



---

# Práctica I

---

Prolog I: Elementos básicos de Prolog





---

# Contenidos

---

- Cláusulas de Prolog.
- Estructuras, operadores e igualdad.
- Listas y recursión.
- Satisfacción de objetivos.
- Predicado corte.



---

# Cláusulas de Prolog

---



---

# Prolog y Programación Lógica

---

- Prolog: implementación secuencial del modelo del Programa Normal
- Programa Prolog: Conjunto de Cláusulas Horn
- Ejecución: inferencia lógica (búsqueda)
  - Como inferencia: Resolución SLDNF
  - Cómo búsqueda: primero en profundidad, resolviendo primero con el primer literal a la izquierda
  - El orden de las cláusulas es relevante
- Elementos extra-lógicos: cláusulas ejecutable, alteración del flujo de control, entre otras.



---

# Cláusulas de Prolog: Hechos

---

- Describen una propiedades de objetos
  - Ejemplo: el diamante es valioso
  - Hecho:  
*valioso(diamante).*
- El programador debe definir la interpretación de los objetos y las relaciones entre ellos:
  - Objetos: juan, libro
  - Relación:  
*tiene(juan, libro)*
- El orden de la relación es importante



---

# Cláusulas de Prolog: Reglas

---

- Establecen dependencias entre hechos
- Estructura de una regla
  - <cabeza> :- <cuerpo>
  - <cabeza> → relación con variables
  - <cuerpo> → conjunción de relaciones con variables
- Ejemplo:
  - abueloPaterno(X, Z):-padre(X,Y), padre(Y, Z)*
  - Variables: X, Y, Z (palabras que empiezan por mayúscula o guion bajo: "\_")
  - Conjunción: ","



---

# Cláusulas de Prolog: consultas I

---

- Comienzan por:  
?-
- Equivale a preguntar: "¿la consulta se deduce de la base de conocimiento (hechos + reglas)?"
- Ante una consulta, PROLOG intenta hacer un *matching* sobre la base de conocimiento:
  - Mismo predicado
  - Mismo número de argumentos
  - Mismos argumentos: términos / variables (instanciadas)

# Cláusulas de Prolog: consultas II

- Las respuestas a una consulta pueden ser:
  - *Yes* → Se deduce de la BC
  - *No* → No se deduce
- ¿Qué responder a una consulta?
  - *[ENTER]* → Termina
  - *;* → ¿Hay más respuestas?

Base de conocimiento	Consultas	Respuestas
<i>le_gusta_a(jose, maria).</i>	<i>?- le_gusta_a(maria, jose).</i>	<i>No</i>
<i>le_gusta_a(maria, libro).</i>	<i>?- le_gusta_a(maria, libro).</i>	<i>Yes</i>
<i>le_gusta_a(juan, coche).</i>	<i>?- le_gusta_a(juan, pescado).</i>	<i>No</i>
<i>le_gusta_a(jose, pescado).</i>	<i>?- le_gusta_a(jose, pescado).</i>	<i>Yes</i>



---



# Un programa ejemplo

---

*/\* Ejemplo1\_0. pl           → Los comentarios como en C \*/*

*/\* Hechos: le\_gusta\_a(A,B) --> a A le gusta B \*/*

*le\_gusta\_a(juan, maria).*

*le\_gusta\_a(pedro, coche).*

*le\_gusta\_a(maria, libro).*

*le\_gusta\_a(maria, juan).*

*le\_gusta\_a(jose, maria).*

*le\_gusta\_a(jose, coche).*

*le\_gusta\_a(jose, pescado).*

*/\* Reglas: es\_amigo\_de (juan, Persona) --> juan es amigo de  
Persona si a Persona le gustan los coches \*/*

*es\_amigo\_de(juan,X):- le\_gusta\_a(X, coche).*



---

# Ejercicio Cláusulas de Prolog

---

1. Elaborar un programa Prolog que describa a las personas de su familia, sujeto a:
  1. Hechos. El programa debe de incluir la siguiente información mediante hechos:
    1. Hombres y mujeres que la componen: *hombre(X)*, *mujer(X)*
    2. La relación de parentesco más sencilla: *es\_hijo\_de(X,Y)*
  2. Reglas. Que permitan saber
    1. quién es abuelo/a de quién,
    2. quién es padre/madre,
    3. quién es hermana/hermano.
  3. No se permite el uso de operadores en las reglas.
  4. Realizar la consulta *?-hermano(X,Y)* y analizar el resultado.



