

# Programación Declarativa: Lógica y Restricciones

## Conceptos Básicos de la Programación en Prolog

**Mari Carmen Suárez de Figueroa Baonza**

[mcsuarez@fi.upm.es](mailto:mcsuarez@fi.upm.es)



POLITÉCNICA

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

...

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

# Temas

Identificación

Estructuras de datos

Recursividad, *backtracking* y búsqueda

Control de ejecución

- - -

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

# Unificación (I)

**Unificación** es el mecanismo que se encarga de verificar las igualdades lógicas y de dar valor a las variables lógicas

En la unificación no se evalúan expresiones

Para evaluar expresiones existe un operador especial “is”

Antes de realizar la unificación evalúa la parte derecha como si se tratase de una expresión aritmética

## Unificación (II): Reglas Generales

... y T2 son **constantes**, entonces T1 y T2 unifican si son idénticas

... y T2 son **variables**, entonces T1 y T2 unifican si T1 es una constante y T2 es la misma constante

... es una **variable** y T2 es cualquier tipo de **término**, entonces T1 y T2 unifican y T1 se instancia con T2

... y T2 son **términos complejos**, unifican si:

... tienen el mismo functor y aridad

... y los argumentos unifican

# Unificación (III). Ejemplos

$pe(A,rojo)=jose(B,rojo)?$

$epe!=jose \rightarrow$  No unifica

$pe(A,rojo)=pepe(Z,rojo)?$

$epe=pepe, A=Z, rojo= rojo \rightarrow$  Si unifican

$a?$

$Y?$

$)=f(X)?$

$,a)=f(b,Y)?$

$,a,X)=f(b,Y,c)?$

$f(X)?$

# Unificación (IV): $=/2$

Simboliza el predicado “*unifica*”:

El predicado  $=/2$  está *predefinido* en ISO-Prolog, no es necesario programar la unificación, que es una característica básica del demostrador automático

Los dos argumentos son las dos expresiones a unificar

El predicado es *verdadero* si sus dos argumentos unifican:

$a = a$  ;  $f(a,b) = f(a,b)$  ;  $X = a$  ;  $f(a,b) = X$  ;  $X = Y$  ;  $f(a,X) = f(a,b)$  ;  $X = f(Y)$

El predicado es *falso* si sus dos argumentos no unifican:

$a = b$  ;  $f(a,b) = f(b,a)$  ;  $X = f(X)$  ;  $f(X,X) = f(g(Y),Y)$

# Unificación implícita y la variable anónima

Unificación de una variable puede realizarse:

explícitamente, empleando `=/2` como un objetivo más

implícitamente, dado que `Resolución = Corte + Unificación`

John es amigo de cualquiera que sea rico"

```
rico(x) -> amigo(juan,x))
```

```
amigo(juan,x):- rico(x).
```

Es necesario unificar el segundo argumento de `amigo/2`, para que su valor se emplee en la prueba de `rico/1`

Variable anónima:

Sintácticamente: `'_'`; `'_Anónima'`; `'_X'`

Semánticamente: no se unifica, no toma nunca valor

Pepe es amigo de todo el mundo"

```
amigo(pepe,x)
```

```
amigo(pepe,_).
```

Es innecesario unificar el segundo argumento de `amigo/2`, cualquier valor es aceptable y no se emplea en ulteriores objetivos

# de la Unificación en Ejecución

## of Unification in Execution

As mentioned before, unification used to *access data* and *give values to variables*.

Example: Consider query `?- animal(A), named(A,Name).` with:

```
animal(dog(barry)).
named(dog(Name),Name).
```

Execution of `animal(A)` assigns a (ground) value to `A`.

Execution of `named(A,Name)` assigns a (ground) value to `Name` by accessing the data in the subfield of the `dog/1` structure.

Also, unification is used to *pass parameters* in procedure calls and to *return values* upon procedure exit.

```
?- animal(A), named(A,Name) returns a value upon exit of animal(A)
?- named(dog(barry),Name) passes a value in first argument
                           of call to named/2
                           returns a value in second argument
                           on exit of named/2
Name = barry
```



**Nota:** En la definición de un procedimiento no hay parámetros predefinidos de "entrada" y/o "salida". El modo de uso de cada parámetro depende de la llamada o pregunta que se haga en cada momento al procedimiento.

ct, argument positions are not fixed a priori to be input or output.

