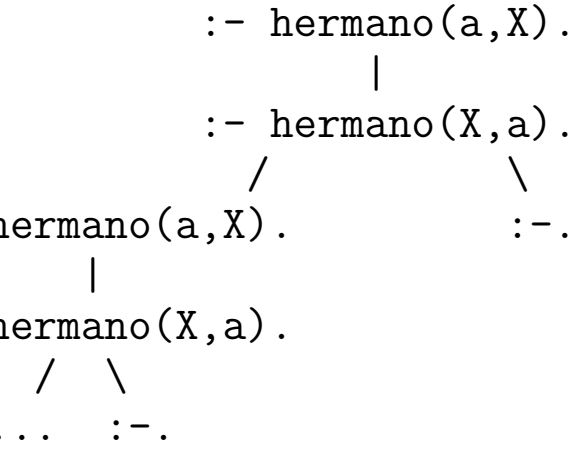
```
hermano(X,Y) :- hermano(Y,X).
hermano(b,a).
```

  - **Sesión:**

```
?- [ej-1].
Yes

?- hermano(a,X).
[WARNING: Out of local stack]
Execution Aborted
```

  - **Arbol:**



Resolución SLD

- **Programa** `ej-2.pl`

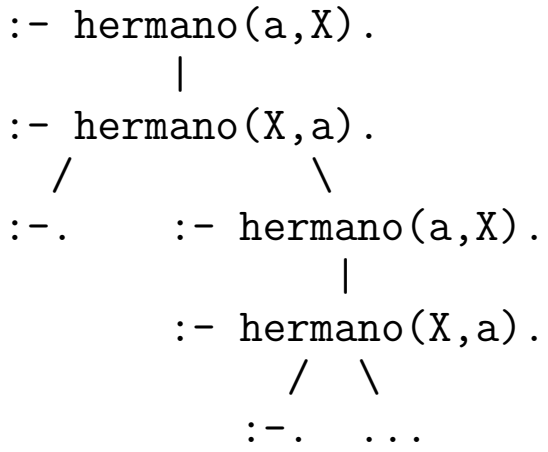
```
hermano(b,a).
hermano(X,Y) :- hermano(Y,X).
```

- **Sesión:**

```
?- [ej-2].
Yes

?- hermano(a,X).
X = b ;
X = b ;
Yes
```

- **Arbol:**



Resolución SLD

- **Ramas infinitas en Prolog por predicados transitivos:**
  - **Programa**

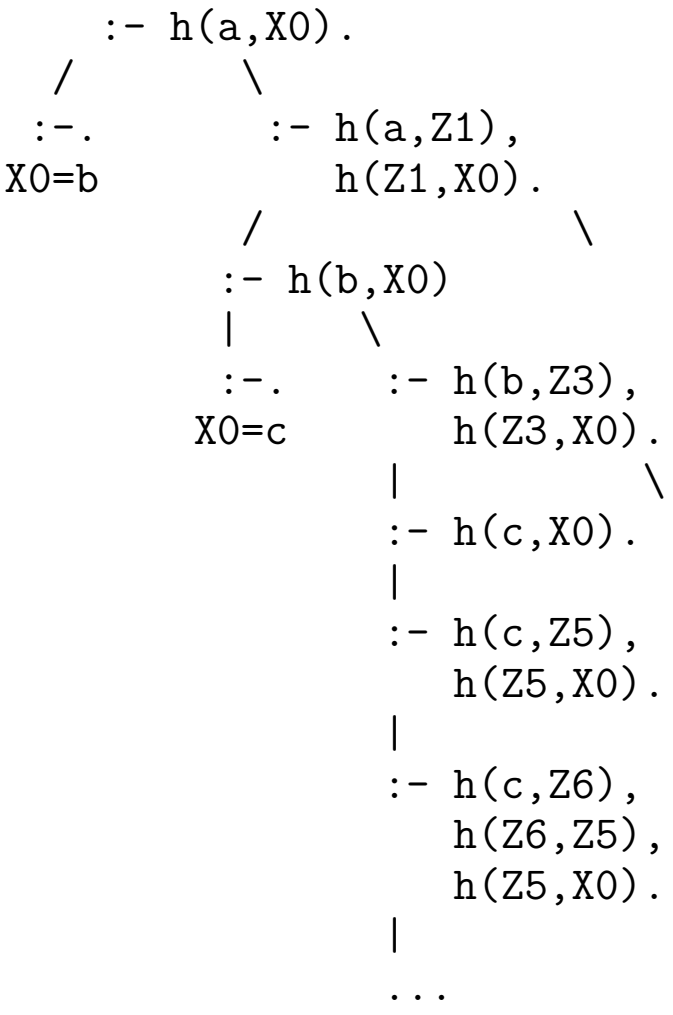
```
hermano(a,b).
hermano(b,c).
hermano(X,Y) :-
   hermano(X,Z),
   hermano(Z,Y).
```

