INGENIERIA MULTIMEDIA

PRÁCTICAS

2016-2017

SESIONES PROLOG

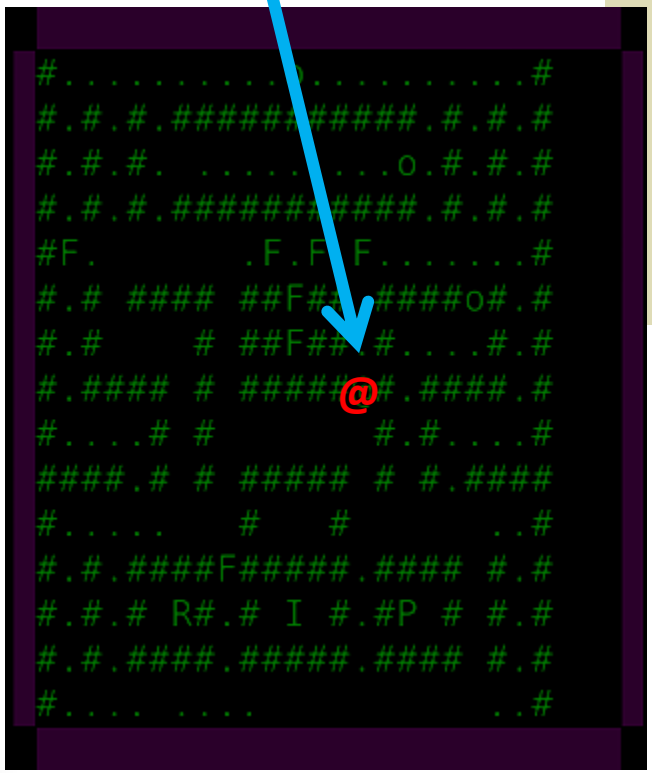
- Presentación, Evaluación.
- Entorno de trabajo.
- Compilar y ejecutar programas Prolog en SWI_Prolog.
- Sentencias de un programa Prolog: Hechos y reglas .
- Ejercicios.

MATEMÁTICAS I

¿ QUÉ VAMOS A HACER ?

"JUGAR" con **PLMAN.**

PLMAN → come cocos voraz



OBJETIVO :

CREAR PROCEDIMIENTOS **"INTELIGENTES"** PARA QUE PLMAN

- → SE **MUEVA**.
- → **COMA** COCOS EVITANDO QUE SE LO "ZAMPEN".

→ LENGUAJE de PROGRAMACIÓN: **PROLOG**

Si Plman se come todos los cocos …
se ha **resuelto** el mapa ☺

Pero si a Plman se lo zampan ….
! Mal asunto ! ☹

demo mapas Plman

MAPAS QUE SE DEBEN RESOLVER

- **Fase 0** (Tutorial) — **5** mapas
- **Fases 1 y 2** — **6** mapas
- **Fase 3** — **2** mapas
- **Fase 4** — **1** mapa

Total — Hasta **14** **Mapas diferentes**

Superar mapa = sumar nota

Mapas limitados

No se puede repetir dificultad en una fase

FASE	DIFICULTAD				
	D1	D2	D3	D4	D5
0	0,100				
1	0,350	0,450	0,500	0,550	0,650
2	0,525	0,675	0,750	0,825	0,975
3	1,225	1,575	1,750	1,925	2,228
4		2,025	2,250	2,475	

Fechas límites de entrega

EVALUACIÓN ENERO

P [40p] = [36p] **M +** [4p] **C.**

M: ejercicios de fases Plman.

C: Control final para validar M.

Si **C < 2p → M = 0 → Suspenso.**

EVALUACIÓN JULIO

P [40p] = **M + C'.**

M [20p]: 20p como máximo: ejercicios fases Plman obtenida durante curso.

No recuperable.

C [20p]: necesario que **C ≥ 10p, ecc → M = 0 → Suspenso**.

WEB : logica.i3a.ua.es

- ASIGNA MAPA
- CORRIGE SOLUCIÓN
- DA RESULTADOS

ELEGIDO MAPA → **EN TU ORDENADOR**

1º PROGRAMAS LA INTELIGENCIA DE PLMAN Y **EJECUTAS**

2º ENTREGAS PROGRAMA AL SISTEMA PARA QUE LO **CORRIJA**

3º OBTIENES **NOTA**

EN EL **MOMENTO DE LA ENTREGA** +

ESTADÍSTICAS DE RESOLUCIÓN

¿ QUÉ SE NECESITA ?