*Example:* Consider query `?- pet(spot).` vs. `?- pet(X).`

In a call to a procedure, any argument may be ground, free, or partially instantiated.

Procedures can be used in different **modes** (different sets of arguments are input or output in each mode).

*Example:* Consider the following queries:

`named(dog(barry), Name). % entrada, salida` `?- named(A, barry). % salida, entrada`

`named(dog(barry), barry). % entrada, entrada` `?- named(A, Name). % salida, salida`

An argument may even be both input and output.

*Example:* Consider query `?- struct(f(A, b)).` with:

`struct(f(a, B)).`

## Using Data

Using subfields of *records*:

*Example:*

```
(date(Day, _Month, _Year), Day) .  
date(_Day, Month, _Year), Month) .  
date(_Day, _Month, Year), Year) .
```

Using subfields:

*Example:*

```
date(day, date(Day, _Month, _Year), Day) .  
date(month, date(_Day, Month, _Year), Month) .  
date(year, date(_Day, _Month, Year), Year) .
```

## Using Data (Contd.)

Initializing variables:

```
Example: ?- init(X), ...
```

```
(date(9,6,2011)).
```

Comparing values:

```
Example: ?- init_1(X), init_2(Y), equal(X,Y).
```

```
equal(X,X).
```

```
Example: ?- init_1(X), init_2(X).
```

# Estructurados (I)

## Structured Data and Data Abstraction (and the '=' Predicate)

data structures are created using (complex) terms.

structuring data is important:

```
course(complog,wed,18,30,20,30,'F.','Bueno',new,5102). course/10
```

When is the Computational Logic course?

```
course(complog,Day,StartH,StartM,FinishH,FinishM,C,D,E,F).
```

structured version:

```
course(complog,Time,Lecturer, Location) :- course/4
    Time = t(wed,18:30,20:30),
    Lecturer = lect('F.','Bueno'),
    Location = loc(new,5102).
```

Example: "X=Y" is equivalent to "= (X,Y)"

where the predicate =/2 is defined as the fact "= (X,X)." – Plain unification!

is equivalent to:

```
course(complog, t(wed,18:30,20:30),
    lect('F.','Bueno'), loc(new,5102)).
```

CLASES PARTICULARES, TUTORIAS TÉCNICAS ONLINE  
 LLAMA O ENVIA WHATSAPP: 689 45 44 70  
 ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

# Estructurados (II)

## Structured Data and Data Abstraction (and The Anonymous Variable)

en:

```
course(complog,Time,Lecturer, Location) :-
    Time = t(wed,18:30,20:30),
    Lecturer = lect('F.', 'Bueno'),
    Location = loc(new,5102).
```

What is the Computational Logic course?

```
course(complog,Time, A, B).
```

solution:

```
Time=t(wed,18:30,20:30), A=lect('F.', 'Bueno'), B=loc(new,5102)}
```

Using the *anonymous variable* (“\_”):

```
course(complog,Time, _, _).
```

solution:

```
Time=t(wed,18:30,20:30)}
```

[ysics\\_prolog.pdf](#)

nsparencias 10-19

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

- - -

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

# Ejercicios (I)

Definir **prefijo(X,Y)**: la lista X es un prefijo de la lista Y  
prefijo([a,b], [a,b,c,d]).

Definir **sufijo(X,Y)**: la lista X es un sufijo de la lista Y  
sufijo([c,d], [a,b,c,d]).

Definir **sublista(X,Y)**: la lista X es una sublista de la lista Y  
sublista([b,c],[a,b,c,d]).

Definir **longitud(X,N)**: N es la longitud de la lista X

## Ejercicios (II)

## Palindromos

Definir el predicado **palindromo/1** tal que:

**palindromo(X)** es cierto si la lista X es palíndromo, es decir, puede leerse de la misma manera al derecho y al revés

Ejemplos: **palindromo([r,o,t,o,r])** es verdadero

**palindromo([r,o,t,a,r])** es falso

**palindromo([r,o,t|X])** es verdadero con  $\{X = [o,r]\}$ , o  $\{X = [t,o,r]\}$  o  $\{X = [_A,t,o,r]\}$  o ...

## Primer y Último

Definir el predicado **primerultimo/1** tal que:

**primerultimo(X)** es cierto si el primer y ultimo elementos de la lista X son el mismo

Ejemplos: **primerultimo([a])** es verdadero

**primerultimo([a,f,t])** es falso

**primerultimo([X,f,t,a])** es verdadero con  $\{X = a\}$



# Tracking (I)

## rama lógico:

de **base de conocimientos** donde se expresan los hechos y las reglas de inferencia de un dominio o problema

**Motor de inferencia** que aplica el **algoritmo de resolución**. Este algoritmo permite inferir nuevos datos relativos al mundo que estamos representando

El problema como **entrada** la base de conocimientos y el objetivo planteado y el algoritmo devuelve como **salida** un resultado de verdadero o falso en función de si ha podido o no demostrar el objetivo según la base de conocimientos

Este algoritmo se basa en el uso de la técnica de **backtracking**, de forma que la inferencia del objetivo planteado se realiza a base de prueba y error

## Tracking (II)

hecho puede hacer que un objetivo se cumpla  
diatamente

regla sólo puede reducir la tarea a la de satisfacer  
conjunción de subobjetivos

se puede satisfacer un objetivo, se inicia un proceso  
*backtracking*

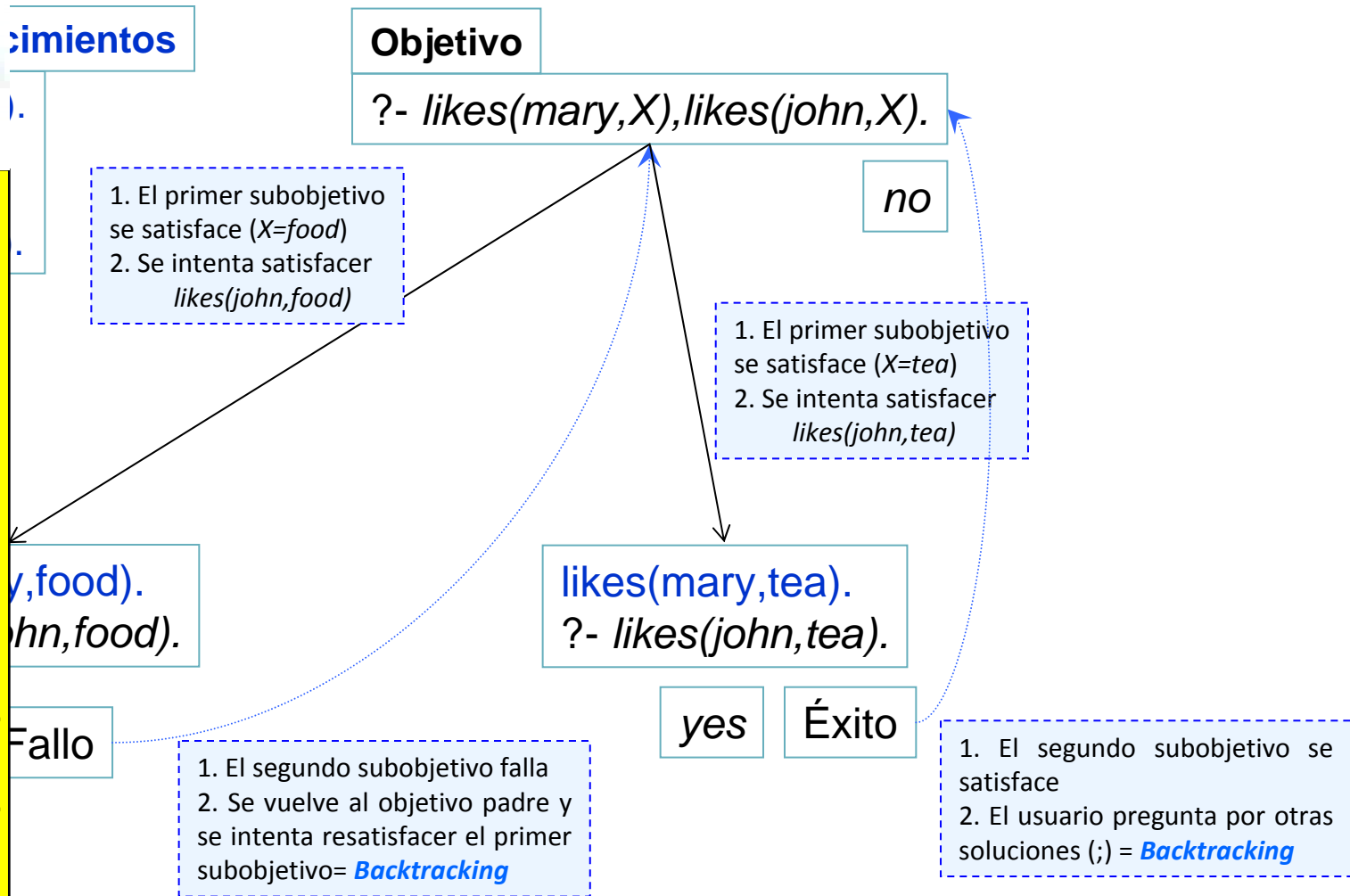
el proceso consiste en intentar satisfacer los objetivos buscando  
forma alternativa de hacerlo

