El Lenguaje de Programación PROLOG

Juan Meza

- PROLOG es un lenguaje de programación declarativo. Los lenguajes declarativos se diferencian de los lenguajes imperativos o procedurales en que están basados en formalismos abstractos (PROLOG está basado en la lógica de predicados de primer orden y LISP, otro lenguaje de programación declarativa, en lambda calculo), y por tanto su semántica no depende de la máquina en la que se ejecutan. Las sentencias en estos lenguajes se entienden sin necesidad de hacer referencia al nivel máquina para explicar los efectos colaterales.
- Un programa escrito en un lenguaje declarativo puede usarse como una especificación o una descripción formal de un problema. Otra ventaja de los programas escritos en lenguajes declarativos es que se pueden desarrollar y comprobar poco a poco, y pueden ser sintetizados o transformados sistemáticamente.

- PROLOG es un lenguaje de programación muy útil para resolver problemas que implican objetos y relaciones entre objetos. Está basado en los siguientes mecanismos básicos, que se irán explicando a lo largo de este capítulo:
  - Unificación
  - Estructuras de datos basadas en árboles
  - Backtracking automático

- La sintaxis del lenguaje consiste en lo siguiente:
  - Declarar hechos sobre objetos y sus relaciones
  - Hacer preguntas sobre objetos y sus relaciones
  - Definir reglas sobre objetos y sus relaciones

Los hechos PROLOG

- Para explicar los fundamentos de PROLOG vamos a utilizar el típico ejemplo de las relaciones familiares. Para decir que Laura es uno de los dos progenitores de Damián, podríamos declarar el siguiente hecho PROLOG:

```
progenitor(laura, damian).
```

- “progenitor” es el nombre de la relación o nombre de predicado y “laura” y “damian” son los argumentos. Los hechos acaban siempre con punto. Nosotros interpretaremos que Laura, primer argumento de la relación, es la madre de Damián, segundo argumento de la relación. Sin embargo, este orden es arbitrario y cada programador puede darle su propio significado. Los nombres de las relaciones y los argumentos que se refieren a objetos o personas concretas se escribirán con minúscula.

le_gusta_a(juan,maria).
valioso(oro).
tiene(juan,libro).
da(juan,libro,maria).

- Los nombres también son arbitrarios y el programador decidirá la interpretación que haga de ellos. La relación le_gusta_a(juan,maria) es equivalente a la relación a(b,c), aunque para que la interpretación sea más sencilla, se recomienda que los nombres se elijan de forma que ayuden a su interpretación. Los hechos no tienen que reflejar el mundo real necesariamente, pero será única y exclusivamente lo que PROLOG tomará como verdadero. Un conjunto de hechos (también llamados cláusulas), junto con un conjunto de reglas, forman lo que se llama una base de datos PROLOG.

Las preguntas PROLOG

- Sobre un conjunto de hechos se pueden realizar una serie de preguntas. Por ejemplo:

?- le_gusta_a(juan,maria).

- PROLOG busca automáticamente en la base de datos si existe un hecho que se puede unificar (es decir, tiene el mismo nombre de predicado, el mismo número de argumentos o aridad y cada uno de los argumentos tiene el mismo nombre, uno a uno) con el hecho que aparece en la pregunta. PROLOG contestará “SI” si encuentra ese hecho y “NO” si no lo encuentra. La contestación “NO” no implica que el hecho sea falso (aunque sí lo sea para nuestra base de datos), sino que no se puede probar (en general) que sea verdadero con el conocimiento almacenado en la base de datos.

- Para realizar preguntas más interesantes, como por ejemplo, qué le gusta a María o cuáles son los padres de Damián, se usarán las variables. En PROLOG las variables empiezan por mayúscula. Por ejemplo:

?-le_gusta_a(maria,X).

?-progenitor(Y,damian).

- Para obtener la o las respuestas, PROLOG recorre la base de datos hasta encontrar el primer hecho que coincide con el nombre de la relación y su aridad y con los argumentos que no son variables. Marca esa posición para poder recordar dónde se quedó en el recorrido por la base de datos. La o las variables se instancian al valor que le corresponde según el lugar que ocupan en la relación, y ese valor es la respuesta que proporciona PROLOG. Si pulsamos RETURN no obtendremos más que la primera respuesta. Si se quieren obtener todas las respuestas (para el caso de que exista más de una) se teclea ";". Cuando pulsamos ";", PROLOG sigue automáticamente la búsqueda desde la marca de posición en la que se había quedado en la base de datos. Se dice entonces que PROLOG intenta resatisfacer la pregunta. Se desinstancian las variables que se habían instanciado, y sigue buscando otro hecho que coincida sintácticamente con la pregunta. A este mecanismo se le llama backtracking, y PROLOG lo hace automáticamente.

- Para resolver preguntas más complejas, como por ejemplo, ¿se gustan Juan y María? o ¿tienen Ana y Damián un progenitor común, es decir, son hermanos? O ¿quién es el nieto(s) de Tomás?, se utilizan conjunciones de objetivos, es decir, preguntas separadas por comas, que en PROLOG corresponden a la “Y” lógica.

?-le_gusta_a(juan,maria), le_gusta_a(maria,juan).

?-progenitor(X,ana), progenitor(X,damian).

?-progenitor(tomas,X), progenitor(X,Y).

- En una conjunción de objetivos correspondientes a la misma pregunta, la misma variable se refiere al mismo objeto (en el segundo ejemplo, X se refiere a la misma persona). En preguntas u objetivos distintos, el mismo nombre de variable se refiere a distintos objetos (la X del segundo y el tercer ejemplo se refieren a personas distintas).

Ejemplo

progenitor(clara,jose).