- **Sistema Operativo: Linux (Ubuntu).**

  para crear máquina virtual para Linux-Ubuntu: UACloud.

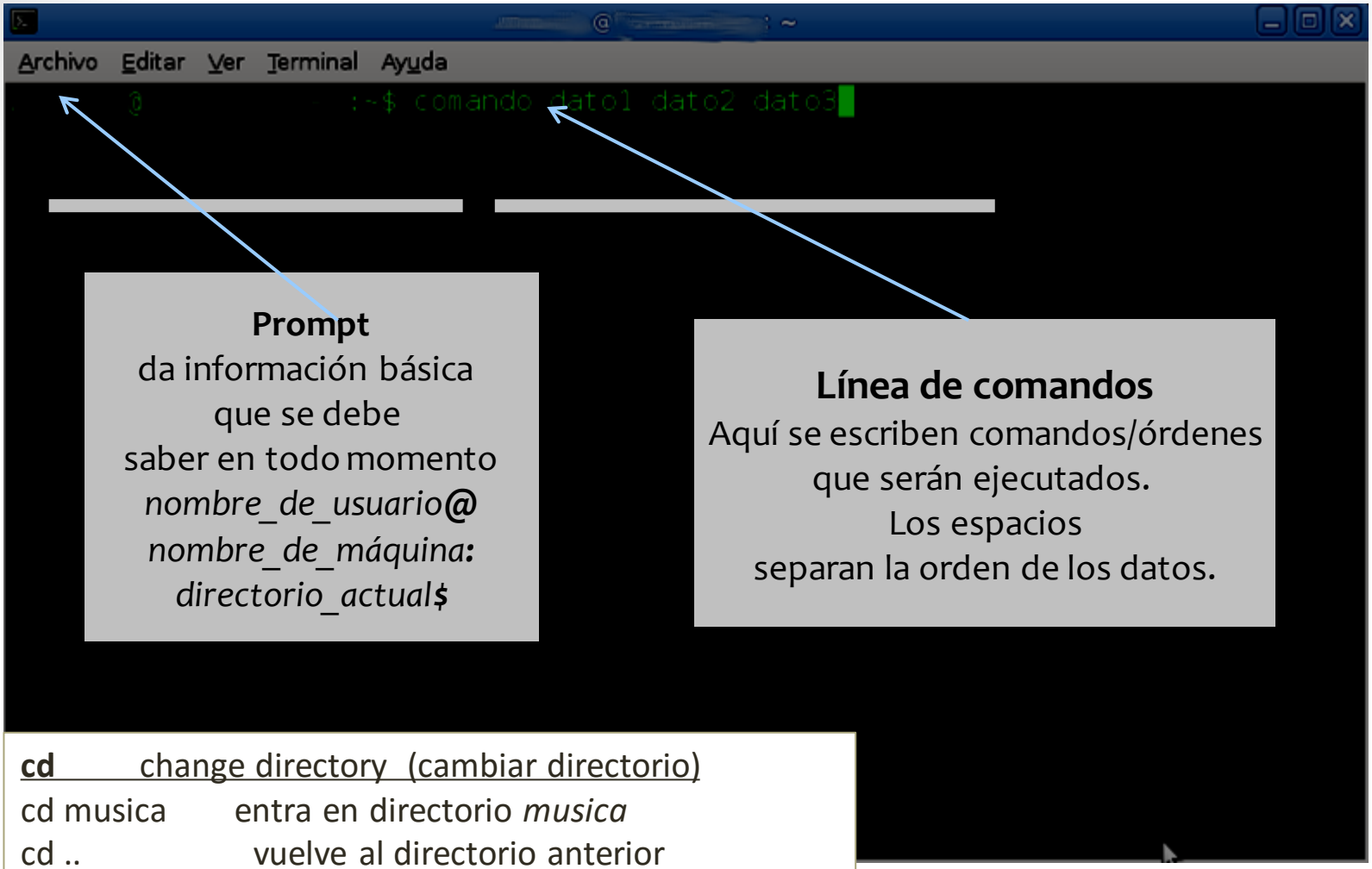
- **Intérprete SWI_Prolog.**

- **Sistema online** para entrega /corrección: **http://logica.i3a.ua.es**

Se os proporciona:

- **Código fuente del juego.**
- **Mapa a resolver**

EMPEZAMOS: TERMINAL LINUX $



cd change directory (cambiar directorio)

cd musica	entra en directorio *musica*
cd ..	vuelve al directorio anterior
cd /	va al directorio raíz del disco duro
cd ~	va directorio del usuario

ls lista contenidos del directorio

ls	lista los contenidos del directorio actual
ls musica	lista contenidos del subdirectorio *musica*