el mecanismo de *backtracking* permite explorar los  
diferentes caminos de ejecución hasta que se encuentre  
solución

*backtracking* por fallo

*backtracking* por acción del usuario

## Tracking (III). Ejemplo



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70  
 ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

# Control de la búsqueda

en 3 formas principales de controlar la ejecución de programa lógico

Orden de las cláusulas en un predicado

Orden de los literales en el cuerpo de una cláusula

Operadores de poda (ej., 'cut')

Orden de las cláusulas en el programa y el orden de los literales en una cláusula son importantes

El orden afecta tanto al correcto funcionamiento del programa, como al recorrido del árbol de llamadas, determinando, entre otras cosas, el orden en que Prolog devuelve las soluciones a una pregunta dada

# Orden de las cláusulas

Orden de las cláusulas determina el orden en que se ven las soluciones

La manera en que se recorren las ramas del árbol de búsqueda de soluciones

Si un árbol de búsqueda tiene alguna rama infinita, el orden de las sentencias puede alterar la obtención de las soluciones, e incluso llegar a la no obtención de ninguna solución

**heurística:** es recomendable que los hechos se prueben antes que las reglas del mismo predicado

# de las cláusulas: Ejemplo (I)

versiones de **miembro de una lista (miembro(X,L))**

ambas versiones tienen las mismas cláusulas pero escritas en distinto orden

versión 1:

`miembro(X,[X|_]).`

`miembro(X,[_|Z):- miembro(X,Z).`

versión 2:

`miembro(X,[_|Z):- miembro(X,Z).`

`miembro(X,[X|_]).`

# de las cláusulas: Ejemplo (II)

En las versiones anteriores les hacemos la misma pregunta

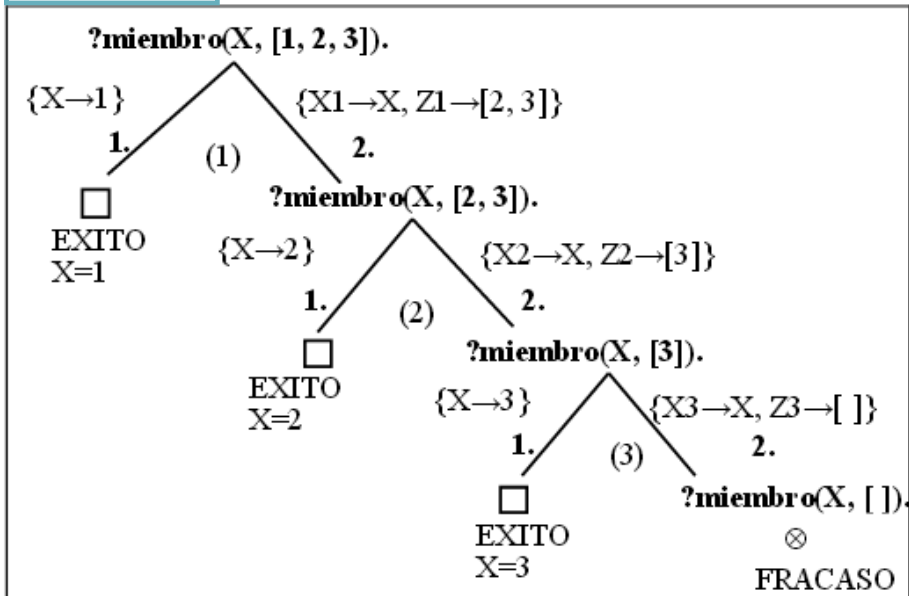
¿miembro (X, [1,2,3]).