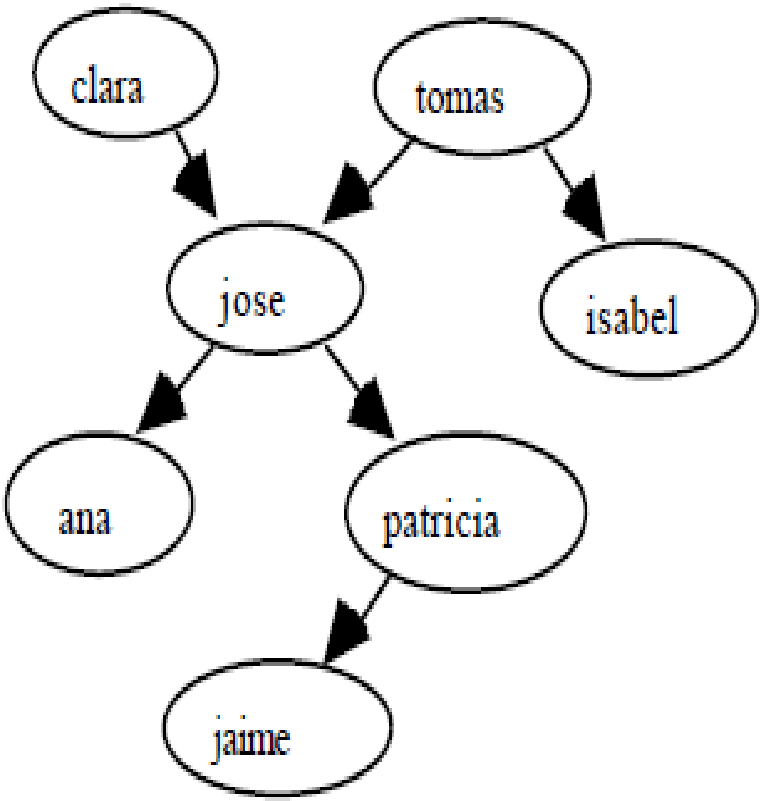
progenitor(tomas, jose).

progenitor(tomas,isabel).

progenitor(jose, ana).

progenitor(jose, patricia).

progenitor(patricia,jaime).



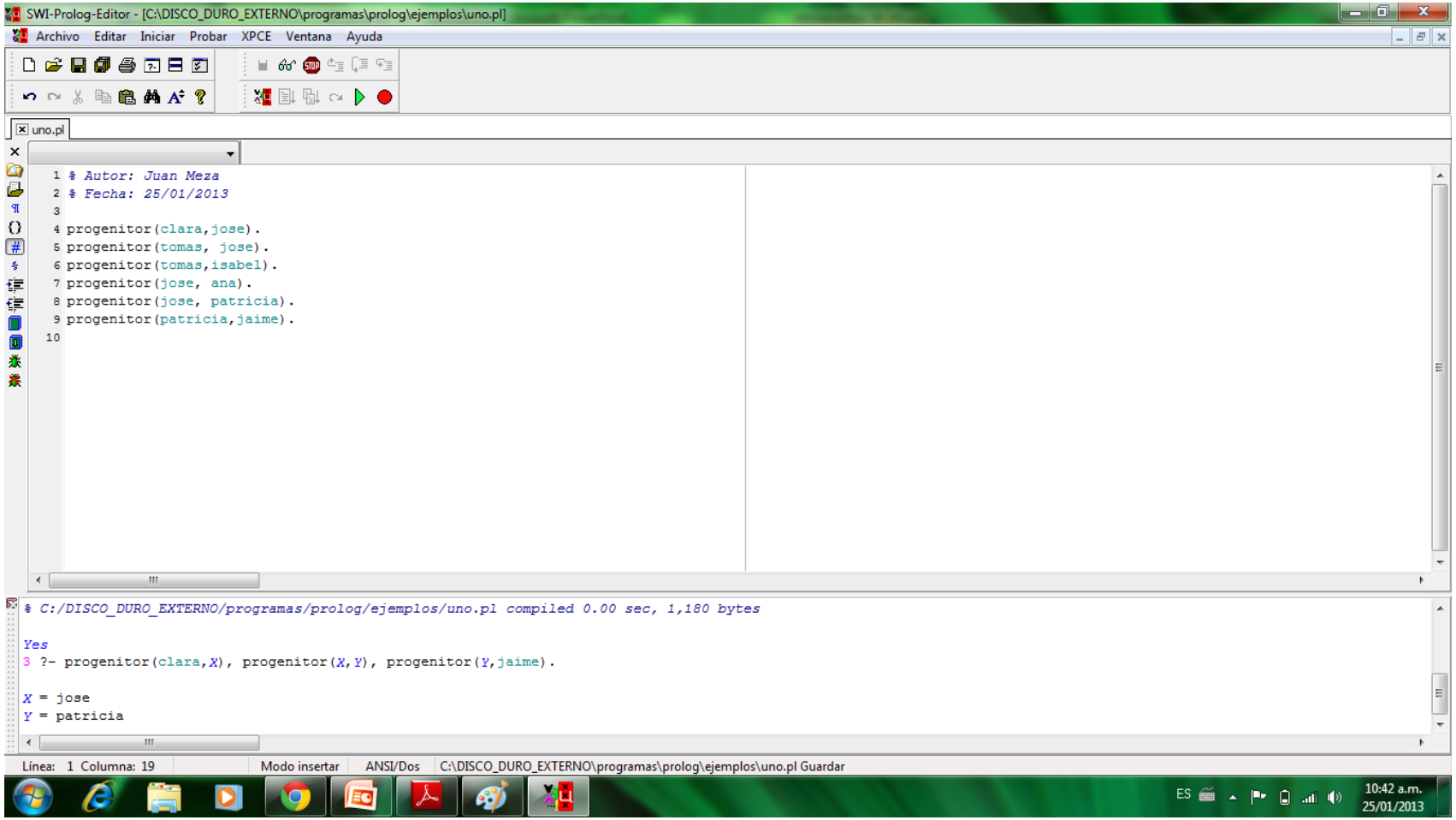
- Para demostrar si Clara es bisabuela de Jaime, utilizaríamos la siguiente conjunción de objetivos:

?-progenitor(clara,X), progenitor(X,Y), progenitor(Y,jaime).