  - **Pregunta**

```
?- hermano(a,X).
X = b ;
X = c ;
[WARNING: Out of local stack]
   Exception: (42407) hermano(c, _L508920) ? a
Execution Aborted
```

Resolución SLD

- **Arbol SLD**



Todas las soluciones

- `bagof(Var,Condicion,Lista)`
  - **Programa**

```
edad(pedro,7).
edad(ana,5).
edad(juan,8).
edad(maria,5).
```

  - **Sesión**

```
?- bagof(_X,edad(_X,5),L).
L = [ana, maria] ;
No

?- bagof(_X,edad(_X,Edad),L).
Edad = 7
L = [pedro] ;
Edad = 8
L = [juan] ;
Edad = 5
L = [ana, maria] ;
No

?- bagof(_X,_E^edad(_X,_E),L).
L = [pedro, ana, juan, maria] ;
No

?- bagof(_X,edad(_X,6),L).
No
```

Todas las soluciones

- `setof(Var,Condicion,Lista)`
  - **Similar a `bagof` salvo que `Lista_de_X` está ordenada y sin elementos duplicados.**
  - **Sesión**

```
?- setof(_X,_E^edad(_X,_E),Lista_niños),
   setof(_E,_X^edad(_X,_E),Lista_edades).
Lista_niños = [ana, juan, maria, pedro]
Lista_edades = [5, 7, 8] ;
No

?- setof(_E-_X,_E^edad(_X,_E),L).
L = [5-ana, 5-maria, 7-pedro, 8-juan] ;
No

?- setof(_E,(edad(ana,_E),edad(maria,_E)),L).
L = [5] ;
No

?- setof(_X,edad(_X,6),L).
No
```

Todas las soluciones

- `findall(Var,Condicion,Lista)`
  - **Similar a `bagof` pero su comportamiento cambia cuando en la condición hay variables libres o si no hay soluciones.**
  - **Sesión**

```
?- findall(_X,edad(_X,_E),L).
L = [pedro, ana, juan, maria] ;
No

?- findall(_X,edad(_X,6),L).
L = [] ;
No
```

Predicados de segundo orden

- **Programa:**

```
padre(andres,beatriz).
padre(carlos,david).
padre(ernesto,elisa).

madre(maria,beatriz).
madre(eva,david).
madre(carmen,elisa).

padres([],[]).
padres([H|R1],[X|R2]) :-
   padre(X,H),
   padres(R1,R2).

madres([],[]).
madres([H|R1],[X|R2]):-
   madre(X,H),
   madres(R1,R2).
```

- **Sesión**

```
?- padres([beatriz,david,elisa],L).
L = [andres, carlos, ernesto] ;
No
?- madres([beatriz,david,elisa],L).
L = [maria, eva, carmen] ;
No
```

Predicados de segundo orden

- **"Abstracción":**
  - **Sesión**

```
?-  map(padre,[beatriz,david,elisa],L).
L = [andres, eva, carmen] ;
No
?- map(padre,[beatriz,david,elisa],L).
L = [andres, eva, carmen]
Yes
?- map(madre,[beatriz,david,elisa],L).
L = [maria, eva, carmen]
Yes
```

  - **Intento de definición**

```
map(Relacion,[],[]).
map(Relacion,[H|R1],[X|R2]):-
   Relacion(X,H),
   map(R1,R2).
```

  - **Prolog no admiten** `Relacion/2`
  - **El predicado** `univ` `=..`

```
?- padre(carlos,david)=.. L.
L = [padre, carlos, david] ;
No
?- Lit =.. [padre,carlos,david].
Lit = padre(carlos, david)
Yes
```

Predicados de segundo orden

- **Definición de `map`**

```
map(_,[],[]).
map(Relacion,[H|R1],[X|R2]):-
   Lit =.. [Relacion,X,H],
   Lit,
   map(Relacion,R1,R2).
```

- **El predicado de segundo orden `call`**

```
?- call(padre,X,beatriz).
X = andres
Yes
?- call(madre,X,beatriz).
X = maria
Yes
```

