Programación lógica (2008–09)

Tema 2: Prolog

José A. Alonso Jiménez

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

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos

6. Transformación entre términos, átomos y listas

7. Procedimientos aplicativos

8. Todas las soluciones

Tema 2: Prolog

1. Listas
   Construcción de listas
   Definición de relaciones sobre listas
      Concatenación de listas
      Relación de pertenencia

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos

Construcción de listas

- Definición de listas:
  - La lista vacía [] es una lista.
  - Si L es una lista, entonces .(a,L) es una lista.
- Ejemplos:

```
?- .(X,Y) = [a].
X = a
Y = []
?- .(X,Y) = [a,b].
X = a
Y = [b]
?- .(X,.(Y,Z)) = [a,b].
X = a
Y = b
Z = []
```

Escritura abreviada

- Escritura abreviada:

```
[X|Y] = .(X,Y)
```

- Ejemplos con escritura abreviada:

```
?- [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]
```

Tema 2: Prolog

1. Listas
   - Construcción de listas
   - Definición de relaciones sobre listas
     - Concatenación de listas
     - Relación de pertenencia

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos

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]
```

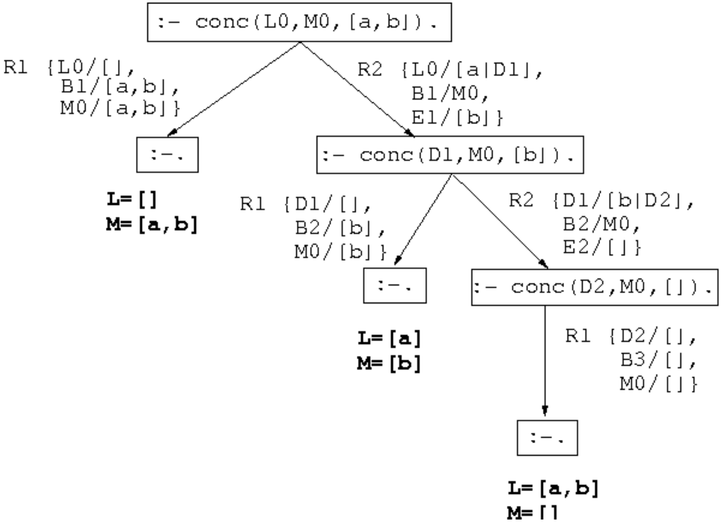
- ¿Qué lista hay que añadirle a la lista `[a,b]` para obtener `[a,b,c,d]`?

```
?- 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
```

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).
```

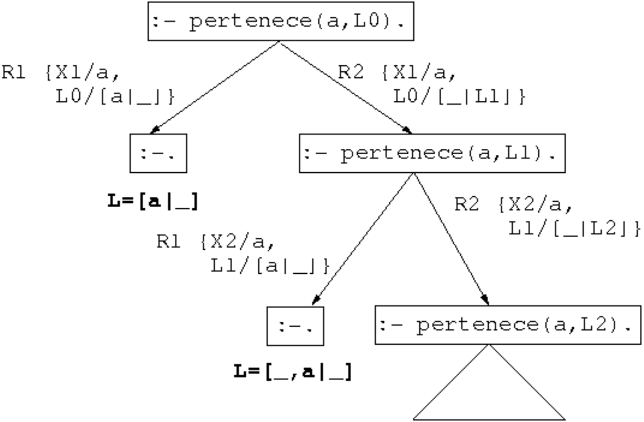
- *Definición 2:*

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

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).`



Disyunciones

- Definición de `pertenece` con disyunción

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

- Definición equivalente sin disyunción

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

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética
   - Operadores
     - Operadores aritméticos
     - Definición de operadores
   - Aritmética
     - Evaluación de expresiones aritméticas
     - Definición de relaciones aritméticas

4. Corte, negación y condicional

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
```

Operadores aritméticos predefinidos

Precedencia	Tipo	Operadores	
500	yfx	+,-	Infijo asocia por la izquierda
500	fx	-	Prefijo no asocia
400	yfx	*, /	Infijo asocia por la izquierda
200	xfy	^	Infijo asocia por la derecha

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 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética
   - Operadores
     - Operadores aritméticos
     - Definición de operadores
   - Aritmética
     - Evaluación de expresiones aritméticas
     - Definición de relaciones aritméticas

4. Corte, negación y condicional

Evaluación de expresiones aritméticas

- Evaluación de expresiones aritmética con `is`.