---

# Estructuras, operadores e igualdad

---

---



# Estructuras

---

- Sintaxis: *functor(comp#1, comp#2, ... ..., comp#n)*.
  - En general, *functor* es un nombre que designa:
    - un hecho,
    - una relación,
    - una función
  - A su vez, *comp#i*, puede ser un functor
- Ejemplo:  
*curso-27(titulacion("Grado en Informática"),  
materia(sistemas\_inteligentes)).*
- Incluso las cláusulas se representan como funtores:  
*?- clause(X, Y). /\* Se satisface con clausulas de cabeza X y  
cuerpo Y \*/*



---

# Operadores aritméticos básicos

---

- $+$ ,  $-$ ,  $*$ , y  $/$ .
- Todos infijo
- Como cualquier otra estructura
  
- Evaluación: ***is***
  - ?-  $X$  *is*  $2 + 2$ .
  - ?-  $X$  *is*  $+(2,2)$ .
  - ?-  $4$  *is*  $+(2,2)$ .



---

# Igualdad y Desigualdad I

---

- El predicado *igualdad*,  $=$ , está predefinido:
  - $?- X = Y.$ 

PROLOG, para satisfacer el objetivo, comprueba si ambas pueden ligarse al mismo objeto.
- Reglas para decidir si  $X$  e  $Y$  son “iguales”:
  - Si  $X$  no está instanciada e  $Y$  sí, entonces son iguales y  $X$  toma como valor el término al que estuviese instanciada  $Y$ .
  - Los números y los átomos siempre son iguales entre sí.
  - Dos estructuras son iguales si tienen el mismo functor y el mismo número de argumentos, y cada uno de esos argumentos son iguales.

# Ejemplos igualdad

consulta	respuesta
?- <i>mesa=mesa.</i>	<i>Yes</i>
?- <i>silla = silla.</i>	<i>Yes</i>
?- <i>mesa = silla.</i>	<i>No</i>
?- <i>2010 = 2009.</i>	<i>No</i>
<i>curso-27(X, materia(Y))=curso-27(titulacion("Grado en Informática"), materia(sistemas_inteligentes)).</i>	<i>X=(titulacion("Grado en Informática")) Y=sistemas_inteligentes</i>
<i>curso-27(_, materia(Y))=curso-27(titulacion("Grado en Informática"), materia(sistemas_inteligentes)).</i>	<i>Y=sistemas_inteligentes</i>



---

# Igualdad y Desigualdad II

---

- Reglas para decidir si  $X$  e  $Y$  son "iguales":
  - Si las 2 variables no están instanciadas, se cumple la igualdad y pasan a ser variables *compartidas*.
    - ?-  $X = Y$ .  
(Cuando una de ellas quede instanciada, fijará el valor de la otra.)  
No tiene interés de forma aislada, pero:
      - ?-  $X=Y$ ,  $X$  is  $2+2$ ,  $Z = Y$ .





---

# Igualdad y Desigualdad III

---

- El predicado *no es igual*,  $\neq$ , también está predefinido:
  - $\neg X = Y$ .  
Se satisface si no se cumple el objetivo  $X = Y$ .



---

## Ejercicio Operadores

---

2. Realiza un programa PROLOG que contenga en la base de conocimiento los signos del Zodiaco. Por ejemplo:  
*horoscopo (aries, 21, 3, 21, 4).*

Escribir dos reglas que permitan calcular el signo del Zodiaco para un día y un mes concreto, por ejemplo:  
*?- signo(Dia, Mes, Signo).*



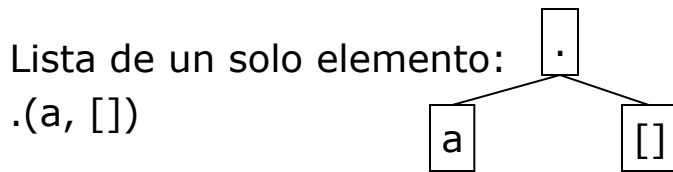
---

# Listas y recursión

---

# Listas

- Tipo particular de árbol: cabeza y cola
  - El functor/estructura asociado es "."
  - El final de la lista es "[]"

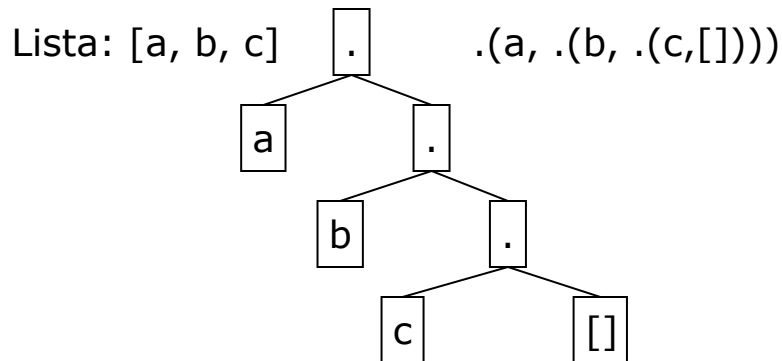


Representación habitual de las listas:

*[a]*

*[a,b,c]*

*[los, hombres, [van, a, pescar]]*



Cabeza es el primer término.