- **Definición de `map` con `call`**

```
map(_,[],[]).
map(Relacion,[H|R1],[X|R2]):-
   call(Relacion,X,H),
   map(Relacion,R1,R2).
```

- **El predicado predefinido `maplist`**

```
?- maplist(reverse,[[a,b],[1,2,3]],L).
L = [[b,a],[3,2,1]]
Yes
```

Poda de la búsqueda mediante corte

- **Ejemplo** `nota`
  - **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.
```

  - **Traza**

```
?- trace.
Yes

?- nota(3,Y).
   Call:  (  7) nota(3, _G98) ?
   Call:  (  8) 3 < 5 ?
   Exit:  (  8) 3 < 5 ?
   Exit:  (  7) nota(3, suspenso) ?

Y = suspenso ;
   Redo:  (  7) nota(3, _G98) ?
   Call:  (  8) 3 >= 5 ?
   Fail:  (  8) 3 >= 5 ?
   Redo:  (  7) nota(3, _G98) ?
   Call:  (  8) 3 >= 7 ?
   Fail:  (  8) 3 >= 7 ?
   Redo:  (  7) nota(3, _G98) ?
   Call:  (  8) 3 >= 9 ?
   Fail:  (  8) 3 >= 9 ?
   Fail:  (  7) nota(3, _G98) ?

No
```

Poda de la búsqueda mediante corte

- **Programa con corte**

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

- **Traza**

```
?- trace.
Yes
?- nota(3,Y).
   Call:  (  7) nota(3, _G101) ?
   Call:  (  8) 3 < 5 ?
   Exit:  (  8) 3 < 5 ?
   Exit:  (  7) nota(3, suspenso) ?
Y = suspenso;
No
?- trace.
Yes
?- nota(6,Y).
   Call:  (  7) nota(6, _G101) ?
   Call:  (  8) 6 < 5 ?
   Fail:  (  8) 6 < 5 ?
   Redo:  (  7) nota(6, _G101) ?
   Call:  (  8) 6 < 7 ?
   Exit:  (  8) 6 < 7 ?
   Exit:  (  7) nota(6, aprobado) ?
Y = aprobado;
No
```

Poda de la búsqueda mediante corte

- **Ventajas e inconvenientes del uso del corte**
  - **Aumento de eficiencia**
  - **Pérdida de sentido declarativo. Ejemplo:**

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

- **Ejemplo** `maximo`
  - **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
```

Poda de la búsqueda mediante corte

- **Ejemplo** `pertenece`
  - **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`

Negación como fallo

- **Negación con corte**
  - **Programa:**

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

  - **Sesión:**

```
?- trace.
Yes

?- q(a).
   Call:  (  7) q(a) ?
   Call:  (  8) q(b) ?
   Fail:  (  8) q(b) ?
   Redo:  (  7) q(a) ?
   Call:  (  8) q(d) ?
   Exit:  (  8) q(d) ?
   Exit:  (  7) q(a) ?
Yes
```

- **Negación como fallo**
  - **Programa**

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

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

Negación como fallo

- **Sesión:**

```
?- q(a).
Yes

?- trace.
Yes

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

- **Comentarios: eficiencia y claridad**

Negación como fallo

- **Ejemplo con fallo de la negación**
  - **Programa**

```
q(a) :- no(q(b)), q(d).
q(a) :- q(b).
q(b).

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

  - **Sesión**

```
?- trace.
Yes

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

- **Metapredicado primitivo:** `not`, `\+`

Negación como fallo

• Problemas con negación como fallo

• Programa

```
aprobado(X) :- no(suspenso(X)), matriculado(X).
matriculado(juan).
matriculado(luis).
suspenso(juan).

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

• Sesión

```
?- trace.
Yes

?- aprobado(luis).
   Call:  (  8) aprobado(luis) ?
   Call:  (  9) no(suspenso(luis)) ?
   Call:  ( 10) suspenso(luis) ?
   Fail:  ( 10) suspenso(luis) ?
   Redo:  (  9) no(suspenso(luis)) ?
   Exit:  (  9) no(suspenso(luis)) ?
   Call:  (  9) matriculado(luis) ?
   Exit:  (  9) matriculado(luis) ?
   Exit:  (  8) aprobado(luis) ?
Yes
```

Negación como fallo

```
?- trace.
Yes

?- aprobado(X).
   Call:  (  8) aprobado(_G112) ?
   Call:  (  9) no(suspenso(_G112)) ?
   Call:  ( 10) suspenso(_G112) ?
   Exit:  ( 10) suspenso(juan) ?
   Call:  ( 10) fail ?
   Fail:  ( 10) fail ?
   Fail:  (  9) no(suspenso(_G112)) ?
   Fail:  (  8) aprobado(_G112) ?
No
```