```
?- X is 2+3^3.
X = 29
?- X is 2+3, Y is 2*X.
X = 5       Y = 10
```

- Relaciones aritméticas: `<`, `=<`, `>`, `>=`, `=:=` y `=/=`

```
?- 3 =< 5.
Yes
?- 3 > X.
% [WARNING: Arguments are not sufficiently instantiated]
?- 2+5 = 10-3.
No
?- 2+5 =:= 10-3.
Yes
```

Definición del factorial

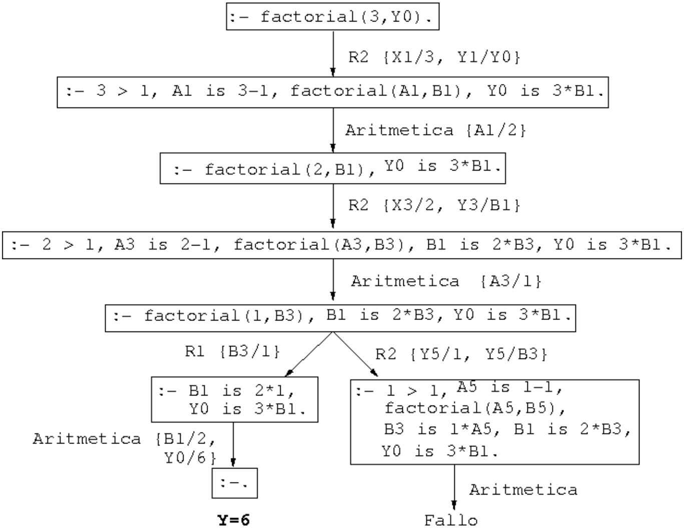
- `factorial(X,Y)` se verifica si `Y` es el factorial de `X`. Por ejemplo,

```
?- factorial(3,Y).
Y = 6 ;
No
```

- Definición:

```
factorial(1,1).
factorial(X,Y) :-
   X > 1,
   A is X - 1,
   factorial(A,B),
   Y is X * B.
```

Árbol de deducción de `?- factorial(3,Y).`



Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional
   - Corte
     - Control mediante corte
     - Ejemplos usando el corte
   - Negación como fallo
     - Definición de la negación como fallo
     - Programas con negación como fallo
   - El condicional

Ejemplo de nota sin corte

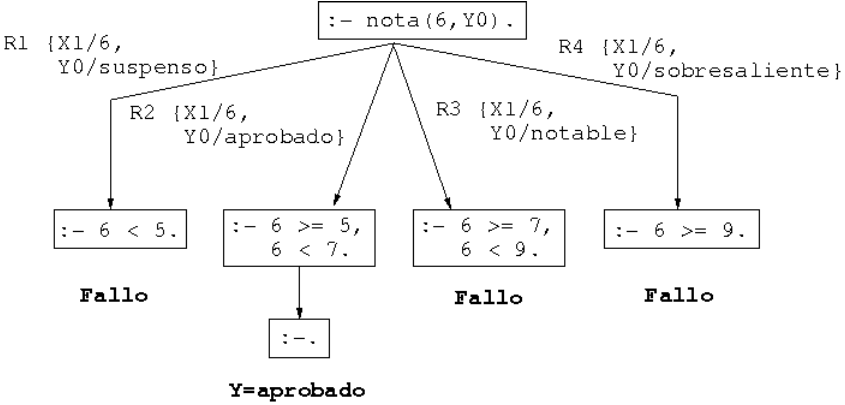
- `nota(X,Y)` se verifica si `Y` es la calificación correspondiente a la nota `X`; es decir, `Y` es `suspenso` si `X` es menor que 5, `Y` es `aprobado` si `X` es mayor o igual que 5 pero menor que 7, `Y` es `notable` si `X` es mayor que 7 pero menor que 9 e `Y` es `sobresaliente` si `X` es mayor que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

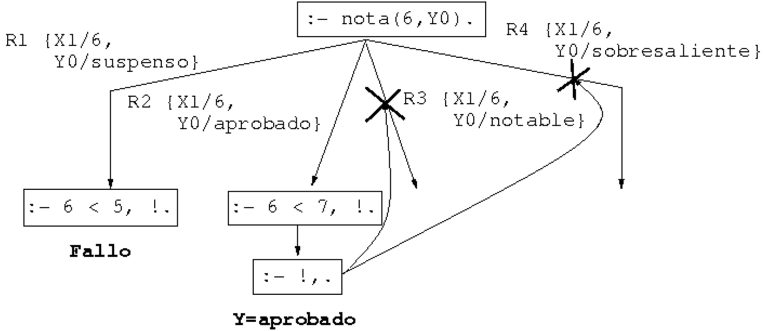


Ejemplo de nota con cortes

- 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
```