Cola es una lista que contiene al resto.

Una forma de instanciar a una lista con cabeza *X* y cola *Y* sería:

*[X|Y]*



---

# Recursión I

---

- Natural en muchas definiciones declarativas.
- Perspectiva operacional: natural si misma operación sobre distintos datos.
- Esquema genérico:
  1. Caso base: *entero(0)*.
  2. Luego recurrir: *entero(X):- entero(Y), X is Y+1*.



---

# Recursión y listas

---

- Pertenencia: saber si un objeto pertenece a una lista.
- Construir el predicado “miembro”:
  - Caso base: comprobar si el elemento está en la cabeza  
*miembro(X, [X|\_]).* (  $\Leftrightarrow$  *miembro(X, [Y|\_]) :- X=Y.* )
  - Luego recurrir sobre la cola  
*miembro(X, [\_|Y]) :- miembro(X,Y).*
    - Verifica la pertenencia a la cola y de forma recursiva va operando sobre otra lista progresivamente más pequeña



---

# Recursión II

---

- Cuidado con la recursión por la izquierda:

```
enteroMal(X):- enteroMal(Y), X is Y+1.  
enteroMal(0).
```

```
?- enteroMal(X).  
ERROR: Out of local stack
```

- Colocar la regla recursiva como última cláusula de programa.
- Evitar definiciones circulares:

```
padre(X, Y) :- hijo(Y, X).  
hijo(A, B) :- padre(B, A).
```

(se entra en un bucle infinito)

---

# Operaciones con listas: Insertar un elemento

---

- Queremos introducir un elemento  $X$  al comienzo de la Lista.
  - El elemento  $X$  pasará a ser la nueva cabeza de la nueva lista.
  - El cuerpo de la nueva lista será la antigua *Lista*.
- Definición:
  - $insertar1(X, Lista, Resultado) :- Resultado = [X|Lista]$ .
  - Versión compacta:
    - $insertar(X, Lista, [X|Lista])$ .
- Ejemplos:
  - ?-  $insertar(1, [3, 5, 7], Primos)$ .  
 $Primos = [1, 3, 5, 7]$   
Yes
  - ?-  $insertar(rojo, [verde, azul], Colores)$ .  
 $Colores = [rojo, verde, azul]$   
Yes



---

# Operaciones con listas: Concatenar listas

---

- Existe un predicado predefinido *append*:

?- *append*([a, b, c], [1, 2, 3], X).

X=[a, b, c, 1, 2, 3]

?- *append*(X, [b, c, d], [a, b, c, d]).

X=[a]

?- *append*([a], [1, 2, 3], [a, 1, 2, 3]).

Yes

?- *append*([a], [1, 2, 3], [alfa, beta, gamma]).

No

- Definición:

*concatena*([], L, L).

*concatena*([X|L1], L2, [X|L3]) :- *concatena*(L1, L2, L3).