En 1:

¿miembro(X,[X|\_]).

¿miembro(X,[\_|Z):- miembro(X,Z).

Versión 1



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70  
 ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

# de las cláusulas: Ejemplo (III)

En ambas versiones les hacemos la misma pregunta

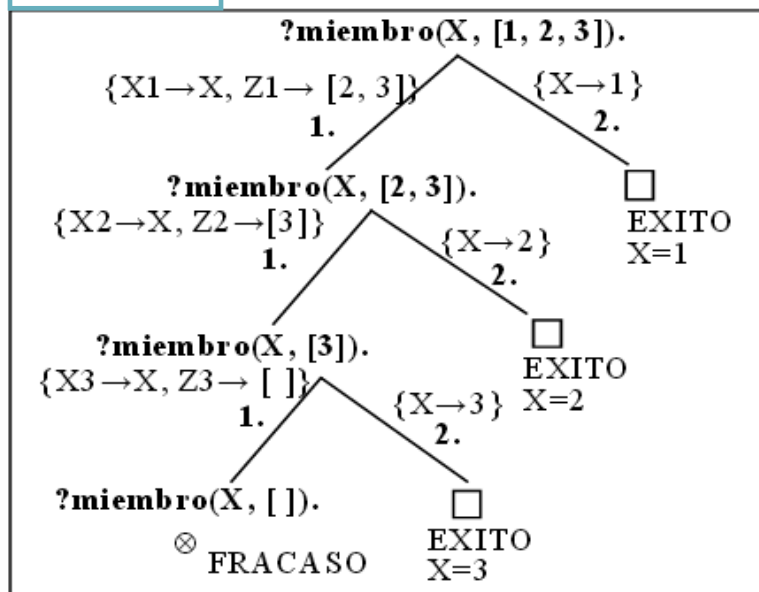
¿miembro (X, [1,2,3]).

versión 2:

miembro(X,[\_|Z):- miembro(X,Z).

miembro(X,[X|\_]).

Versión 2



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70  
 ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP: 689 45 44 70



# Orden de los literales (I)

Orden de los literales dentro de una cláusula afecta al tiempo de búsqueda y a la complejidad de los cálculos

ej.

Diferentes opciones en el orden de los literales pueden ser útiles para distintos modos de uso

$(X, Y) :- \text{hombre}(X), \text{padre}(Y, X).$

para modo (in, out): Se comprueba primero que el X es hombre y después se busca a su padre Y

$(X, Y) :- \text{padre}(Y, X), \text{hombre}(X).$

para modo (out, in): Se buscan los hijos de Y y después se seleccionan si son hombres

# de los literales (II)

den de los literales en el cuerpo de una regla influye  
también en la terminación

ersa([ ], [ ]).

ersa([C|R], Z) :- inversa(R, Y), concatenar(Y, [C], Z).

ara preguntas en modo (in, out) termina

ara preguntas en modo (out, in) el árbol de búsqueda tiene una rama  
infinita, por lo que tras dar la respuesta correcta se queda en un bucle

é sucede si intercambiamos los literales en la regla?

# Control con poda: *cut* (I)

El algoritmo proporciona un predicado predefinido llamado *cut* (I/O) que influye en el comportamiento procedural de los programas

La principal función es reducir el espacio de búsqueda eliminando dinámicamente el árbol de búsqueda

Este tipo puede usarse:

- para aumentar la eficiencia

- para eliminar puntos de *backtracking* que se sabe que no pueden producir ninguna solución

- para modificar el comportamiento del programa

- para eliminar puntos de *backtracking* que pueden producir soluciones válidas.

- para implementar de este modo una forma débil de negación

Este tipo de corte debe utilizarse lo menos posible

## ol con poda: *cut* (II)

predicado *cut* como objetivo se satisface siempre y no se re-satisfacese

esto permite podar ramas del árbol de búsqueda de soluciones

Por consecuencia, un programa que use el *corte* será normalmente más rápido y ocupará menos espacio en memoria (no tiene que recordar los puntos de *backtracking* para una posible reevaluación)

El operador de corte '!' limita el *backtracking*

Cuando se ejecuta el operador de corte elimina todos los puntos de *backtracking* anteriores dentro del predicado donde está definido (incluido el propio predicado)