Uso de corte para respuesta única

- Diferencia entre member y memberchk

```
?- member(X,[a,b,a,c]), X=a.
X = a ;
X = a ;
No
?- memberchk(X,[a,b,a,c]), X=a.
X = a ;
No
```

- Definición de member y memberchk:

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

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

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional
   - Corte
     - Control mediante corte
     - Ejemplos usando el corte
   - Negación como fallo
     - Definición de la negación como fallo
     - Programas con negación como fallo
   - El condicional

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:

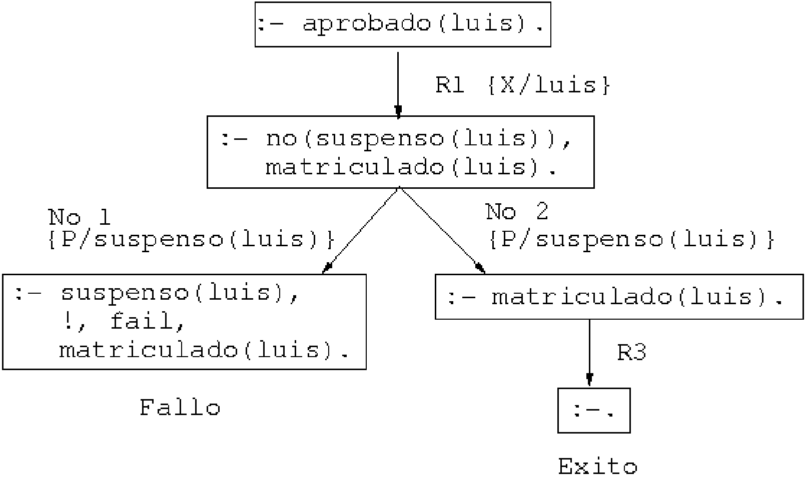
```
aprobado(X) :-                  % R1
   no(suspenso(X)),
   matriculado(X).
matriculado(juan).              % R2
matriculado(luis).              % R3
suspenso(juan).                 % R4
```

- Consultas:

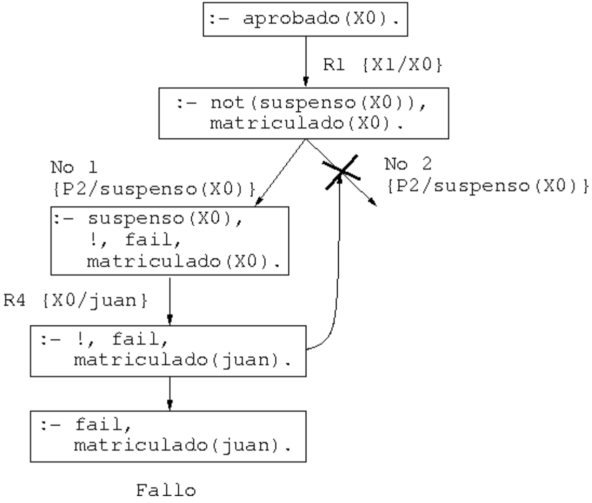
```
?- aprobado(luis).
Yes
```

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

Árbol de deducción de `?- aprobado(luis).`



Árbol de deducción de ?- aprobado(X).



Modificación del orden de los literales

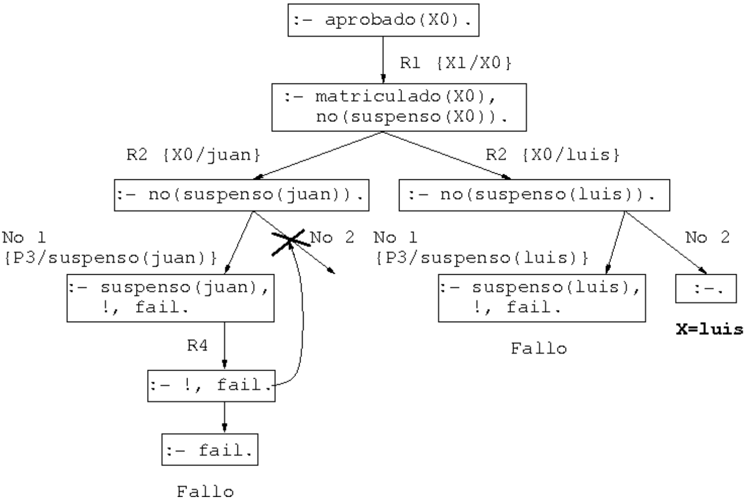
- Programa:

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

- Consulta:

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

Árbol de deducción de `?- aprobado(X).`



Ejemplo de definición con not y con corte

- borra(L1,X,L2) se verifica si L2 es la lista obtenida eliminando los elementos de L1 unificables simultáneamente con X. Por ejemplo,

```
?- borra([a,b,a,c],a,L).
L = [b, c] ;
No
?- borra([a,Y,a,c],a,L).
Y = a
L = [c] ;
No
?- borra([a,Y,a,c],X,L).
Y = a
X = a
L = [c] ;
No
```

Ejemplo de definición con `not` y con corte

- Definición con `not`:

```
borra_1([],_,[]).
borra_1([X|L1],Y,L2) :-
   X=Y,
   borra_1(L1,Y,L2).
borra_1([X|L1],Y,[X|L2]) :-
   not(X=Y),
   borra_1(L1,Y,L2).
```

Ejemplo de definición con `not` y con corte

- Definición con corte:

```
borra_2([],_,[]).
borra_2([X|L1],Y,L2) :-
   X=Y, !,
   borra_2(L1,Y,L2).
borra_2([X|L1],Y,[X|L2]) :-
   % not(X=Y),
   borra_2(L1,Y,L2).
```

Ejemplo de definición con `not` y con corte

- Definición con corte y simplificada

```
borra_3([],_,[]).
borra_3([X|L1],X,L2) :-
   !,
   borra_3(L1,Y,L2).
borra_3([X|L1],Y,[X|L2]) :-
   % not(X=Y),
   borra_3(L1,Y,L2).
```

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional
   - Corte
     - Control mediante corte
     - Ejemplos usando el corte
   - Negación como fallo
     - Definición de la negación como fallo
     - Programas con negación como fallo
   - El condicional