El **Código fuente** del juego **+**

procedimientos para **programar** la inteligencia de Plman

Lenguaje de Programación Lógica: PROLOG

ENTORNO DE PROGRAMACIÓN:

?

http://www.swi-prolog.org

Interfaz : consola de comando textual de dominio público para ordenadores PC desarrollado en U. Amsterdam

Permite: **Compilar / Interpretar / Ejecutar programas Prolog**

70's, Alain Colmerauer y P. Roussell, U Aix-Marseille

"mezclan": **lógica matemática y programación**,

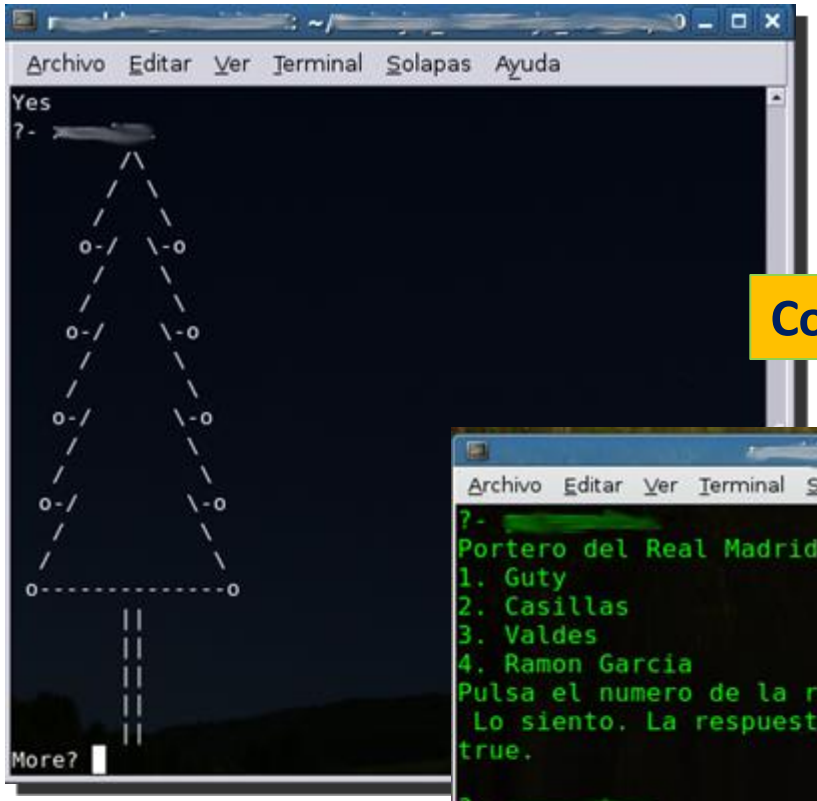
→ Kowalski (1970) **Algoritmo = lógica + control**

Papel importante en IA:

Sistemas expertos

Bases de datos inteligentes…

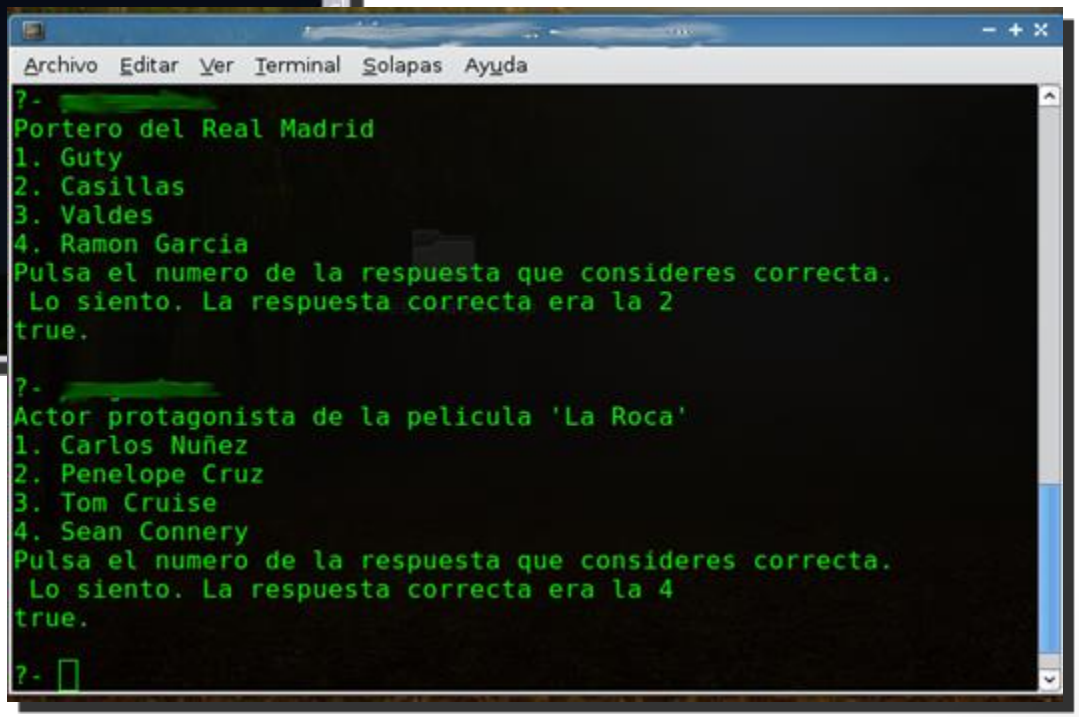
https://logica.i3a.ua.es/descargas



Lanzar consola SWI_Prolog

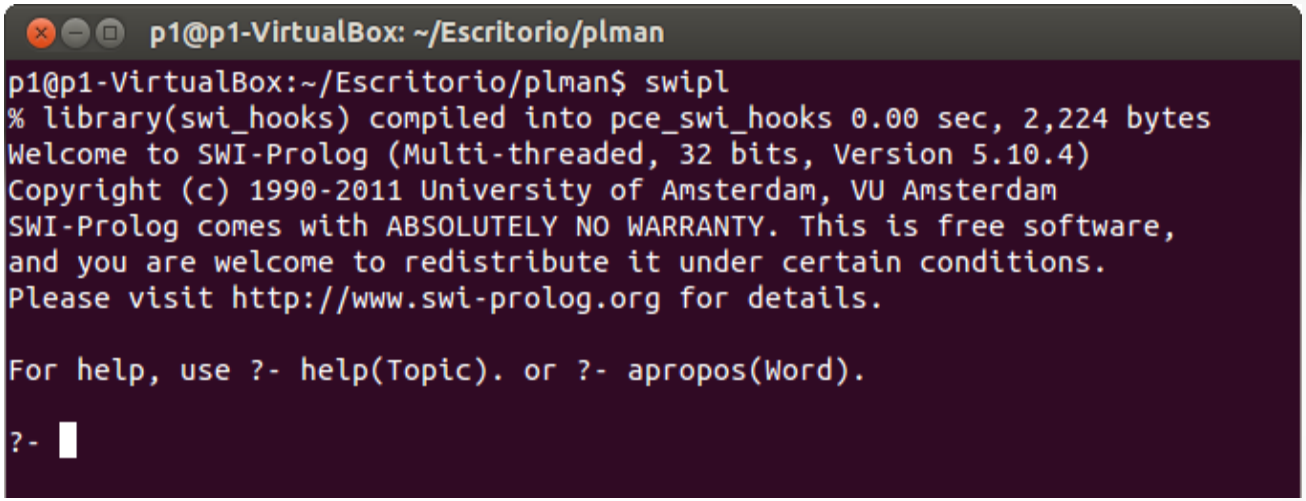
Editar ejemplos (emacs, gedir, kate…)

Compilar / Ejecutar / Modificar



Comando de **lanzamiento** de Swi-Prolog: **> swipl**

Debería aparecer:



Salir de Swi-Prolog y volver a Terminal, escribir:

?- halt.

PASOS para ESCRIBIR y EJECUTAR programas PROLOG

→ **LANZAR el intérprete en Ubuntu:** abrir terminal y teclear **:**

$ swipl

Se abre consola de SWI_Prolog cuyo prompt es : **?**

→ **EDITAR** fichero con código Prolog (*extensión* ***.pl)***:

? emacs('fichero.pl').

→ **COMPILAR** programa : testea errores sintácticos. Se carga en memoria ? listing.

Desde el menú del editor emacs: **Compile /Compile buffer**

→ **OBTENER resultado:** pregunta. **? pregunta.**

→ **COMPILAR y CARGAR en el sistema** un programa.pl, sin editarlo:

? consult('fichero.pl').

→ **LANZAR** el intérprete cargando una base de conocimiento:

$ swipl -f fichero.pl

→ **CREAR EJECUTABLES**: **? swipl –o ejecutable –c fichero.pl**.