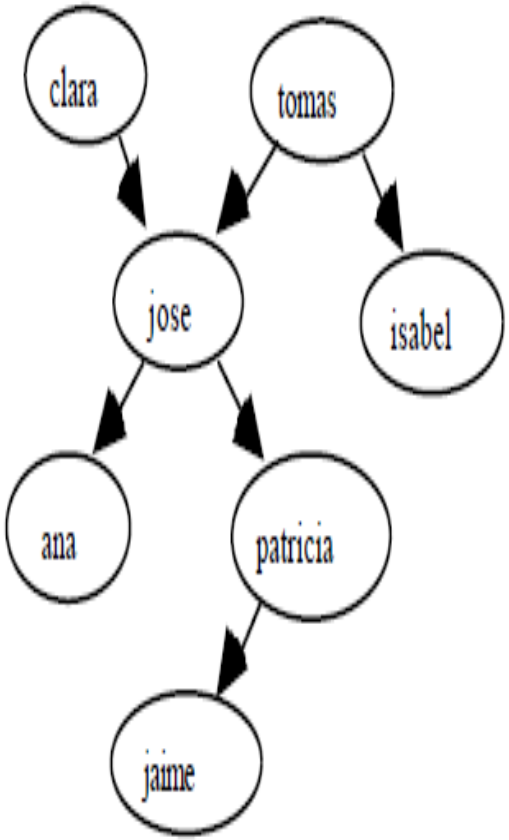
Programa Necesario

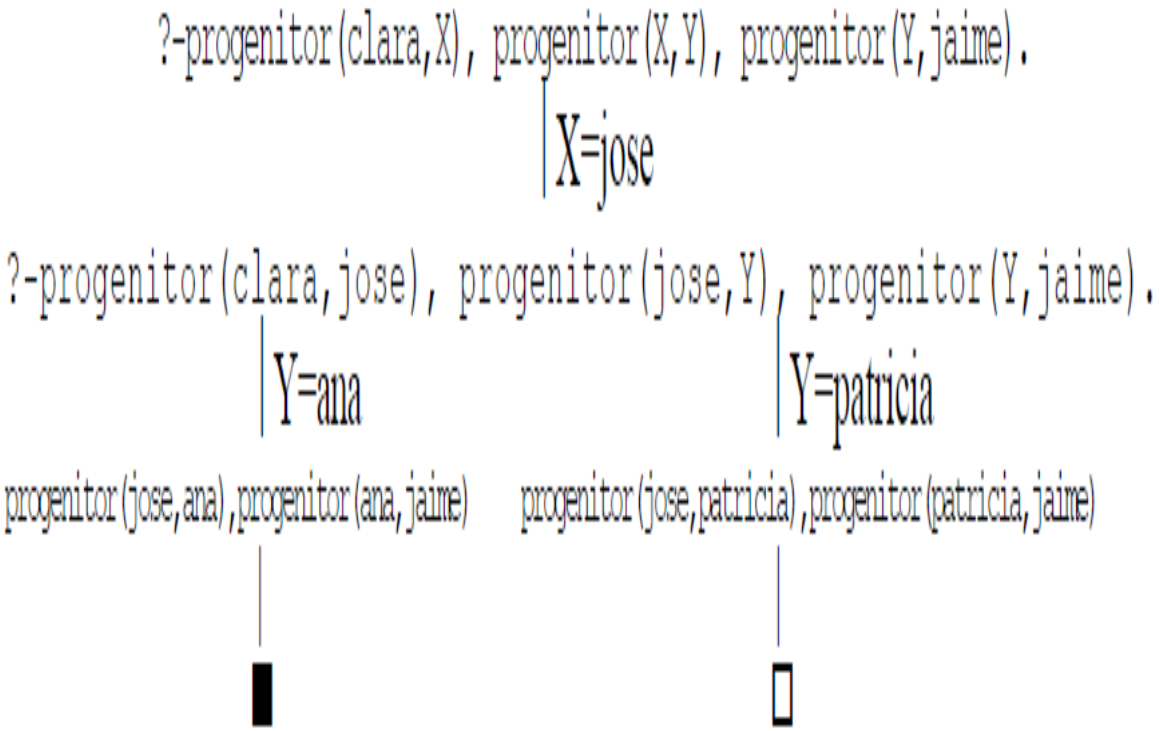
- Ir a la pagina → http://quantum.cucei.udg.mx/~jjme29/PRUEBA/instala_pro.rar

- Es muy importante leer el archivo .txt que esta hay


SWI-Prolog-Editor - [C:\DISCO_DURO_EXTERNO\programas\prolog\ejemplos\uno.pl]
Archivo Editar Iniciar Probar XPCE Ventana Ayuda
uno.pl
1 % Autor: Juan Meza
2 % Fecha: 25/01/2013
3
4 progenitor(clara,jose).
5 progenitor(tomas, jose).
6 progenitor(tomas,isabel).
7 progenitor(jose, ana).
8 progenitor(jose, patricia).
9 progenitor(patricia,jaime).
10
% C:/DISCO_DURO_EXTERNO/programas/prolog/ejemplos/uno.pl compiled 0.00 sec, 1,180 bytes
Yes
3 ?- progenitor(clara,X), progenitor(X,Y), progenitor(Y,jaime).
X = jose
Y = patricia
Línea: 1 Columna: 19
Modo insertar
ANSI/Dos
C:\DISCO_DURO_EXTERNO\programas\prolog\ejemplos\uno.pl Guardar
ES
10:42 a.m.
25/01/2013

- Para explicar el funcionamiento del backtracking que permite obtener todas las respuestas de la anterior conjunción de objetivos, vamos a utilizar el esquema de deducción natural.





X = jose
Y = patricia

- Dada la base de datos familiar del ejemplo anterior, se pide la respuesta de PROLOG y el enunciado verbal de las siguientes preguntas:

?-progenitor(jaime,X).

?-progenitor(X,jaime).

?-progenitor(clara,X), progenitor(X,patricia).

?-progenitor(tomas,X), progenitor(X,Y), progenitor(Y,Z).

- Dada la base de datos familiar del ejemplo angterior, formula en PROLOG las siguientes preguntas:

a) ¿Quién es el progenitor de Patricia?

b) ¿Tiene Isabel un hijo o una hija?

c) ¿Quién es el abuelo de Isabel?

d) ¿Cuáles son los tíos de Patricia? (no excluir al padre)

Las reglas PROLOG

- Existe en PROLOG la posibilidad de definir la relación “abuelo(X,Y)” o la relación “tio(X,Y)” como reglas, además de poderlo hacer como hechos o como conjunción de objetivos.

```
abuelo(X,Y):- progenitor(X,Z), progenitor(Z,Y).
tio(X,Y):- progenitor(Z,Y), progenitor(V,Z), progenitor(V,X).
```

abuelo(X,Y):- progenitor(X,Z), progenitor(Z,Y).

tio(X,Y):- progenitor(Z,Y), progenitor(V,Z), progenitor(V,X).

- A la primera parte de la regla se le llama cabeza o conclusión, el símbolo ":-" es el condicional (SI), y a la parte de la regla que está después de “:-“ es el cuerpo o parte condicional. El cuerpo puede ser una conjunción de objetivos separados por comas. Para demostrar que la cabeza de la regla es cierta, se tendrá que demostrar que es cierto el cuerpo de la regla.

```
progenitor(clara,jose).
progenitor(tomas, jose).
progenitor(tomas,isabel).
progenitor(jose, ana).
progenitor(jose, patricia).
progenitor(patricia,jaime).

abuelo(X,Y):- progenitor(X,Z), progenitor(Z,Y).
tio(X,Y):- progenitor(Z,Y), progenitor(V,Z), progenitor(V,X).
```

- Por lo visto hasta ahora, las cláusulas PROLOG son de tres tipos: hechos, reglas y preguntas. Las cláusulas PROLOG consisten en una cabeza y un cuerpo. Los **hechos son cláusulas que tienen cabeza pero no tienen cuerpo. Las preguntas sólo** tienen cuerpo. Las **reglas tienen siempre cabeza y cuerpo. Los hechos son siempre** ciertos. Las reglas declaran cosas que son ciertas dependiendo de una condición. El programa PROLOG (o base de datos PROLOG) está formado por hechos y reglas y para PROLOG no hay ninguna distinción entre ambas. Las preguntas se le hacen al programa para determinar qué cosas son ciertas.