Definición de nota con el condicional

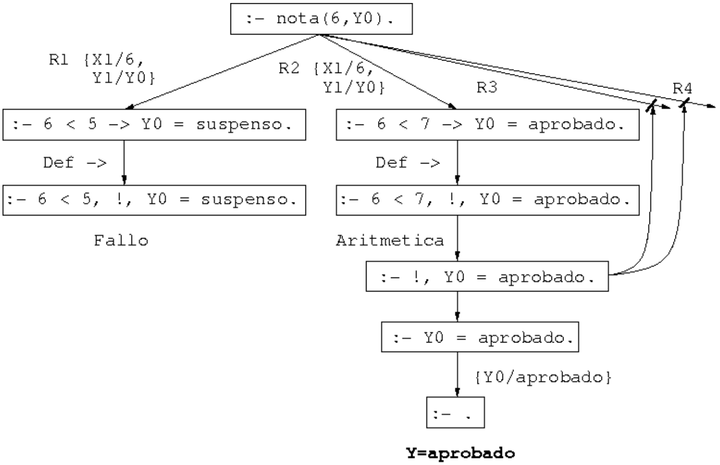
- Definición de nota con el condicional:

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

- Definición del condicional y verdad:

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

Árbol de deducción de `?- nota(6,Y).`



Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos
   - Predicados sobre tipos de término
   - Comparación y ordenación de términos

6. Transformación entre términos, átomos y listas

Predicados sobre tipos de término

- `var(T)` se verifica si T es una variable.
- `atom(T)` se verifica si T es un átomo.
- `number(T)` se verifica si T es un número.
- `compound(T)` se verifica si T es un término compuesto.
- `atomic(T)` se verifica si T es una variable, átomo, cadena o número.

```
?- var(X1).                 =>  Yes
?- atom(átomo).             =>  Yes
?- number(123).             =>  Yes
?- number(-25.14).          =>  Yes
?- compound(f(X,a)).        =>  Yes
?- compound([1,2]).         =>  Yes
?- atomic(átomo).           =>  Yes
?- atomic(123).             =>  Yes
```

Programa con predicados sobre tipos de término

- suma_segura(X,Y,Z) se verifica si X e Y son enteros y Z es la suma de X e Y. Por ejemplo,

```
?- suma_segura(2,3,X).
X = 5
Yes
?- suma_segura(7,a,X).
No
?- X is 7 + a.
% [WARNING: Arithmetic: 'a' is not a function]
```

```
suma_segura(X,Y,Z) :-
   number(X),
   number(Y),
   Z is X+Y.
```

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos
   - Predicados sobre tipos de término
   - Comparación y ordenación de términos

6. Transformación entre términos, átomos y listas

Relaciones de comparación de términos

- `T1 = T2` se verifica si `T1` y `T2` son unificables.
- `T1 == T2` se verifica si `T1` y `T2` son idénticos.
- `T1 \== T2` se verifica si `T1` y `T2` no son idénticos.

```
?- f(X) = f(Y).
X = _G164
Y = _G164
Yes
?- f(X) == f(Y).
No
?- f(X) == f(X).
X = _G170
Yes
```

Programa con comparación de términos

- `cuenta(A,L,N)` se verifique si `N` es el número de ocurrencias del átomo `A` en la lista `L`. Por ejemplo,

```
?- cuenta(a,[a,b,a,a],N).
N = 3
?- cuenta(a,[a,b,X,Y],N).
N = 1
```

```
cuenta(_,[],0).
cuenta(A,[B|L],N) :-
   A == B, !,
   cuenta(A,L,M),
   N is M+1.
cuenta(A,[B|L],N) :-
   % A \== B,
   cuenta(A,L,N).
```

Relaciones de ordenación de términos

- `T1 @< T2` se verifica si el término `T1` es anterior que `T2` en el orden de términos de Prolog.

```
?- ab @< ac.              =>  Yes
?- 21 @< 123.             =>  Yes
?- 12 @< a.               =>  Yes
?- g @< f(b).             =>  Yes
?- f(b) @< f(a,b).        =>  Yes
?- [a,1] @< [a,3].        =>  Yes
```