## Control con poda: *cut* (III)

funcionamiento implica que:

un corte poda todas las alternativas correspondientes a cláusulas debajo de él

un corte poda todas las soluciones alternativas de la conjunción de objetivos que aparezcan a su izquierda en la cláusula

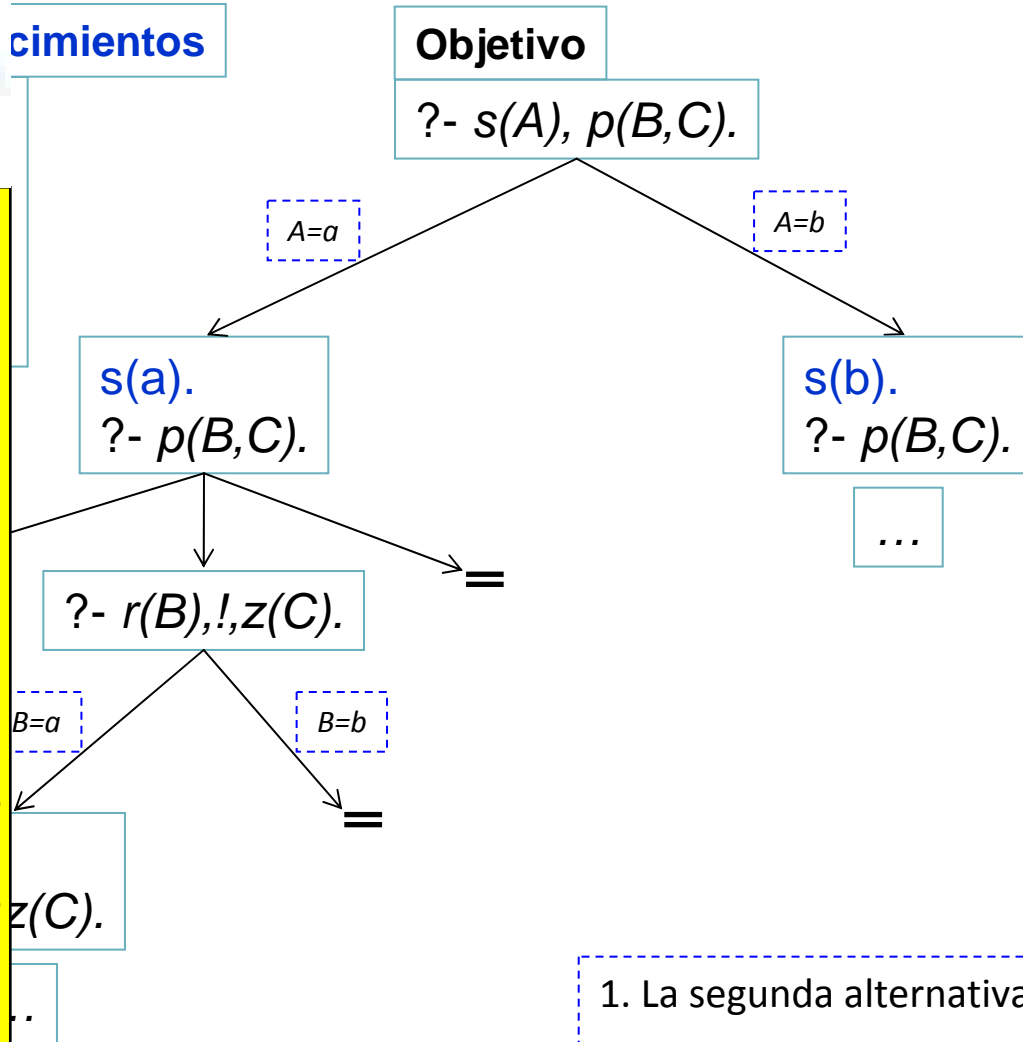
esto es, una conjunción de objetivos seguida por un corte producirá como máximo una solución

un corte no afecta a los objetivos que estén a su derecha en la cláusula

estos objetivos pueden producir más de una solución, en caso de *backtracking*

sin embargo, una vez que esta conjunción fracasa, la búsqueda continuará a partir de la última alternativa que había por encima de la elección de la sentencia que contiene el corte

# Control con poda: *cut* (IV). Ejemplo



1. La segunda alternativa de r/1 ( $r(b)$ ) no se considera
2. La tercera cláusula de p/2 no se considera

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70  
 ...  
 ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP: 689 45 44 70

## Control con poda: *cut* (V)

Analizando, de forma general, el efecto de un corte en una regla C de la forma  $A :- B_1, \dots, B_k, !, B_{k+2}, \dots, B_n$ , es el siguiente:

El objetivo actual G se unifica con A y los objetivos  $B_1, \dots, B_k$  se satisfacen, entonces el programa fija la elección de esta regla para producir G; cualquier otra regla alternativa para A que pueda unificarse con G se ignora.