- Existen dos tipos de cláusulas: Hechos y Reglas. Una regla es del tipo:

Cabeza :- Cuerpo.

- y se lee como "La cabeza es verdad si el cuerpo es verdad".

Ejercicio

- Dada la base de datos familiar del ejemplo primero, y suponiendo definidas las siguientes cláusulas:

hombre(X).
mujer(X).
progenitor(X,Y).
dif(X,Y):- X\=Y.

- Donde las 3 primeras cláusulas se definirán como hechos (por tanto no se podrá poner una variable como argumento, ya que una variable haría que el hecho fuera cierto para cualquier objeto) y la última como una regla (donde el símbolo \= significa distinto). Escribir las reglas de PROLOG que expresen las siguientes relaciones:
  - es_madre(X).
  - es_padre(X).
  - es_hijo(X).
  - hermana_de(X,Y).
  - abuelo_de(X,Y) y abuela_de(X,Y).
  - hermanos(X,Y). Tened en cuenta que una persona no es hermano de sí mismo.
  - tia(X,Y). Excluid a los padres.

- Con la definición del tipo de reglas anterior se pueden resolver problemas interesantes, sin embargo, la gran potencia del PROLOG está en la definición de reglas recursivas. Como hemos visto en ejemplos anteriores se puede definir la relación progenitor, y las reglas abuelo, bisabuelo, tatarabuelo, etc. En general, puede ser interesante definir la relación predecesor(X,Y). Un predecesor de X podrá ser el progenitor de X. También será predecesor si es abuelo/a, si es tatarabuelo/a, etc., es decir, necesitaríamos un conjunto de reglas como:

```
predecesor(X,Y):-progenitor(X,Y).
predecesor(X,Y):-progenitor(X,Z), progenitor(Z,Y).
predecesor(X,Y):-progenitor(X,Z), progenitor(Z,V), progenitor(V,Y).
```

```
progenitor(clara,jose).
progenitor(tomas, jose).
progenitor(tomas,isabel).
progenitor(jose, ana).
progenitor(jose, patricia).
progenitor(patricia,jaime).

abuelo(X,Y):- progenitor(X,Z), progenitor(Z,Y).
tio(X,Y):- progenitor(Z,Y), progenitor(V,Z), progenitor(V,X).

predecesor(X,Y):-progenitor(X,Y).
predecesor(X,Y):-progenitor(X,Z), progenitor(Z,Y).
predecesor(X,Y):-progenitor(X,Z), progenitor(Z,V), progenitor(V,Y).
```

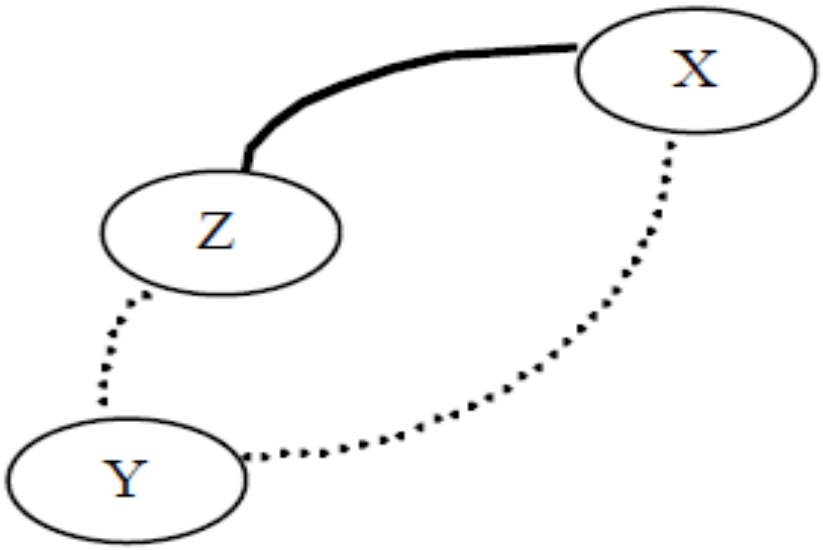
?- predecesor(clara,Y).

Y = jose

- La definición de varias reglas con el mismo nombre de relación equivale en PROLOG a la “O” lógica o disyunción. Pero la definición de este conjunto de reglas es infinito, nunca terminaríamos de escribirlo

predecesor(X,Y):-progenitor(X,Y).

predecesor(X,Y):-progenitor(X,Z), predecesor(Z,Y).



Ejercicio

- Dada la base de datos familiar del primer ejemplo:
  - Define una regla que permita obtener los sucesores de una persona.
  - Comprueba el funcionamiento de PROLOG para obtener los sucesores de Clara.

- Escribir el árbol de derivación natural.
  - ¿Es una alternativa válida a la definición de predecesor la siguiente?

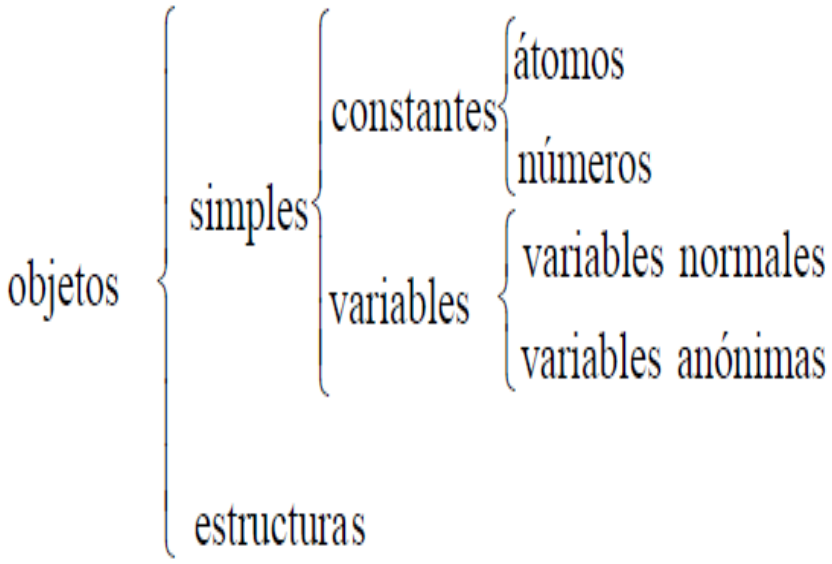
```
predecesor(X,Z):-progenitor(X,Z).
predecesor(X,Z):- progenitor(Y,Z), predecesor(X,Y).
```

- Dibuja un diagrama que explique la definición.