---

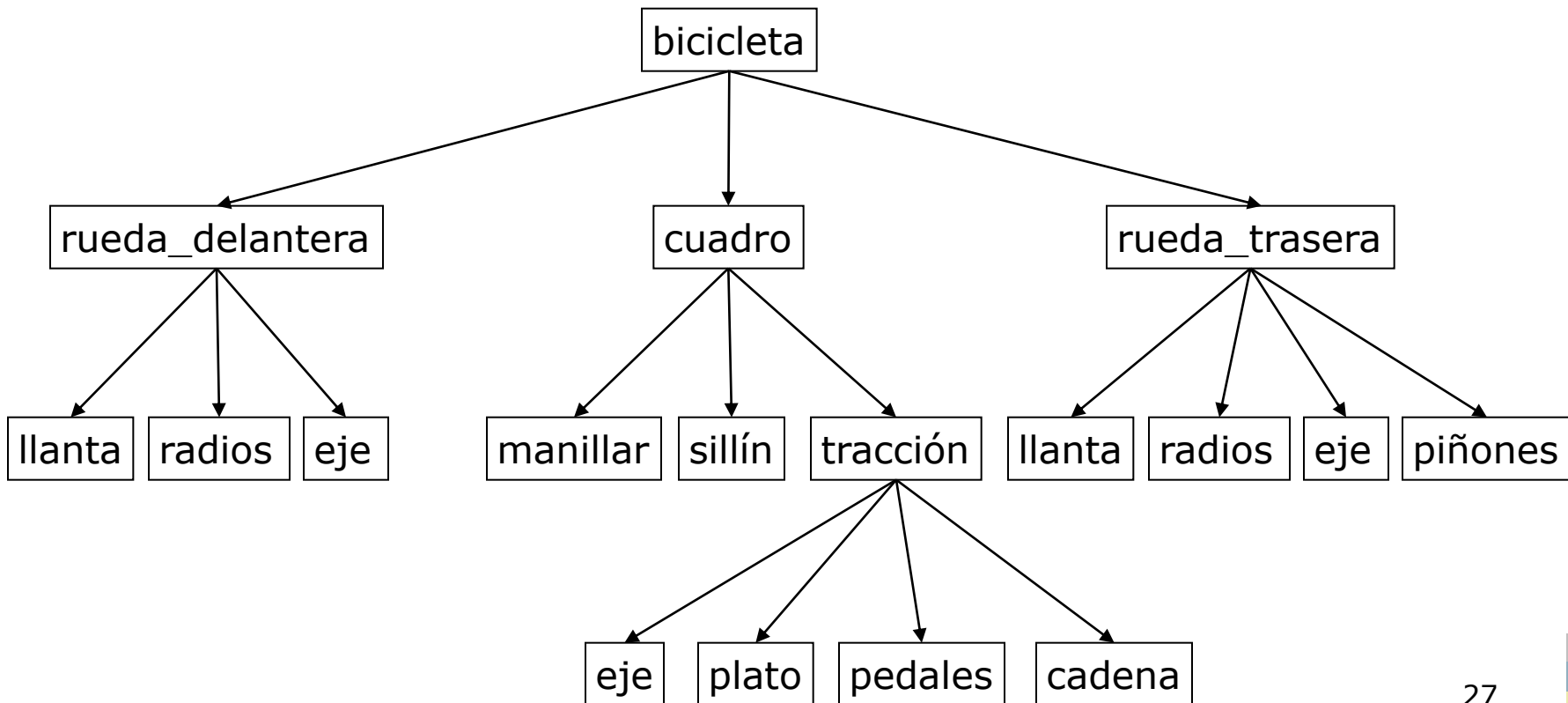
# Ejercicios Listas y recursión (I)

---

3. Definir procedimientos (predicados) para las siguientes operaciones con listas, sin utilizar el correspondiente operador Prolog
  1. *es\_lista? (Lista)*  
*¿Se corresponde Lista con la definición de lista en PROLOG?.*
  2. *longitud(L, Num)*  
*Determina el número de elementos de L.*

# Ejercicios Listas y recursión (II)

## Inventario de piezas





---

## Ejercicios Listas y recursión (III)

---

4. Definir el árbol “Inventario de piezas” mediante las relaciones:
  - `pieza_basica(cadena)`.
  - `ensamblaje(bicicleta, [rueda_delantera, cuadro, rueda_trasera])`.

Definir el procedimiento “`piezas_de`”, que sirva para obtener la lista de piezas básicas para construir una determinada parte de (o toda) la bicicleta.