Además, si los  $B_i$  con  $i > k$  fracasan, la vuelta atrás sólo puede darse hasta el corte. Las demás elecciones que quedaran para computar los  $B_i$  con  $i \leq k$  se han cortado del árbol de búsqueda.

El *backtracking* llega de hecho al corte entonces éste fracasa y la búsqueda continúa desde la última elección hecha antes de que G unificara la regla C.

# Algoritmo con poda: *cut*. Ejemplo (I)

función  $mezcla(L1, L2, L)$ , en modo (in,in,out), mezcla dos listas ordenadas de números  $L1$  y  $L2$  en la lista ordenada  $L$

$mezcla([X | Xs], [Y | Ys], [X | Zs]) :- X < Y, mezcla(Xs, [Y | Ys], Zs).$

$mezcla([X | Xs], [Y | Ys], [X, Y | Zs]) :- X = Y, mezcla(Xs, Ys, Zs).$

$mezcla([X | Xs], [Y | Ys], [Y | Zs]) :- X > Y, mezcla([X | Xs], Ys, Zs).$

$mezcla(Xs, [], Xs).$

$mezcla([], Ys, Ys).$

La mezcla de dos listas ordenadas es una operación determinista

Por lo tanto una de las cinco cláusulas se aplica para cada objetivo (no ambiguo) en una computación dada

Por ejemplo, concretamente, cuando comparamos dos números  $X$  e  $Y$ , sólo una de las tres comprobaciones  $X < Y$ ,  $X = Y$  ó  $X > Y$  es cierta

Por lo tanto, cada vez una comprobación se satisface, no existe posibilidad de que alguna otra comprobación se satisfaga



# Control con poda: *cut*. Ejemplo (II)

El *cut* puede usarse para expresar la naturaleza mutuamente exclusiva de las comprobaciones de las primeras cláusulas

Cláusula 4:  $!mezcla([X | Xs], [Y | Ys], [X | Zs]) :- X < Y, !, mezcla(Xs, [Y | Ys], Zs).$

Cláusula 5:  $mezcla([X | Xs], [Y | Ys], [X, Y | Zs]) :- X = Y, !, mezcla(Xs, Ys, Zs).$

Cláusula 6:  $mezcla([X | Xs], [Y | Ys], [Y | Zs]) :- X > Y, !, mezcla([X | Xs], Ys, Zs).$

En otra parte, los dos casos básicos del programa (cláusulas 4 y 5) son también deterministas

Cláusula 4:  $!mezcla(Xs, [], Xs) :- !.$

Cláusula 5:  $mezcla([], Ys, Ys).$

La cláusula correcta a utilizar se elige por unificación con la cabeza, por eso el *cut* aparece como el primer objetivo (en este caso el *!cut*) en el cuerpo de la cláusula 4

Dicho *cut* elimina la solución redundante (que se volvería a obtener con la cláusula 5) dado el objetivo “?-mezcla([], [], X)”

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70  
 ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

# de Corte

es: descartan soluciones correctas que no son  
sarias

afectan al sentido declarativo del programa

o afectan a la eficiencia del programa

ctan a la completitud pero no a la corrección

an ramas inútiles, redundantes o infinitas

es: descartan soluciones que no son correctas

ctan a la semántica declarativa del programa

modifican el significado lógico del programa

eliminar el operador de corte se obtiene un programa incorrecto

an soluciones erróneas podando ramas que conducen a éxitos  
deseados

teran el significado declarativo del programa  
 n programa semánticamente correcto se añade el  
 para obtener un programa más eficiente  
 ralmente se usan para expresar determinismo  
 arte del cuerpo que precede al corte (o a veces el patrón de la  
 eza) comprueba un caso que excluye a todos los demás

s:

ress(X,Add):- home\_address(X,Add), !.

ress(X,Add):- business\_address(X,Add).

mbercheck(X,[X | Xs]):- !.

mbercheck(X,[Y | Xs]):- membercheck(X,Xs).