La sintaxis PROLOG

- Los objetos o términos PROLOG pueden ser objetos simples o estructuras. Los objetos simples pueden ser constantes o variables. Las constantes serán átomos o números. Los átomos empiezan con letra minúscula (nunca con números), pueden contener caracteres especiales y pueden ser nombres entre comillas simples. Los números serán enteros o reales, sin una definición explícita de tipos. PROLOG se utiliza para una programación simbólica, no numérica, por eso los enteros se utilizarán por ejemplo para contar el número de elementos de una lista, pero los reales son poco utilizados.

- Las variables empiezan con mayúscula o con subrayado. Las variables anónimas son aquellas cuyo nombre es sólo el carácter subrayado (_). Se usan cuando no es importante el nombre de la variable o cuando la variable no puede unificar con otra, dentro de la misma cláusula



- Por ejemplo:

```
tiene_un_hijo(X):-progenitor(X,Y).
```

- “Y” no unifica con otra variable en la definición de la relación “tiene_un_hijo”, por lo que es aconsejable sustituirla por una variable anónima (en algunos entornos de programación PROLOG se advierte sobre la conveniencia de ello).

```
tiene_un_hijo(X):-progenitor(X,_).
```

- Es importante señalar que el alcance de una variable es la cláusula donde aparece, y el alcance de una constante es todo el programa PROLOG.

- La sintaxis de las estructuras es la misma que la de los hechos. Los **funtores de las** estructuras son los nombres de los predicados de hechos y reglas. Los argumentos de los hechos y las reglas son los componentes de las estructuras.

tiene(juan,libro).

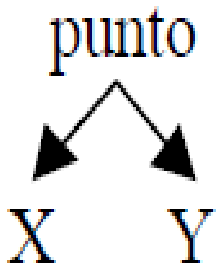
tiene(juan,libro(don_quijote,autor(miguel,cervantes),3256)).

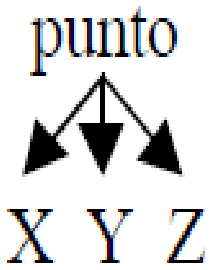
Ejemplo

tiene(juan,libro).

tiene(juan,libro(don_quijote,autor(miguel,cervantes),3256)).

- Las estructuras se pueden representar mediante árboles. Por ejemplo, un punto en 2- dimensiones se puede representar con la estructura punto(X,Y) y un punto en 3- dimensiones con la estructura punto(X,Y,Z). Ambas estructuras son distintas porque tienen distinta aridad o número de argumentos. La estructura en árbol correspondiente sería:





- Un segmento se puede representar con 2 puntos, segmento(punto(X1,Y1), punto(X2,Y2)); y un triángulo con 3 puntos, triangulo(punto(X1,Y1), punto(X2,Y2), punto(X3,Y3)).

- La operación más importante sobre términos es la unificación. Dos términos pueden unificarse si son idénticos o las variables de ambos términos pueden instanciarse a objetos tales que después de la sustitución de las variables por esos objetos, los términos sean idénticos.

fecha(D,M,1998) = fecha(D1,mayo,A)

piloto(A,Londres) = piloto(londres,paris)

punto(X,Y,Z) = punto(X1,Y1,Z1)

f(X,a(b,c)) = f(Z,a(Z,c))

Ejercicio

- Dado el siguiente programa:

```
f(1,uno).
f(s(1),dos).
f(s(s(1)),tres).
f(s(s(s(X))),N):- f(X,N).
```

- ¿Cómo se comporta PROLOG ante las siguientes preguntas?

  a) ?-f(s(1),A).
  b) ?-f(s(s(1)),dos).
  c) ?-f(s(s(s(s(s(1))))),C).
  d) ?-f(D,tres).

Ejercicio

- Dada la siguiente base de datos familiar:

```
progenitor(clara,jose).
progenitor(tomas, jose).
progenitor(tomas,isabel).
progenitor(jose, ana).
progenitor(jose, patricia).
progenitor(patricia,jaime).
mujer(clara).
mujer(isabel).
mujer(ana).
mujer(patricia).
hermana_de(X,Y):- mujer(X), progenitor(Z,X), progenitor(Z,Y).
tia(X,Y):- hermana_de(X,Z), progenitor(Z,Y).
```

- Construir el esquema de deducción natural para las siguientes preguntas:

  a) ?-tia(isabel,ana).
  b) ?-tia(clara,ana).
  c) Si añado la cláusula progenitor(tomas, maria), ¿cómo quedarían a) y b) si pulsamos ;?

Significado declarativo y procedural de los programas

- En un lenguaje declarativo puro, sería de esperar que el orden en el que aparecen los hechos y las reglas en la base fuera independiente de los datos, sin embargo en PROLOG no es así.

- El significado declarativo tiene que ver sólo con las relaciones definidas por el programa. De esta manera, el significado declarativo determina **cuál será la salida** del programa. Por otro lado, el significado procedural determina **cómo se ha** obtenido esta salida; es decir, como evalúa las relaciones PROLOG.

Se describen una serie de hechos conocidos sobre una familia. **Sintaxis de Prolog: Constantes y predicados,** empiezan por minúscula. Los hechos acaban en punto. Variables comienzan por mayúscula.

```
/* Relacion Progenitor */
progenitor(pilar,belen).
progenitor(tomas,belen).
progenitor(tomas,lucia).
progenitor(belen,ana).
progenitor(belen,pedro).
progenitor(pedro,jose).
progenitor(pedro,maria).
```

El programa anterior debe ser cargado en el sistema. Una vez cargado, es posible realizar preguntas:

progenitor(pilar,belen).

Yes: Se puede deducir y el objetivo no tiene variables.

progenitor(pilar,lucia).

No: No puede deducir a partir del programa La respuesta *No indica que no se puede deducir.*

progenitor(belen,X).