El ejecutable creado incorpora un intérprete completo de swi-prolog

1: Carlos es alum.
2: Para que un sujeto sea alum es necesario que tenga buen tipo y
3: ésta es una condición suficiente para que esté macizo.
4: Si un sujeto no es alum, es atractivo.
5: Si un sujeto es atractivo, está macizo.

Reponder : **¿Carlos está macizo? y Luis ? Quién está macizo?**

Formalización

Lpo

Alu(carlos).
Alu(x) → Btipo(x).
Btipo(x) → Ma(x).
¬Alu(x) → At(x).
At(x) → Ma(x).

Conclusiones:
Ma(carlos).
Ma(luis)

Prolog

```
alum(carlos).
buentipo(X) :- alum(X).
macizo(X)   :- buentipo(X).
atractivo(X) :- not(alum(X)).
macizo(X)   :- atractivo(X).
```

Conclusiones/Objetivos:

```
?- macizo(carlos).
?- macizo(luis).
?- macizo(X).
```

BASE DE CONOCIMIENTO

1º ESCRIBIR en un fichero:
? emacs('macizo.pl').

2º COMPILAR desde emacs:
Compile /Compile buffer

¿qué mensaje sale …?
Arreglar y seguir

3º EJECUTAR preguntas desde la consola swipl
? macizo(carlos).
…

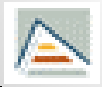
¿Cómo responde Prolog a las preguntas…?
Escribir en consola el depurador
? trace

```
alum(carlos).
buentipo(X) :- alum(X).
macizo(X)   :- buentipo(X).
macizo(X)   :- atractivo(X).
atractivo(X) :- not(alum(X)).
```

Consulta:
?- macizo(carlos).

```
[trace] 2 ?- macizo(carlos).
Call: (7) macizo(carlos) ? creep
Call: (8) buentipo(carlos) ? creep
Call: (9) alum(carlos) ? creep
Exit: (9) alum(carlos) ? creep
Exit: (8) buentipo(carlos) ? creep
Exit: (7) macizo(carlos) ? creep

Yes
```

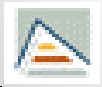



Pregunta para averiguar
todos los sujetos que son "macizos":
? macizo(X).

```
alum(carlos).
buentipo(X) :- alum(X).
macizo(X)   :- buentipo(X).
macizo(X)   :- atractivo(X).
atractivo(X) :- not(alum(X)).
```

```
[trace] 4 ?- macizo(X).
   Call: (8) macizo(_G425) ? creep
   Call: (9) buentipo(_G425) ? creep
   Call: (10) alum(_G425) ? creep
   Exit: (10) alum(carlos) ? creep
   Exit: (9) buentipo(carlos) ? creep
   Exit: (8) macizo(carlos) ? creep
X = carlos
Yes
```

Añade *más sujetos que sean "macizos" y* ***comprueba*** *cómo responde Prolog a la pregunta anterior.*



*Pregunta por un sujeto que **no** esté declarado como "macizo", por ejemplo,*

?- macizo(friqui).

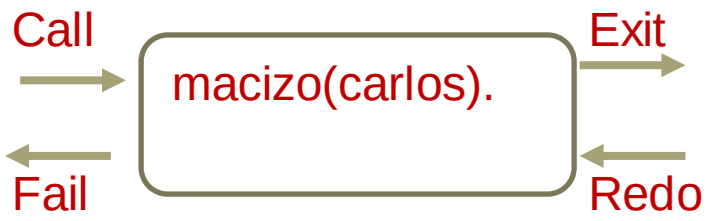
```
alum(carlos).
buentipo(X) :- alum(X).
macizo(X)   :- buentipo(X).
macizo(X)   :- atractivo(X).
atractivo(X) :- not(alum(X)).
```

```
[trace] 7 ?- macizo(friqui).
   Call: (8) macizo(friqui) ? creep
   Call: (9) buentipo(friqui) ? creep
   Call: (10) alum(friqui) ? creep
   Fail: (10) alum(friqui) ? creep
   Fail: (9) buentipo(friqui) ? creep
   Redo: (8) macizo(friqui) ? creep
   Call: (9) atractivo(friqui) ? creep
^  Call: (10) not(alum(friqui)) ? creep
   Call: (11) alum(friqui) ? creep
   Fail: (11) alum(friqui) ? creep
^  Exit: (10) not(alum(friqui)) ? creep
   Exit: (9) atractivo(friqui) ? creep
   Exit: (8) macizo(friqui) ? creep

Yes
```

En los predicados de un sistema Prolog se consideran **4** estados (2 entrada y 2 salida):

- **Call** (llamada): comienza la ejecución del objetivo (predicado de la pregunta).
- **Exit** (salida): salida con éxito del objetivo.
- **Redo** (reintentar): reintentar el predicado utilizando otra alternativa.
- **Fail** (fallo): salida con fallo del objetivo. No se encuentran soluciones.