---

# Satisfacción de Objetivos I

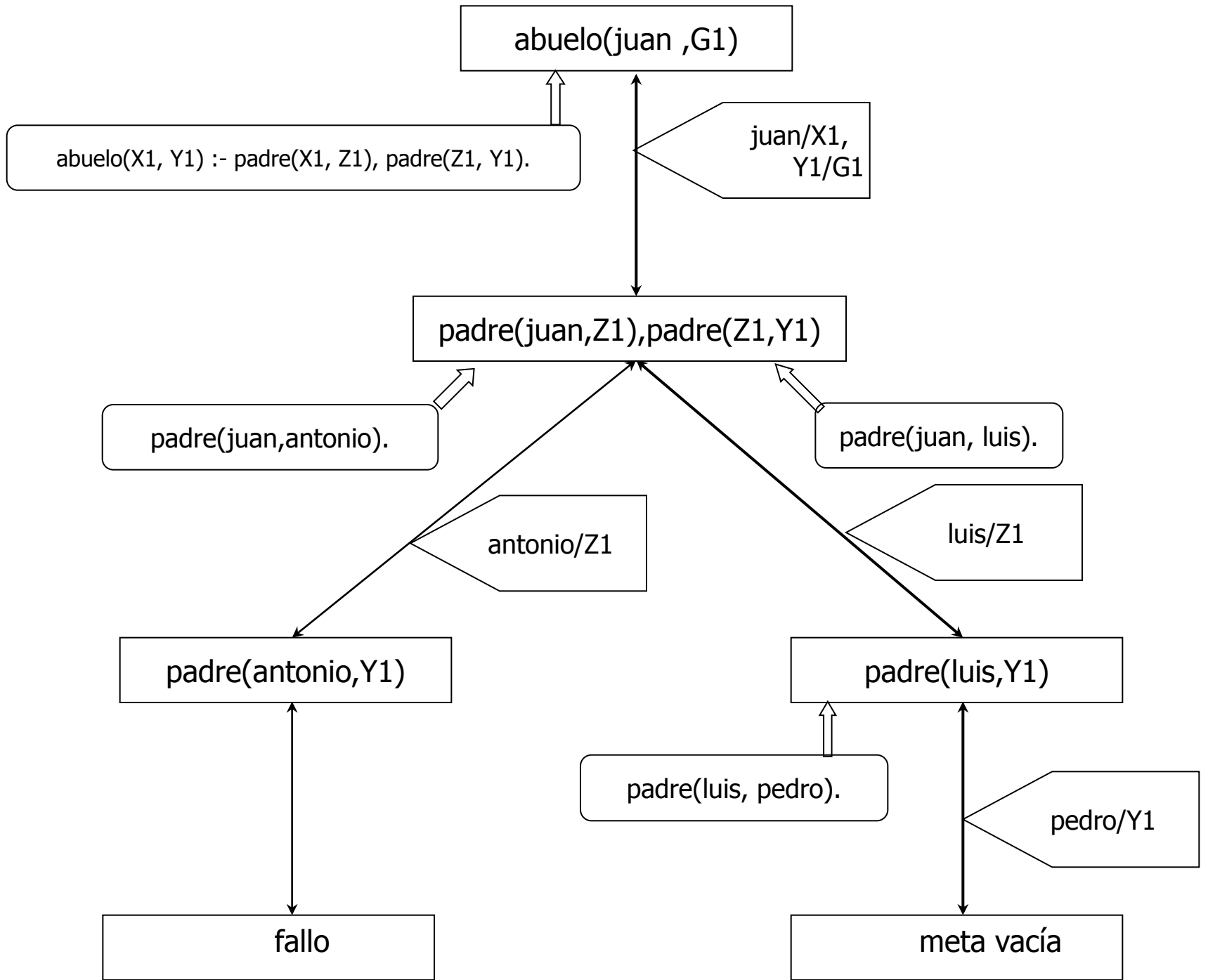
---

```
padre(juan, antonio).
padre(juan, luis).
padre(luis, pedro).
abuelo(X, Y) :- padre(X, Z), padre(Z, Y).
```

```
?- trace, abuelo(juan, X), notrace.
   Call: (8) abuelo(juan, _G382) ? creep
   Call: (9) padre(juan, _L162) ? creep
   Exit: (9) padre(juan, antonio) ? creep
   Call: (9) padre(antonio, _G382) ? creep
   Fail: (9) padre(antonio, _G382) ? creep
   Redo: (9) padre(juan, _L162) ? creep
   Exit: (9) padre(juan, luis) ? creep
   Call: (9) padre(luis, _G382) ? creep
   Exit: (9) padre(luis, pedro) ? creep
   Exit: (8) abuelo(juan, pedro) ? creep
```

X = pedro ; [trace]