Substitución de Respuesta: Se puede deducir y el objetivo tiene variables. Se indica el valor que toman las variables en la resolución del objetivo.

```
/* Reglas */
cuida(belen,pedro):-paro(belen),bueno(pedro).
/* 2 hechos más */
paro(belen).
bueno(pedro).
```

Reglas con Variables

```
madre(X,Y):-mujer(X),progenitor(X,Y).
mujer(pilar).
mujer(belen).
mujer(lucia).
mujer(ana).
mujer(maria).
hombre(tomas).
hombre(pedro).
hombre(jose).
```

- *Para todo X e Y, si X es mujer y X es el progenitor de Y, entonces X es la madre de Y*

Preguntas

madre(belen,pedro).

madre(X,belen).

madre(belen,X).

madre(X,Y).

La programación lógica basa su modelo en la utilización de relaciones, lo cual permite que un mismo procedimiento sirva para diferentes propósitos dependiendo de qué variables están instanciadas2

En el ejemplo, una misma regla *madre sirve para:* Comprobar si *belen es madre de pedro.* Calcular la *madre de belén,* Calcular los hijos de *belén,* Calcular parejas de madres/hijos

Obsérvese que las variables de los objetivos, corresponden a cuantificadores existenciales

- Si tenemos un conjunto de hechos y reglas con el mismo nombre de relación y la misma aridad, puede ser conveniente que los hechos estén situados en la base de datos antes que las reglas, (sobre todo, si los hechos son excepciones de las reglas). Además también suele ser aconsejable poner la regla para salirse de la recursividad antes que la regla recursiva.

- La habilidad de PROLOG para calcular de forma procedural es una de las ventajas específicas que tiene el lenguaje. Como consecuencia esto anima al programador a considerar el significado declarativo de los programas de forma relativamente independiente de su significado procedural.

Reglas Recursivas

```
antepasado(X,Y):-progenitor(X,Y).
antepasado(X,Y):-progenitor(X,Z),
antepasado(Z,Y).
```

En general, en una definición recursiva, es necesario considerar 2 casos:

Caso básico: Momento en que se detiene la computación

Caso Recursivo: Suponiendo que ya se ha solucionado un caso más simple, cómo descomponer

el caso actual hasta llegar al caso simple.

Tanto el caso básico como el caso recursivo no tienen porqué ser únicos (puede haber varios casos básicos y varios casos recursivos)

Preguntas

antepasado(belen,X).

antepasado(X,belen).

Las definiciones recursivas se resuelven de la misma forma que las reglas comunes. En la traza de este tipo de definiciones tiene especial importancia el renombramiento de variables.

Considerando la relación *progenitor como un* enlace entre dos nodos de un grafo. La relación *antepasado indicaría si hay camino entre dos nodos* del grafo dirigido acíclico formado por la relación *progenitor. Este tipo de relaciones se utiliza en* diversos contextos como la búsqueda de caminos entre ciudades, la simulación de movimientos de un autómata, etc

Utilización de funciones

```
grande(pepe).
grande(cabeza(juan)).
grande(X):-mayor(X,Y).
mayor(cabeza(X),cabeza(Y)):-
    progenitor(X,Y).
```

Se utiliza la función: *cabeza(x)=”cabeza de x”*

El programa indica: *“Pepe es grande, la cabeza de juan es grande, si X es*

mayor que Y, entonces X es grande, además: La cabeza de X es mayor que la de Y si X es el progenitor de Y”

Preguntas

grande(X).

Las variables en Prolog no tienen tipo, de ahí que la respuesta X puede ser una persona (*pepe)* o *una* cabeza (*cabeza(juan))*

Datos Compuestos

horizontal(seg(punto(X,Y), punto(X1,Y))).

vertical(seg(punto(X,Y), punto(X,Y1))).

punto(X,Y) representa un punto de coordenadas (x,y) seg(p1,p2) representa un segmento cuyos extremos son los puntos p1 y p2 Los argumentos de una función pueden ser funciones

Preguntas

horizontal(seg(punto(1,2),punto(3,2))).

P = punto(_47796,2) indica que P es un punto cuya primera coordenada es una variable sin instanciar3 y cuya segunda coordenada es 2 La última respuesta indica que para que un segmento sea vertical y horizontal a la vez, sus coordenadas deben ser las mismas (los números de las variables X e Y coinciden)

- Es decir, las ventajas de la forma declarativa de este lenguaje son claras (es más fácil pensar las soluciones y muchos detalles procedurales son resueltos automáticamente por el propio lenguaje) y podemos aprovecharlas. Los aspectos declarativos de los programas son, habitualmente, más fáciles de entender que los procedurales. Esta es la principal razón por la que el programador debe concentrarse en el significado declarativo y evitar distraerse por los detalles de cómo se ejecutan los programas.

- El acercamiento a la programación declarativa no es suficiente. Una vez se tenga el concepto claro, y cuando se trabaje con programas grandes, los aspectos procedurales no se pueden ignorar por completo por razones de eficiencia. No obstante, el estilo declarativo a la hora de pensar en PROLOG debe ser estimulado y los aspectos procedurales ignorados para favorecer el uso de las restricciones.

Ejercicio

- Construir el árbol de resolución lineal para la pregunta:

?-predecesor(clara,patricia).

- teniendo en cuenta las siguientes 4 definiciones de predecesor:

a) predecesor(X,Y):-progenitor(X,Y).
predecesor(X,Y):-progenitor(X,Z), predecesor(Z,Y).

b) predecesor(X,Y):-progenitor(X,Z), predecesor(Z,Y).
predecesor(X,Y):-progenitor(X,Y).

c) predecesor(X,Y):-progenitor(X,Y).
predecesor(X,Y):-predecesor(Z,Y), progenitor(X,Z).

d) predecesor(X,Y):-predecesor(Z,Y), progenitor(X,Z).
predecesor(X,Y):-progenitor(X,Y).

```
progenitor(clara,jose).
progenitor(tomas, jose).
progenitor(tomas,isabel).
progenitor(jose, ana).
progenitor(jose, patricia).
progenitor(patricia,jaime).
mujer(clara).
mujer(isabel).
mujer(ana).
mujer(patricia).
hermana_de(X,Y):- mujer(X), progenitor(Z,X), progenitor(Z,Y).
tia(X,Y):- hermana_de(X,Z), progenitor(Z,Y).

predecesor(X,Y):-progenitor(X,Y).
      predecesor(X,Y):-progenitor(X,Z), predecesor(Z,Y).
```

Ejemplo

```
%%
%% declaraciones
%%
padrede('juan', 'maria'). % juan es padre de maria
padrede('pablo', 'juan'). % pablo es padre de juan
padrede('pablo', 'marcela').
padrede('carlos', 'debora').

% A es hijo de B si B es padre de A
hijode(A,B) :- padrede(B,A).
% A es abuelo de B si A es padre de C y C es padre B
abuelode(A,B) :-
  padrede(A,C),
  padrede(C,B).
% A y B son hermanos si el padre de A es también el padre de B y si A y B no son lo mismo
hermanode(A,B) :-
  padrede(C,A) ,
  padrede(C,B),
  A \== B.

% A y B son familiares si A es padre de B o A es hijo de B o A es hermano de B
familiarde(A,B) :-
  padrede(A,B).
familiarde(A,B) :-
  hijode(A,B).
familiarde(A,B) :-
  hermanode(A,B).
```

% juan es hermano de marcela?