Desactivar depurador: ? **notrace/0**

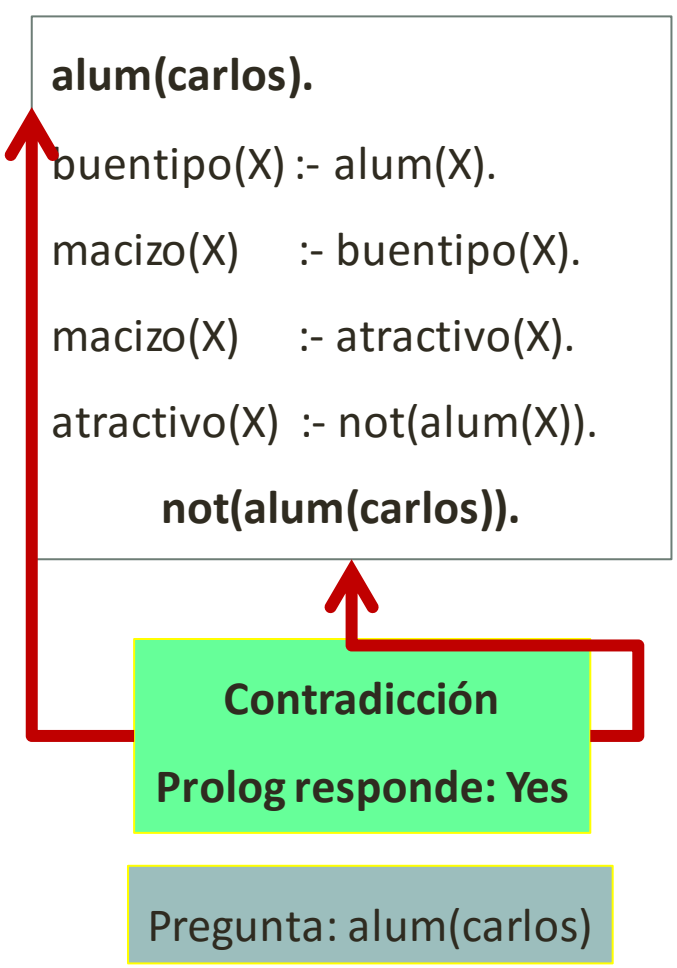
Al hacer una **pregunta a** Prolog se activa un proceso para obtener una respuesta:

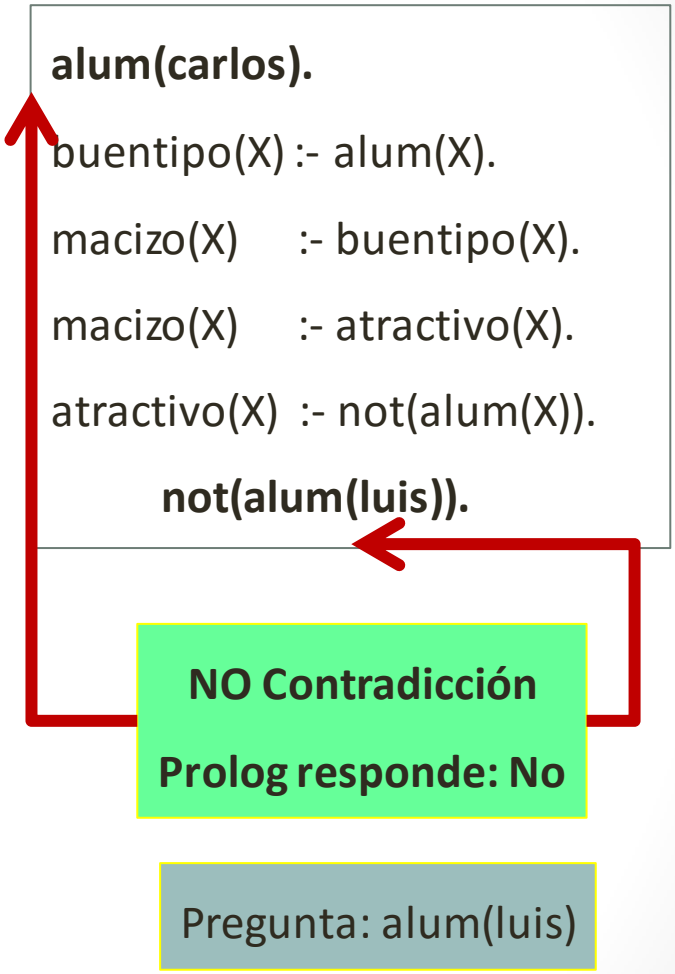
ÁRBOL de RESOLUCIÓN SLD → Da una respuesta.