- **Interpretación de cláusula**

  ```
  aprobado(X) :- no(suspenso(X)), matriculado(X).
  ```

  "X está aprobado si nadie está suspenso y X está matriculado"

- **Cambio del orden de literales**

```
aprobado(X) :- matriculado(X), no(suspenso(X)).
matriculado(juan).
matriculado(luis).
suspenso(juan).

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

Negación como fallo

- **Sesión**

```
?- trace.
Yes

?- aprobado(X).
   Call:  (  8) aprobado(_G112) ?
   Call:  (  9) matriculado(_G112) ?
   Exit:  (  9) matriculado(juan) ?
   Call:  (  9) no(suspenso(juan)) ?
   Call:  ( 10) suspenso(juan) ?
   Exit:  ( 10) suspenso(juan) ?
   Call:  ( 10) fail ?
   Fail:  ( 10) fail ?
   Fail:  (  9) no(suspenso(juan)) ?
   Redo:  (  9) matriculado(_G112) ?
   Exit:  (  9) matriculado(luis) ?
   Call:  (  9) no(suspenso(luis)) ?
   Call:  ( 10) suspenso(luis) ?
   Fail:  ( 10) suspenso(luis) ?
   Redo:  (  9) no(suspenso(luis)) ?
   Exit:  (  9) no(suspenso(luis)) ?
   Exit:  (  8) aprobado(luis) ?
X = luis
Yes
```

- **Consejo: Asegurar que**
  + se llame con átomos básicos

Alternativas al corte

• Programa con alternativas

• Programa

```
a(p) :- a(q), a(r), a(s), !, a(t).
a(p) :- a(q), a(r), a(u).
a(q).
a(r).
a(u).
```

• Sesión

```
?- trace.
Yes

?- a(p).
   Call:  (  8) a(p) ?
   Call:  (  9) a(q) ?
   Exit:  (  9) a(q) ?
   Call:  (  9) a(r) ?
   Exit:  (  9) a(r) ?
   Call:  (  9) a(s) ?
   Fail:  (  9) a(s) ?
   Redo:  (  8) a(p) ?
   Call:  (  9) a(q) ?
   Exit:  (  9) a(q) ?
   Call:  (  9) a(r) ?
   Exit:  (  9) a(r) ?
   Call:  (  9) a(u) ?
   Exit:  (  9) a(u) ?
   Exit:  (  8) a(p) ?
Yes
```

Alternativas al corte

- **Programa con `if_then_else`**
  - **Programa**

```
a(p) :- a(q), a(r), if_then_else(a(s),a(t),a(u)).
a(q).
a(r).
a(u).

if_then_else(X,Y,Z) :- X, !, Y.
if_then_else(X,Y,Z) :- Z.
```

  - **Sesión**

```
?- trace.
Yes

?- a(p).
   Call:  (  8) a(p) ?
   Call:  (  9) a(q) ?
   Exit:  (  9) a(q) ?
   Call:  (  9) a(r) ?
   Exit:  (  9) a(r) ?
   Call:  (  9) if_then_else(a(s), a(t), a(u)) ?
   Call:  ( 10) a(s) ?
   Fail:  ( 10) a(s) ?
   Redo:  (  9) if_then_else(a(s), a(t), a(u)) ?
   Call:  ( 10) a(u) ?
   Exit:  ( 10) a(u) ?
   Exit:  (  9) if_then_else(a(s), a(t), a(u)) ?
   Exit:  (  8) a(p) ?
Yes
```

Alternativas al corte

- **Programa con meta–predicado ->**
  - **Programa**

```
a(p) :- a(q), a(r),
        (a(s) -> a(t)
        ;        a(u)).
a(q).
a(r).
a(u).
```

  - **Sesión**

```
?- trace.
Yes

?- a(p).
   Call:  (  8) a(p) ?
   Call:  (  9) a(q) ?
   Exit:  (  9) a(q) ?
   Call:  (  9) a(r) ?
   Exit:  (  9) a(r) ?
   Call:  (  9) a(s) ?
   Fail:  (  9) a(s) ?
   Call:  (  9) a(u) ?
   Exit:  (  9) a(u) ?
   Exit:  (  8) a(p) ?
Yes
```

Alternativas al corte

- **Programa con -> múltiple**
  - **Programa**