# s Verdes. Ejemplo (I)

le verde para **evitar soluciones redundantes**

\_padre(X):-padre(X,Y),!.

dre(antonio,juan).

dre(antonio,maria).

dre(antonio,jose).

--

[Corte.pl](http://Corte.pl)

# Clases Verdes. Ejemplo (II)

Clase verde para **evitar búsquedas inútiles**

El predicado `ordenar(L,R)` indica que R es el resultado de ordenar la lista L por medio de intercambios sucesivos. Este predicado se usa para `ordenada(L)` que nos dice si la lista L está ordenada.

```
ordenar (L, R) :- concatenar(P, [X,Y | S], L), X>Y, !, concatenar(P, [Y,X| S], NL), ordenar(NL, R).
```

```
ordenar (L, L) :- ordenada(L).
```

El programa sabe que sólo hay una lista ordenada. Por tanto, no tiene sentido buscar otras alternativas una vez se ha encontrado la lista ordenada.

an a la semántica declarativa del programa  
n emplearse con cuidado

ejemplos:

$k(X,Y,X) :- X>Y, !.$

$k(X,Y,Y).$

$max(5,2,2).$

genitores(adan,0):-!.

genitores(eva,0):-!.

genitores(P,2).

# ejercicios (I): Uso del Corte

Devolver todas las apariciones de un cierto elemento en una lista dada

`eliminar(X, L1, L2)`

`L2` es la lista obtenida al borrar todas las apariciones de `X` en la lista `L1`

El modo de uso (`in, in, out`), es decir, primer y segundo parámetros de entrada y tercer parámetro de salida

--

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

# ejercicios (II): Uso del Corte

Definir un predicado `agregacion/3`

`agregacion(X,L,L1)`: L1 es la lista obtenida añadiendo el elemento X a la lista L (si X no pertenece a L), y es L en caso contrario

`agregacion(a,[b,c],L).`

`L=[a,b,c]`

`agregacion(b,[b,c],L).`

`L=[b,c]`

...

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70



# ejercicios (III): Uso del Corte

Comprobar si una lista es sublistas de otra

lista(X, Y)

es sublistas de Y

Modo de uso (in, in)

- - -

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

# Programas Generate & Test

básicamente programas que **generan** soluciones candidatas que se evalúan para **comprobar** si son o no correctas

En algunas ocasiones es más sencillo comprobar si algo es una solución a un problema que crear la solución a dicho problema

Existen en dividir la resolución de problemas en dos etapas:

1. **Generar** soluciones candidatas

2. **Comprobar** que las soluciones sean correctas

Los programas con la siguiente estructura:

1. Una serie de objetivos **generan** posibles soluciones vía *backtracking*

2. Los objetivos **comprueban** si dichas soluciones son las apropiadas

# Algoritmos Generate & Test. Ejemplo

## Ordenación de listas

Ordenación(X,Y)

es la lista resultante de ordenar la lista X de forma ascendente  
 la lista Y contiene, en orden ascendente, los mismos elementos que la lista  
 la lista Y es una permutación de la lista X con los elementos en orden  
 ascendente

Ordenación([2,1,2,3], L).

L = [1,2,2,3]

Ordenación(X,Y) :-

Permutacion(X,Y), (1) **Generador:** se obtiene una permutación de X en Y que pasa al objetivo (2) para comprobar si Y está ordenada

Ordenada\_ascendente(Y). (2) **Prueba:** comprueba si la lista está ordenada. Si no lo está, el *backtracking* se encarga de re-satisfacer el objetivo (1) buscando una nueva permutación

[generateTest.pl](#)

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70  
 ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

# Algoritmos *Generate & Test*. Ejercicio

Definir el predicado `numeroParMenor/2` que es verdadero cuando `X` es un número par menor que `N`

`numeroParMenor(X,5).`

`X=0; X=2; X=4`

`numeroParMenor(2,4).`

`es`

`numeroParMenor(3,5).`

`o`

`numeroParMenor(10,7).`

`o`

# Programación Declarativa: Lógica y Restricciones

## Conceptos Básicos de la Programación en Prolog

**Mari Carmen Suárez de Figueroa Baonza**

[mcsuarez@fi.upm.es](mailto:mcsuarez@fi.upm.es)



POLITÉCNICA

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

...

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP: 689 45 44 70