Si se precisan más respuestas → **REEVALUACIÓN / BACKTRACKING**

PROCESO :

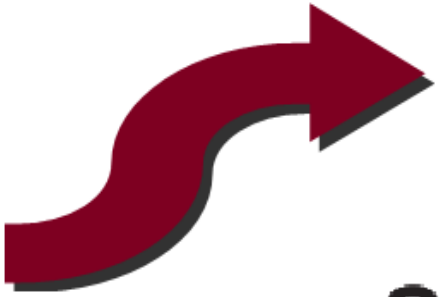
- El sistema tiene cargada la BC como programa Prolog
- **Refutación** : la pregunta se niega y se añade a la BC.
- El sistema **busca** la aparición de un absurdo recorriendo la **BC de arriba-debajo**.
- Usa punteros para marcar el hecho que **coincida** con el de la pregunta.
- Si encuentra **absurdo** responde resultados y YES, si no, responde NO (false).


alum(carlos).
buentipo(X) :- alum(X).
macizo(X) :- buentipo(X).
macizo(X) :- atractivo(X).
atractivo(X) :- not(alum(X)).
not(alum(carlos)).
Contradicción
Prolog responde: Yes
Pregunta: alum(carlos)


alum(carlos).
buentipo(X) :- alum(X).
macizo(X) :- buentipo(X).
macizo(X) :- atractivo(X).
atractivo(X) :- not(alum(X)).
not(alum(luis)).
NO Contradicción
Prolog responde: No
Pregunta: alum(luis)

Flujo de trabajo

INTÉRPRETE

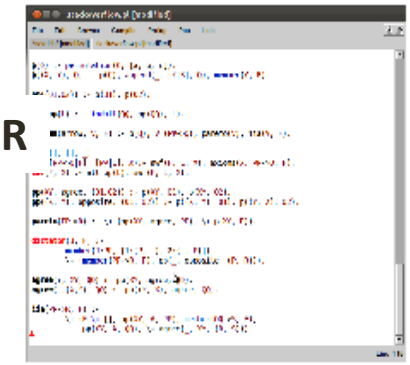


SWI Prolog

BASE DE CONOCIMIENTOS

PREGUNTAS

RESPUESTAS



EDITOR

USUARIO

Programa Prolog

```
/* Prácticas de Matemáticas 1
   Universidad de Alicante
*/

% Programa Prolog
```

Comentarios

```
alum(carlos).
moreno(carlos).
```

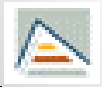
Hechos

Clausulas de Horn

```
buentipo(X) :- alum(X), moreno(X).
macizo(X)   :- buentipo(X), deporte(X), cachas(X).
```

Reglas

BASE DE CONOCIMIENTO



HECHOS

Informan sobre las propiedades /relaciones que afectan a los sujetos del problema

ESTRUCTURA

predicado(arg_1,...arg_n).

predicado: nombre de la propiedad / relación.

- ***Empieza por minúscula.***
- ***No hay espacio entre nombre y paréntesis.***

Al final se pone punto **(.)**

arg_i : argumento del predicado, representa al sujeto afectado por el predicado.

- ***Empieza por minúscula.***
- ***Varios argumentos se separan por comas.***

Proposiciones	HECHOS - PROLOG
Juan es hombre.	hombre(juan).
A Juan le gusta el pan	le_gusta(juan,pan).
Juan y Luis son hombres	eshombre(juan). eshombre(luis).
Juan o Luis es un hombre	NO SE PUEDE REPRESENTAR

PREDICADOS

Predicado: *construcción lógica formada por el nombre de la propiedad / relación y lista de argumentos*

aridad: *número de argumentos del predicado.*

➔ **predicado/1 (aridad 1)**

de propiedad/ cualidad

"Carlos y Juan son alumnos"
alumno(X) : propiedad del sujeto X
alumno(carlos).
alumno(juan).