?- hermanode('juan', 'marcela').

yes

% carlos es hermano de juan?

?- hermanode('carlos', 'juan').

No

% pablo es abuelo de maria?

?- abuelode('pablo', 'maria').

yes

% maria es abuela de pablo?

?- abuelode('maria', 'pablo').

no

Factorial de un número

% La sintaxis es factorial(N, F) -> Factorial de N es F (el resultado se guarda en F)

```
 factorial(0, 1).
factorial(N, F) :- N>0, N1 is N - 1, factorial(N1, F1), F is N * F1.
```

%el factorial se llama recursivamente dejando el resultado en F

```
 ?- factorial(2, F).
F = 2
```

Expresiones

- Prolog cuenta con operadores para la unificación y comparación, sea con evaluación o sea simbólica, como los siguientes:
  - **X is Y** %unificación con evaluación.
  - **X = Y** %unificación simbólica
  - **X=:=Y** %comparación con evaluación
  - **X == Y** %comparación simbólica.

?- X is 3+5.

X = 8

?- X = 3+5.

X = 3+5

?- 3+5 =:= 2+6.

yes

?- 3+5 == 2+6.

no

?- 3+5 == 3+5.

yes

Listas en PROLOG

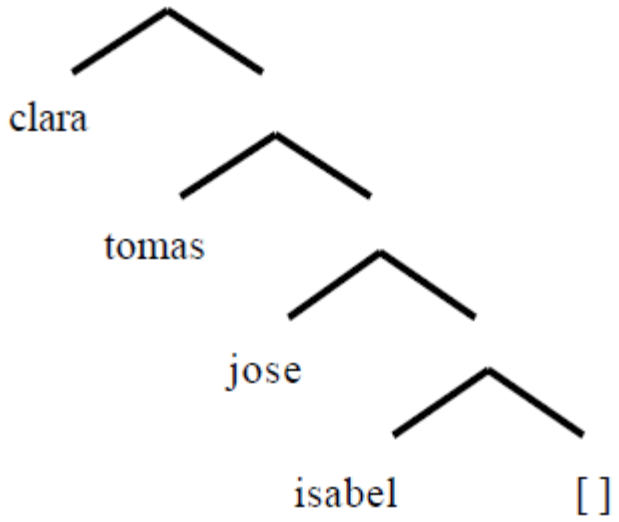
- La lista es una estructura de datos simple, muy usada en la programación no numérica. Una lista es una secuencia de elementos tales como:

clara,tomas,jose,isabel

- La sintaxis de PROLOG para las listas consiste en englobar las secuencias de elementos entre corchetes.

[clara,tomas,jose,isabel]

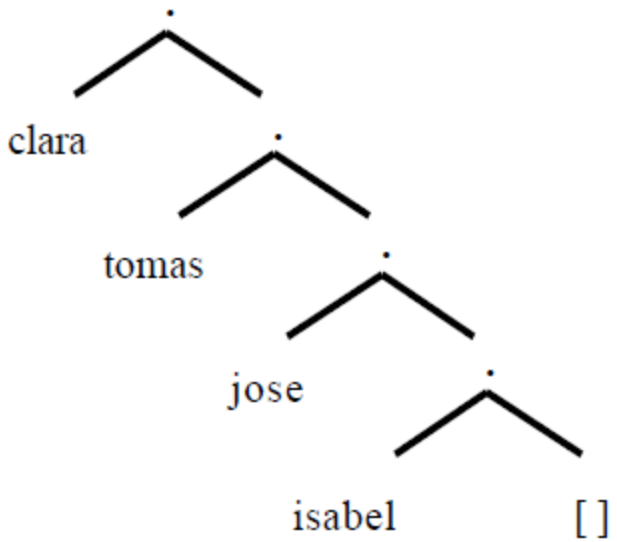
- La representación interna de las listas en PROLOG es con árboles binarios, donde la rama de la izquierda es el primer elemento –o cabeza– de la lista y la rama de la derecha es el resto –o cola– de la lista. De la rama que contiene el resto de la lista también se distinguen dos ramas: la cabeza y la cola. Y así sucesivamente hasta que la rama de la derecha contenga la lista vacía (representado por [])



- De cualquier forma consideraremos dos casos de listas: Las listas vacías y las no vacías. En el primer caso escribiremos la lista vacía como un átomo de PROLOG, []. Y en el segundo caso, la lista la consideraremos formada por dos partes; el primer elemento, al que llamaremos la cabeza de la lista; y el resto de la lista al que llamaremos la cola de la lista. En el ejemplo anterior, la cabeza será clara y la cola [tomas,jose,isabel].

- Aunque como hemos visto anteriormente no es necesario, en general la estructura lista se representa en PROLOG con el nombre de predicado o funtor “.”.

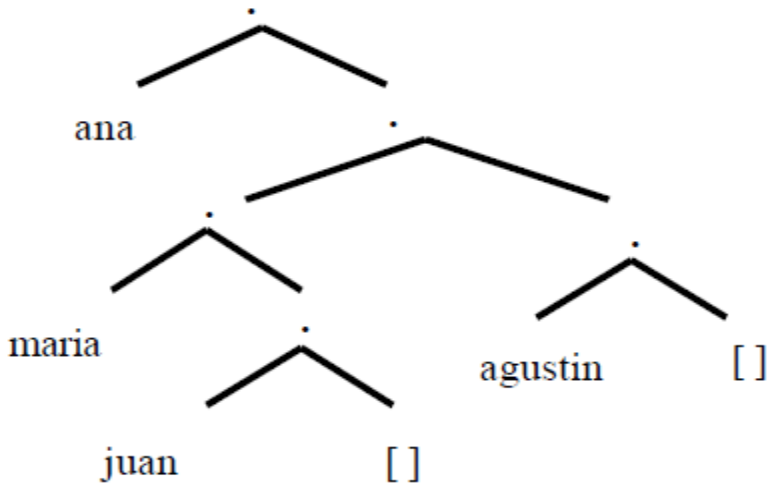
- .(clara,.(tomas,.(jose,.(isabel,[])))) Que PROLOG representará internamente como un árbol:



- El último elemento siempre es la lista vacía ([]).