- `sort(+L1,-L2)` se verifica si `L2` es la lista obtenida ordenando de manera creciente los distintos elementos de `L1` y eliminando las repeticiones.

```
?- sort([c4,2,a5,2,c3,a5,2,a5],L).
L = [2, a5, c3, c4]
```

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos

6. Transformación entre términos, átomos y listas
   - Transformación entre términos y listas
   - Transformaciones entre átomos y listas

Relación de transformación entre términos y listas

- `?T =.. ?L` se verifica si `L` es una lista cuyo primer elemento es el functor del término `T` y los restantes elementos de `L` son los argumentos de `T`. Por ejemplo,

```
?- padre(juan,luis) =.. L.
L = [padre, juan, luis]
?- T =.. [padre, juan, luis].
T = padre(juan,luis)
```

Programa con transformación entre términos y listas

- `alarga(+F1,+N,-F2)` se verifica si `F1` y `F2` son figuras del mismo tipo y el tamaño de `F1` es el de `F2` multiplicado por `N`,

```
?- alarga(triángulo(3,4,5),2,F).
F = triángulo(6, 8, 10)
?- alarga(cuadrado(3),2,F).
F = cuadrado(6)
```

```
alarga(Figura1,Factor,Figura2) :-
   Figura1 =.. [Tipo|Arg1],
   multiplica_lista(Arg1,Factor,Arg2),
   Figura2 =.. [Tipo|Arg2].

multiplica_lista([],_,[]).
multiplica_lista([X1|L1],F,[X2|L2]) :-
   X2 is X1*F, multiplica_lista(L1,F,L2).
```

Las relaciones functor y arg

- `functor(T,F,A)` se verifica si `F` es el functor del término `T` y `A` es su aridad.
- `arg(N,T,A)` se verifica si `A` es el argumento del término `T` que ocupa el lugar `N`.

```
?- functor(g(b,c,d),F,A).
F = g
A = 3
?- functor(T,g,2).
T = g(_G237,_G238)
?- arg(2,g(b,c,d),X).
X = c
?- functor(T,g,3),arg(1,T,b),arg(2,T,c).
T = g(b, c, _G405)
```

Tema 2: Prolog

1. Listas
2. Disyunciones
3. Operadores y aritmética
4. Corte, negación y condicional
5. Relaciones sobre términos
6. Transformación entre términos, átomos y listas
   - Transformación entre términos y listas
   - Transformaciones entre átomos y listas

Relación de transformación entre átomos y listas: name

- `name(A,L)` se verifica si `L` es la lista de códigos ASCII de los caracteres del átomo `A`. Por ejemplo,

```
?- name(bandera,L).
L = [98, 97, 110, 100, 101, 114, 97]
?- name(A,[98, 97, 110, 100, 101, 114, 97]).
A = bandera
```

Programa con transformación entre átomos y listas

- `concatena_átomos(A1,A2,A3)` se verifica si `A3` es la concatenación de los átomos `A1` y `A2`. Por ejemplo,

```
?- concatena_átomos(pi,ojo,X).
X = piojo
```

```
concatena_átomos(A1,A2,A3) :-
   name(A1,L1),
   name(A2,L2),
   append(L1,L2,L3),
   name(A3,L3).
```

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos

6. Transformación entre términos, átomos y listas

7. Procedimientos aplicativos

La relación apply

- `apply(T,L)` se verifica si es demostrable `T` después de aumentar el número de sus argumentos con los elementos de `L`; por ejemplo,

```
plus(2,3,X).                          =>   X=5
apply(plus,[2,3,X]).                  =>   X=5
apply(plus(2),[3,X]).                 =>   X=5
apply(plus(2,3),[X]).                 =>   X=5
apply(append([1,2]),[X,[1,2,3]]).     =>   X=[3]
```

```
n_apply(Término,Lista) :-
   Término =.. [Pred|Arg1],
   append(Arg1,Lista,Arg2),
   Átomo =.. [Pred|Arg2],
   Átomo.
```