➔ **predicado/n (aridad n**, n > 1)

de relación

"Carlos y Juan estudian física"
estudia(X,Y) : relación entre los sujetos X, Y.
estudia(carlos, fisica).
estudia(juan, fisica).

En HECHOS
SÓLO
ARGUMENTOS
CONSTANTES

➔ Cadena formada por letras, dígitos y/o símbolo de subrayado (_),

Empiezan siempre con letra **minúscula**.

Ej. válidas: luis, luis1, luis_Lopez.

Ej. no válidas: 1luis, Luis, _luis.

➔ Cadenas **entre comillas simples** permiten espacios, mayúsculas y otros

Ej. 'Luis López', '28003 Madrid'.

PRÁCTICA PROLOG: rufi

Escribir las siguientes proposiciones como HECHOS

Fichero **rufi.pl**

Hipótesis

- **Bertoldo y Bartolo son rufianes.**
- **Romeo y Bertoldo son nobles.**
- **Bartolo es un plebeyo.**
- **Gertrudis y Julieta son damas.**
- **Julieta es hermosa.**
- **Bartolo pelea con un palo y Bertoldo con una espada.**

REGLAS

Es un hecho que para ser cierto depende de que lo sean otros hechos.

$$Q \text{ :- } P_1, P_2, \ldots P_N$$

Se lee: el hecho Q (objetivo) es cierto si son ciertos cada uno de los hechos Pi.

Formalizan proposiciones **condicionales** : $P_1 \wedge P_2 \wedge \ldots P_N \rightarrow Q$

***OjO**: como máximo un literal en el consecuente.*

Ej. **"Si Ana estudia medicina, es feliz"**

fbf: **estudia(ana, medicina) → feliz(ana)**

Prolog: **feliz(ana) :- estudia(ana, medicina).**

*Ana es feliz **si** es cierto que estudia medicina.*

Ej. **"Un sujeto estudia medicina si es feliz"**

feliz(X) :- estudia(X, medicina).

En REGLAS
ARGUMENTOS
CONSTANTES
Y/O
VARIABLES

Representa a un objeto cualquiera del dominio.

- Empieza por **letra mayúscula o subrayado "_"**

  Ej. X, Juan1, _juan

- El símbolo **"_"** es una **variable anónima**.

Una variable puede estar:

- **Sin instanciar**: aún no tiene valor.
- **Instanciada**: tiene asociado un valor.

ESTRUCTURA DE LAS REGLAS

CABEZA **:-** **CUERPO**

máximo **1** hecho

SI

conjunción de hechos y/o

- predicados predefinidos (write/1,…).
- conectivas: conjunción (,), disyunción (;), negación(not).
- Operadores: aritméticos (+…), relacionales. (<,…)

```
Ej: listo(carlos)  :-   alumn(carlos),
                        estudia(carlos, fisica),
                        write('listorro Carlos').
```

```
Ej: listo(X)  :-   alumn(X),
                   estudia(X, fisica),
                   write('listorro'),
                   write(X).
```

Al final poner un punto

Escribir las siguientes proposiciones condicionales como **REGLAS**

Fichero **rufi.pl**

1· Bartolo está cachas **SI** pelea con un palo.

2. Todo el que pelea con un palo está cachas.

3. Los plebeyos desean a Julieta si ésta es una dama.

4. Los plebeyos desean a cualquier dama.

5. Los nobles desean sólo a las damas hermosas.

6· Los rufianes raptan a las personas a las que desean.

7. Toda dama hermosa desea a todo sujeto que pelee con espada.

8· Un sujeto X gana la batalla a un sujeto Y si X pelea con espada e Y con palo.

9. Cualquier dama se deja raptar por un sujeto que gane batallas.

Escribir en 1ª línea **:- write(' nobles y plebeyos').**

TUTORIALES Y APUNTES

Apuntes de Prolog y material (castellano)
Campus virtual /Materiales
http://www.dccia.ua.es/logica/prolog/docs/prolog.pdf
http://www.dccia.ua.es/logica/prolog/material.htm

Adventure in prolog (inglés)
http://www.amzi.com/AdventureInProlog/advfrtop.htm

LIBROS : Buscar en Biblioteca y Archivo, web UA http://www.ua.es

The Art of Prolog
http://gaudi.ua.es/uhtbin/cgisirsi/AkB95t6saS/0/253110069/9

Programación en Prolog
http://gaudi.ua.es/uhtbin/cgisirsi/LO4Ho5QWQ5/0/253110069/9