No



---



# Satisfacción de Objetivos II

---

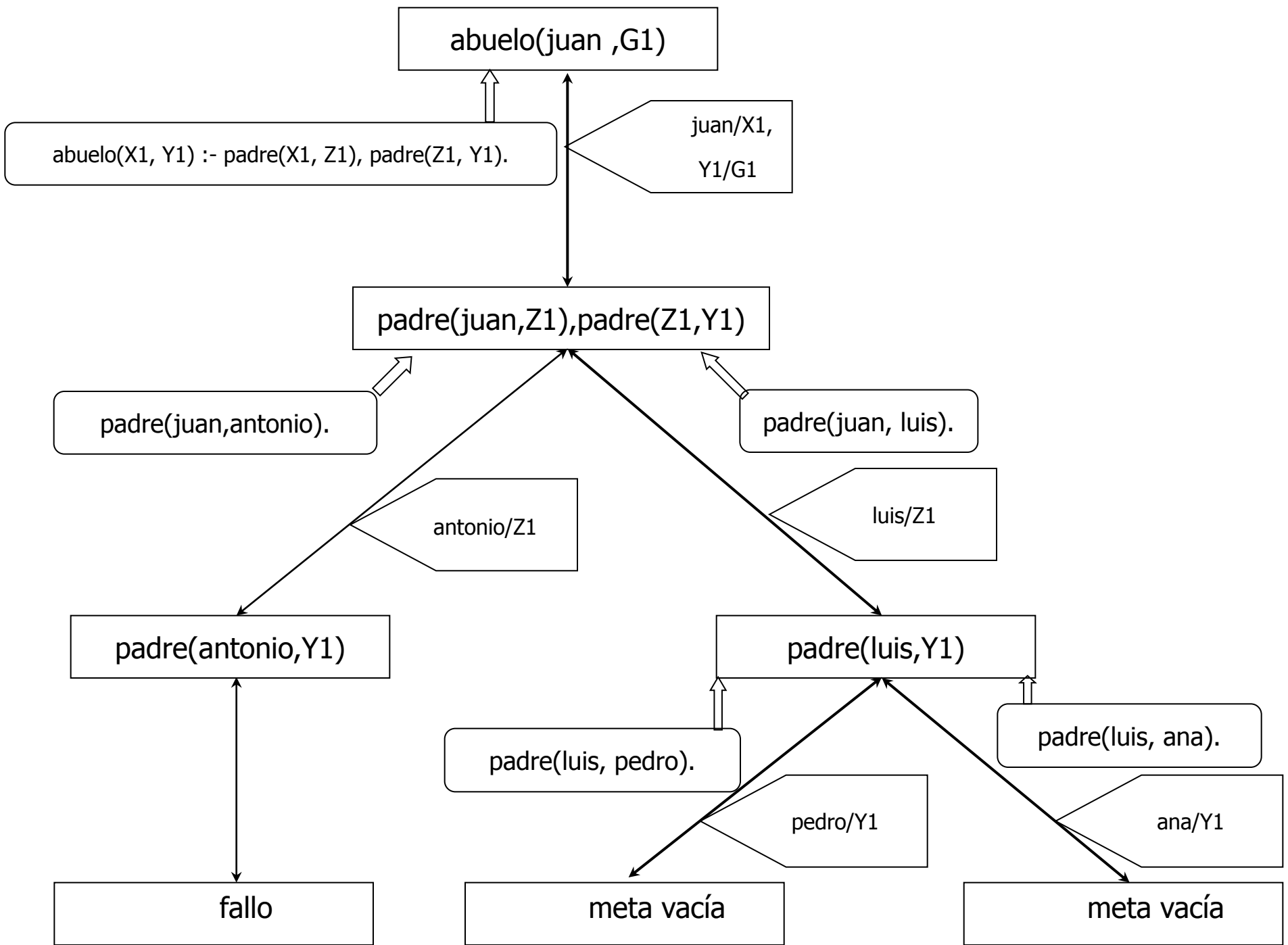
padre(juan, antonio). padre(juan, luis). padre(luis, pedro). padre(luis, ana).  
abuelo(X, Y) :- padre(X, Z), padre(Z, Y).

?- trace, abuelo(juan, X), notrace.  
Call: (8) abuelo(juan, \_G537) ? creep  
Call: (9) padre(juan, \_L200) ? creep  
Exit: (9) padre(juan, antonio) ? creep  
Call: (9) padre(antonio, \_G537) ? creep  
Fail: (9) padre(antonio, \_G537) ? creep  
Redo: (9) padre(juan, \_L200) ? creep  
Exit: (9) padre(juan, luis) ? creep  
Call: (9) padre(luis, \_G537) ? creep  
Exit: (9) padre(luis, pedro) ? creep  
Exit: (8) abuelo(juan, pedro) ? creep

X = pedro ; [trace]  
Redo: (9) padre(luis, \_G537) ? creep  
Exit: (9) padre(luis, ana) ? creep  
Exit: (8) abuelo(juan, ana) ? creep

X = ana ; [trace]

No








---

# El predicado corte: !

---

- Altera el flujo de control
  - Evita explorar ramas del árbol de búsqueda
  - Eficiencia
  - Correcto funcionamiento
  - Pero: altera el significado declarativo del programa
- 

---



# Ejemplo corte I

---

- Ejemplo: Una biblioteca.
  - Libros existentes.
  - Libros prestados y a quién.
  - Fecha de devolución del préstamo...
- Servicios básicos (accesibles a cualquiera):
  - Biblioteca de referencias o mostrador de consulta
- Servicios adicionales (regla):
  - Préstamo normal o interbibliotecario.
- Regla: no permitir servicios adicionales a personas con libros pendientes de devolver.