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos

6. Transformación entre términos, átomos y listas

7. Procedimientos aplicativos

La relación maplist

- maplist(P,L1,L2) se verifica si se cumple el predicado P sobre los sucesivos pares de elementos de las listas L1 y L2; por ejemplo,

```
?- succ(2,X).                     =>  3
?- succ(X,3).                     =>  2
?- maplist(succ,[2,4],[3,5]).     =>  Yes
?- maplist(succ,[0,4],[3,5]).     =>  No
?- maplist(succ,[2,4],Y).         =>  Y = [3,5]
?- maplist(succ,X,[3,5]).         =>  X = [2,4]
```

```
n_maplist(_,[],[]).
n_maplist(R,[X1|L1],[X2|L2]) :-
   apply(R,[X1,X2]),
   n_maplist(R,L1,L2).
```

Tema 2: Prolog

1. Listas

2. Disyunciones

3. Operadores y aritmética

4. Corte, negación y condicional

5. Relaciones sobre términos

6. Transformación entre términos, átomos y listas

7. Procedimientos aplicativos

Lista de soluciones (findall)

- findall(T,O,L) se verifica si L es la lista de las instancias del término T que verifican el objetivo O.

```
?- assert(clase(a,voc)), assert(clase(b,con)),
   assert(clase(e,voc)), assert(clase(c,con)).
?- findall(X,clase(X,voc),L).
X = _G331   L = [a, e]
?- findall(_X,clase(_X,_Clase),L).
L = [a, b, e, c]
?- findall(X,clase(X,vocal),L).
X = _G355   L = []
?- findall(X,(member(X,[c,b,c]),member(X,[c,b,a])),L).
X = _G373   L = [c, b, c]
?- findall(X,(member(X,[c,b,c]),member(X,[1,2,3])),L).
X = _G373   L = []
```

Conjunto de soluciones (setof)

- `setof(T,O,L)` se verifica si L es la lista ordenada sin repeticiones de las instancias del término T que verifican el objetivo O.

```
?- setof(X,clase(X,Clase),L).
X = _G343   Clase = voc   L = [a, e] ;
X = _G343   Clase = con   L = [b, c] ;
No
?- setof(X,Y^clase(X,Y),L).
X = _G379   Y = _G380   L = [a, b, c, e]
?- setof(X,clase(X,vocal),L).
No
?- setof(X,(member(X,[c,b,c]),member(X,[c,b,a])),L).
X = _G361   L = [b, c]
?- setof(X,(member(X,[c,b,c]),member(X,[1,2,3])),L).
No
```

Multiconjunto de soluciones (bagof)

- `bagof(T,O,L)` se verifica si `L` es el multiconjunto de las instancias del término `T` que verifican el objetivo `O`.

```
?- bagof(X,clase(X,Clase),L).
X = _G343   Clase = voc   L = [a, e] ;
X = _G343   Clase = con   L = [b, c] ;
No
?- bagof(X,Y^clase(X,Y),L).
X = _G379   Y = _G380   L = [a, b, e, c]
?- bagof(X,clase(X,vocal),L).
No
?- bagof(X,(member(X,[c,b,c]),member(X,[c,b,a])),L).
X = _G361   L = [c, b, c]
?- bagof(X,(member(X,[c,b,c]),member(X,[1,2,3])),L).
No
```

Bibliografía

1. J.A. Alonso *Introducción a la programación lógica con Prolog.*
2. I. Bratko *Prolog Programming for Artificial Intelligence (3 ed.)*
3. T. Van Le *Techniques of Prolog Programming*
4. W.F. Clocksin y C.S. Mellish *Programming in Prolog (Fourth Edition)* (Springer Verlag, 1994)