- Como en PROLOG el manejo de listas es muy usual, se utiliza la notación simplificada con corchetes, que es equivalente a la anterior ([clara, tomas, jose, isabel]). Interiormente PROLOG lo interpretará con la notación árbol. Los elementos de una lista pueden ser de cualquier tipo, incluso otra lista.

- [ana, [maria, juan], agustin] PROLOG representará internamente como un árbol:



- La cabeza y la cola de una lista se pueden separar con el símbolo “|”.

- [] es una lista
- Si Xs es una lista entonces [X|Xs] es una lista

- [] denota la lista vacía
- [X|Xs] denota una lista de cabeza X y cola Xs

- Las siguientes son expresiones válidas sobre listas y se refieren a la misma lista:

[a,b,c] [a|[b,c]] [a,b|[c]] [a,b,c|[]] [a|X],[Y|[b,c]]

- El orden de los elementos en la lista importa y un elemento se puede repetir en una lista, de forma similar a como ocurre en las tuplas y pares ordenados.

Ejemplos de solución de problemas de listas en PROLOG

- La práctica en el manejo de listas con PROLOG es básica para el aprendizaje del lenguaje. Para resolver un problema de listas con PROLOG, primero pondremos varios ejemplos de su funcionamiento, para determinar los argumentos que son necesarios en la definición de la relación y la respuesta de PROLOG en distintos casos, que nos indicará el comportamiento deseado de la cláusula.

- Supongamos que queremos determinar si un elemento es miembro de una lista. Los siguientes serían ejemplos del funcionamiento de la relación “miembro”.

```
miembro(b,[a,b,c]). %PROLOG respondería sí.
miembro(b,[a,[b,c]]). %PROLOG respondería no.
miembro([b,c],[a,[b,c]]). %PROLOG respondería sí.
```

- El símbolo “%” permite poner comentarios que ocupen sólo la línea donde aparece dicho símbolo. Para comentarios más largos se puede utilizar los símbolos “/*” para empezar y “*/” para terminar. La siguiente podría ser una definición de la relación “miembro”:

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

```
cursa(juan, [matematicas, historia, computacion]).
cursa(ana, [matematicas, arte, historia]).
cursa(pedro, [geologia, logica, geografia]).
cursa(maria, [logica, arte, computacion]).
cursa(jose, [logica, historia, geografia]).

?- cursa(juan,X).
X = [matematicas, historia, computacion]

?- cursa(X,[_, historia, computacion]).
X = juan

?- cursa(X,[_, _, computacion]).
X = juan ;
X = maria ;
```

Creación y consulta de listas

plantas([manzana, naranja, limon, espinaca, gardenia, alfalfa, pino]).
lista([1,2,3]).

?-lista([H|T]).
H=1 T=[2,3]

?-lista([H,J|T]).
H=1
J=2
T=[3]

Longitud de una lista

% Si queremos hallar la longitud de una lista.
% La longitud de una lista vacia es 0.
% La longitud de cualquier lista es la longitud de la cola + 1.

```
longitud([],0).
longitud([_|T],N):-longitud(T,N0), N is N0 + 1.
```

```
 ?- longitud([a,b,c],L).
L = 3
```

```
?- longitud([a,b,c],4).
 No
```

Búsqueda de un elemento

```
% Si queremos determinar si un elemento pertenece a una lista.
 % El elemento pertenece a la lista si coincide con la cabeza de la lista.
% El elemento pertenece a la lista si se encuentra en la cola de la lista.
pertenece(X,[X|_]) .
 pertenece(X,[_|R]):- pertenece(X,R).

?- pertenece(b,[a,b,c]).
Yes

?- pertenece(b,[a,[b,c]]).
No

?- pertenece([b,c],[a,[b,c]]).
Yes

?- pertenece(X,[a,b]).
X = a ;
 X = b
```

Eliminar elemento de una lista

```
% Si queremos eliminar un elemento de la lista.
% Si X es la cabeza de la lista, la cola T es la lista sin X
 % Si X no es la cabeza de la lista, conservamos la cabeza de la lista
 % como parte de la respuesta y continuamos eliminando X de la cola T.
 elimina(X,[X|T],T).
 elimina(X,[H|T],[H|T1]):- elimina(X,T,T1).

 ?- elimina(1,[1,2,3,4],R).
 R = [2,3,4]

 ?- elimina(1,R,[2,3]).
R = [1, 2, 3] ;
 R = [2, 1, 3] ;
 R = [2, 3, 1]

 ?- elimina(X,[1,2,3],Y).
 X = 1, Y = [2,3] ;
 X = 2, Y = [1,3] ;
 X = 3, Y = [1,2]
```

Concatenar listas

```
% Si queremos concatenar dos listas lista.
% Concatenar una lista vacia con L es L.
% Concatenar X|L1 con L2 es poner el primer
% elemento de la primera lista (X) más la
% concatenación del resto de la lista (L1) con L2
concatenar([],L,L).
 concatenar([X|L1],L2,[X|L3]):-concatenar(L1,L2,L3).

 ?- concatenar([1,2],[3,4],R).
 R = [1, 2, 3, 4].

?- concatenar(X,Y,[1,2]).
 X = [], Y = [1,2] ;
 X = [1], Y = [2] ;
 X = [1,2], Y = []
```

Comprobar si una lista es la inversa de otra

```
% Si queremos calcular la inversa de una lista.
% La inversa de una lista vacia es una lista vacia.
% La inversa de H|T es la inversa de T concatenada con H.
concatenar([],L,L).
concatenar([X|L1],L2,[X|L3]):-concatenar(L1,L2,L3).
inversa([],[]).
 inversa([H|T],L):- inversa(T,R), concatenar(R,[H],L).

 ?- inversa([a,b,c,d],[d,c,b,a]).

 Yes/Si
```

Ejercicio

- Utilizar la definición de concatenación para:
  a) Descomponer una lista en dos, obteniendo todas las posibles descomposiciones.
  b) Borrar algunos elementos de una lista.
- Por ejemplo: considera la lista L1=[a,b,c,d,z,z,e,z,z,z,f,g] y borra todos los elementos que siguen a la secuencia z,z,z.
  c) Borrar los tres últimos elementos de una lista.
  d) Definir de nuevo la operación miembro.
  e) Definir la relación para extraer el último elemento de la lista
    e.1) Utilizando la definición de concatenación.
    e.2) Sin utilizarla.