```
calificacion(X,Y) :-
   nota(X,N),
   ( N < 5 -> Y=suspenso
   ; N < 7 -> Y=aprobado
   ; N < 9 -> Y=notable
   ; true  -> Y=sobresaliente).

nota(juan,6).
```

  - **Sesión**

```
?- trace.
Yes

?- calificacion(juan,X).
   Call:  (  8) calificacion(juan, _G167) ?
   Call:  (  9) nota(juan, _L128) ?
   Exit:  (  9) nota(juan, 6) ?
   Call:  (  9) 6 < 5 ?
   Fail:  (  9) 6 < 5 ?
   Call:  (  9) 6 < 7 ?
   Exit:  (  9) 6 < 7 ?
   Call:  (  9) _G167 = aprobado ?
   Exit:  (  9) aprobado = aprobado ?
   Exit:  (  8) calificacion(juan, aprobado) ?
X = aprobado ;
No
```

Acumuladores

- **Procedimiento `longitud` recursivo**

```
longitud([],0).
longitud([X|R], N) :-
   longitud(R,M),
   N is M+1.
```

- **Procedimiento `longitud` recursivo con acumuladores**

```
longitud(L,N) :-
   longitud_ac(L,0,N).

longitud_ac([],N,N).
longitud_ac([X|R],N0,N) :-
   N1 is N0+1,
   longitud_ac(R,N1,N).
```

- **Comentarios:**
  - **Acumuladores**
  - **Ventaja de procedimientos recursivos finales**
  - **Procediminento primitivo `length`**

Acumuladores

- **Procedimiento `inversa`**

```
inversa([],[]).
inversa([X|L1],L2) :-
   inversa(L1,L3),
   conc(L3,[X],L2).

conc([],L,L).
conc([X|L1],L2,[X|L3]) :-
   conc(L1,L2,L3).
```

- **Procedimiento `inversa` con acumuladores**

```
inversa(L1,L2) :-
   inversa(L1,[],L2).

inversa([],L,L).
inversa([X|L1],L2,L3) :-
   inversa(L1,[X|L2],L3).
```

Acumuladores

- **Comparación de `inversa`**
  - **`inversa` sin acumuladores**

```
?- findall(_X,between(1,100,_X),_L),time(inversa(_L,_L1)).
5,153 inferences in 0.12 seconds (42942 Lips)
?- findall(_X,between(1,1000,_X),_L),time(inversa(_L,_L1)).
501,503 inferences in 15.89 seconds (31561 Lips)
?- findall(_X,between(1,100,_X),_L),profile(inversa(_L,_L1),plain,10).
Predicate          Box Entries =     Calls+Redos = Exits+Fails    Time
=============================================================================
conc/3                   5,050 =         5,050+0 = 5,050+0       76.9%
inversa/2                  101 =           101+0 = 101+0          7.6%
?- findall(_X,between(1,1000,_X),_L),profile(inversa(_L,_L1),plain,10).
Predicate          Box Entries =     Calls+Redos = Exits+Fails    Time
=============================================================================
conc/3                 500,500 =       500,500+0 = 500,500+0     99.6%
inversa/2                1,001 =         1,001+0 = 1,001+0        0.2%
```

Acumuladores

- **inversa con acumuladores**

```
?- findall(_X,between(1,100,_X),_L),time(inversa(_L,_L1)).
104 inferences in 0.00 seconds (Infinite Lips)
?- findall(_X,between(1,1000,_X),_L),time(inversa(_L,_L1)).
1,004 inferences in 0.01 seconds (100400 Lips)
?- findall(_X,between(1,10000,_X),_L),time(inversa(_L,_L1)).
10,004 inferences in 0.19 seconds (52653 Lips)
?- findall(_X,between(1,1000,_X),_L),profile(inversa(_L,_L1),plain,10).
Predicate          Box Entries =     Calls+Redos = Exits+Fails    Time
=============================================================================
inversa/3                1,001 =         1,001+0 = 1,001+0       33.3%
inversa/2                    1 =             1+0 = 1+0            0.0%
?- findall(_X,between(1,10000,_X),_L),profile(inversa(_L,_L1),plain,10).
Predicate          Box Entries =     Calls+Redos = Exits+Fails    Time
=============================================================================
inversa/3               10,001 =        10,001+0 = 10,001+0      90.4%
inversa/2                    1 =             1+0 = 1+0            0.0%
```

Acumuladores

- **Representaciones de [a,b,c] como listas de diferencias**