---



## Ejemplo corte II


---

```
libros_por_devolver(c_perez, libro109). libros_por_devolver(a_ramos, libro297).
cliente(a_ramos). cliente(c_perez). cliente(p_gonzalez).
servicio_basico(referencia). servicio_basico(consulta).
servicio_adicional(prestamo). servicio_adicional(pres_inter_biblio).
```

```
servicio_general(X) :- servicio_basico(X).
servicio_general(X) :- servicio_adicional(X).
```

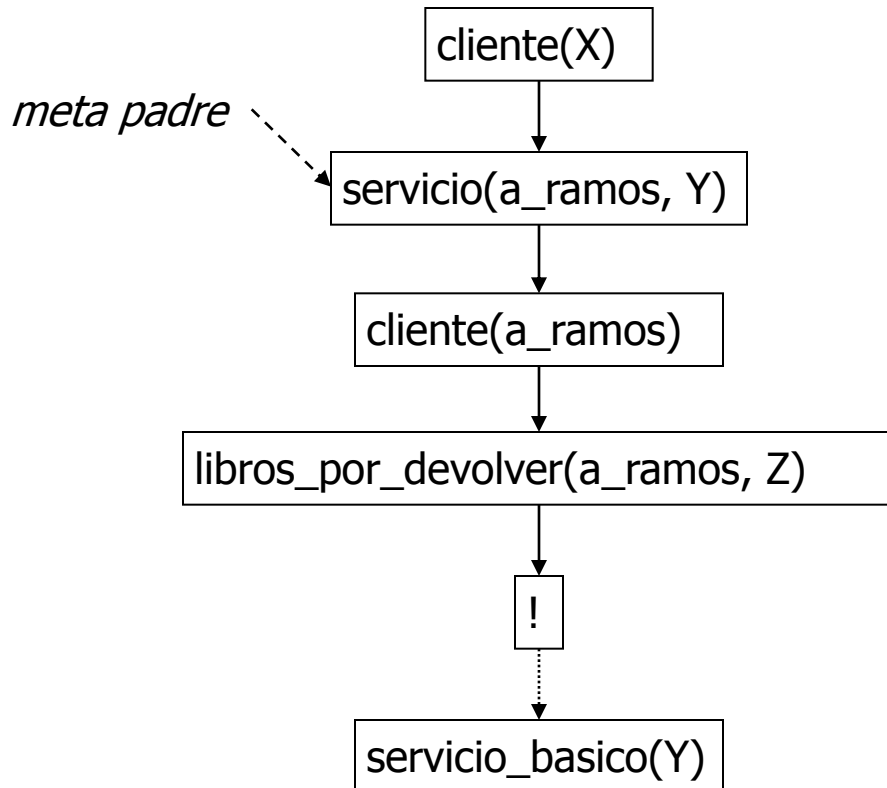
```
servicio(Pers, Serv):-
    cliente(Pers),
    libros_por_devolver(Pers, Libro),
    !,
    servicio_basico(Serv).
```

```
servicio(Pers, Serv):-
    cliente(Pers),
    servicio_general(Serv).
```