```
[a,b,c,d] - [d]
[a,b,c,1,2,3] - [1,2,3]
[a,b,c|X] - [X]
[a,b,c] - []
```

- **Concatenación de listas de diferencias**
  - **Programa**

```
conc_ld(X-RX,Y-RY,X-RY) :- RX=Y.
```

  - **Sesión**

```
?- conc_ld([a,b|RX]-RX,[c,d|RY]-RY,Z-[]).
RX = [c, d]
RY = []
Z = [a, b, c, d]
Yes

?- conc_ld([a,b|_RX]-_RX,[c,d|_RY]-_RY,Z-[]).
Z = [a, b, c, d]
Yes
```

Acumuladores

- **Procedimiento `inversa` con listas de diferencias**
  - **Programa**

```
inversa(L1,L2) :-
   inversa_ld(L1,L2-[]).

inversa_ld([],L2-L2).
inversa_ld([X|L1],L2-L3) :-
   inversa_ld(L1,L2-[X|L3]).
```

  - **Traza**

```
?- trace.
Yes

?- inversa([a,b],L).
   Call:  (  8) inversa([a, b], _G149) ?
   Call:  (  9) inversa_ld([a, b], _G149 - []) ?
   Call:  ( 10) inversa_ld([b], _G149 - [a]) ?
   Call:  ( 11) inversa_ld([], _G149 - [b, a]) ?
   Exit:  ( 11) inversa_ld([], [b, a] - [b, a]) ?
   Exit:  ( 10) inversa_ld([b], [b, a] - [a]) ?
   Exit:  (  9) inversa_ld([a, b], [b, a] - []) ?
   Exit:  (  8) inversa([a, b], [b, a]) ?
L = [b, a]
Yes
```

Base de datos interna

• Base de datos interna

```
?- assert(clase(a,vocal)).
Yes

?- listing(clase).
clase(a, vocal).
Yes

?- assert(clase(b,consonante)).
Yes

?- listing(clase).
clase(a, vocal).
clase(b, consonante).
Yes

?- retract(clase(a,vocal)).
Yes
```

Base de datos interna

```
?- listing(clase).
clase(b, consonante).
Yes

?- asserta(clase(a,vocal)).
Yes

?- listing(clase).
clase(a, vocal).
clase(b, consonante).
Yes

?- abolish(clase,2).
Yes

?- listing(clase).
Correct to 'close'? n
Correct to 'clause'? n
[WARNING: No predicates for 'clase']
No
```

Metodogía de programación lógica

- **Enunciado del problema:**

```
?- ord_quicksort([3,2,1,5],L).
L = [1, 2, 3, 5]
```

- **Procedimiento quicksort: Para ordenar la lista no vacía `L`:**
  - **Elegir un elemento `X` de `L` y dividir el resto en las dos listas `Menores` (que contiene los elementos de `L` menores que `X`) y `Mayores` (que contiene los elementos de `L` mayores o iguales que `X`)**
  - **Ordenar los `Menores` obteniendo `MenoresOrdenados`**
  - **Ordenar los `Mayores` obteniendo `MayoresOrdenados`**
  - **La lista `L` ordenada es la concatenación de `MenoresOrdenados` y `[X|MayoresOrdenados]`**

Metodogía de programación lógica

- **Programa**

```
ord_quicksort([],[]).
ord_quicksort([X|R],Ordenada) :-
   divide(X,R,Menores,Mayores),
   ord_quicksort(Menores,MenoresOrdenados),
   ord_quicksort(Mayores,MayoresOrdenados),
   append(MenoresOrdenados,
          [X|MayoresOrdenados],
          Ordenada).

divide(_,[],[],[]).
divide(X,[Y|R],[Y|Menores],Mayores) :-
   menor(Y,X), !,
   divide(X,R,Menores,Mayores).
divide(X,[Y|R],Menores,[Y|Mayores]) :-
   divide(X,R,Menores,Mayores).
```

Bibliografía

- **Bratko, I. "Prolog Programming for Artificial Intelligence (2nd ed.)" (Addison–Wesley, 1990)**
- **Clocksin, W.F. y Mellish, C.S. "Programming in Prolog (Fourth ed.)" (Springer, 1994)**
- **Flach, P. "Simply Logical (Intelligent Reasoning by Example)" (John Wiley, 1994)**
  - **Cap. 3: "Logic programming and Prolog".**
- **Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach)* (Oxford University Press, 1998)**
  - **Ap. B: "The Prolog programming language"**