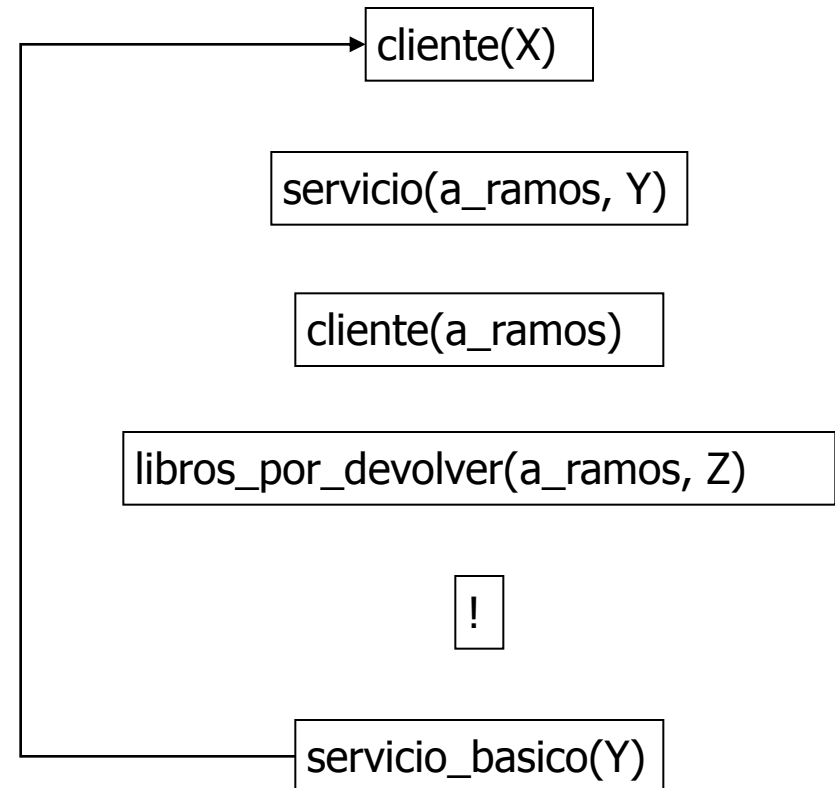


# El predicado corte: alteración del flujo de control

?- cliente(X), servicio(X, Y).



Si intentamos re-satisfacer:





---

# El predicado corte: caracterización operacional

---

- El predicado corte siempre se satisface
- Definición operacional
  - Meta padre: meta que se redujo con la cláusula que contiene el predicado corte
  - Si se alcanza el predicado corte
    - Se satisface eliminándolo de la meta
  - Si se vuelve al predicado corte (backtracking)
    - Devuelve control a la meta anterior a la meta padre
- Obliga a mantener todas las elecciones realizadas entre la meta padre y el corte
  - Las posibles alternativas no se consideran



---

# Aplicaciones del predicado corte

---

- Indicar al sistema que ha llegado a un regla adecuada para satisfacer un objetivo (“si has llegado hasta aquí, has escogido la opción adecuada”)
- Indicar al sistema que debe fallar y no buscar soluciones alternativas (“si has llegado hasta aquí, debes dejar de intentar satisfacer el objetivo”)
- Evitar la generación de soluciones múltiples mediante reevaluaciones (“si has llegado hasta aquí, has encontrado una solución, no sigas buscando”)

---



## Predicado *fail*

---

- *fail* es un predicado predefinido en Prolog.
- Siempre produce un fracaso en la satisfacción del objetivo.
- Desencadena proceso de reevaluación.

```
carnet_uva(X):-  
    matriculado(X), write(X), nl, fail.  
matriculado(juan).  
matriculado(pedro).  
matriculado(maria).  
matriculado(ana).
```

```
?- carnet_uva(X).  
juan  
pedro  
maria  
ana  
No
```

---



# Combinación *cut-fail* y negación I

---

- 'A Elena le gustan los animales, salvo las serpientes'

```
gusta1(elena, X):- serpiente(X), !, fail.  
gusta1(elena, X):-animal(X).
```

```
animal(snoopy).  
animal(lamia).  
serpiente(lamia).  
gusta1(elena, X):- serpiente(X), !, fail.  
gusta1(elena, X):-animal(X).
```

```
?- gusta1(elena, lamia).  
no  
?- gusta1(elena, snoopy).  
Yes  
?-gusta1(elena,X).  
no
```





---

# Combinación corte-*fail* y negación II

---

- Negación : '\+'

animal(snoopy).

animal(lamia).

serpiente(lamia).

gusta(elena, X):- animal(X), \+ serpiente(X).

?- gusta(elena, lamia).

no

?- gusta(elena, snoopy).

Yes

?-gusta(elena,X).

no



---

# Ejercicios predicado corte

---

5. Implementar una función  $f(X, Y)$  que nos proporcione el valor de una función tipo doble escalón con la siguiente definición:
  5. Si  $X < 3$  entonces  $Y := 0$  fin
  6. Si  $X \geq 3$  y  $X < 6$  entonces  $Y := 2$  fin
  7. Si  $X > 6$  entonces  $Y := 4$  fin
6. Implementar el predicado  $borrar\_primera(X, Y, Z)$ , que elimina la primera ocurrencia de  $X$  en la lista  $Y$  generando la lista  $Z$ .
7. Implementar el predicado  $borrar\_todas(X, Y, Z)$ , que elimina todas las ocurrencias de  $X$  en la lista  $Y$  generando la lista  $Z$ . (sugerencia: utilizar el predicado auxiliar  $borrar\_cabeza$ , que toma el elemento y al cabeza y devuelve la lista resultante).
8. Implementar el predicado  $añadir\_sin\_duplicar(X, Y, Z)$  que crea la lista  $Z$  añadiendo  $X$  a la lista  $Y$  si y sólo si  $X$  no está en  $Y$ .