

PROLOG - I

Introducción

- PROLOG (“PROgrammation en LOGique”).
- Inteligencia Artificial.
- Historia ...
 - Alain Colmerauer y sus colaboradores alrededor de **1970** en la Universidad de Marseille-Aix.
 - Robert Kowalski proporcionó el marco teórico.
 - David Warren, de la Universidad de Edimburgh, desarrolló el primer compilador de Prolog (WAM – “Warren Abstract Machine”).

Introducción

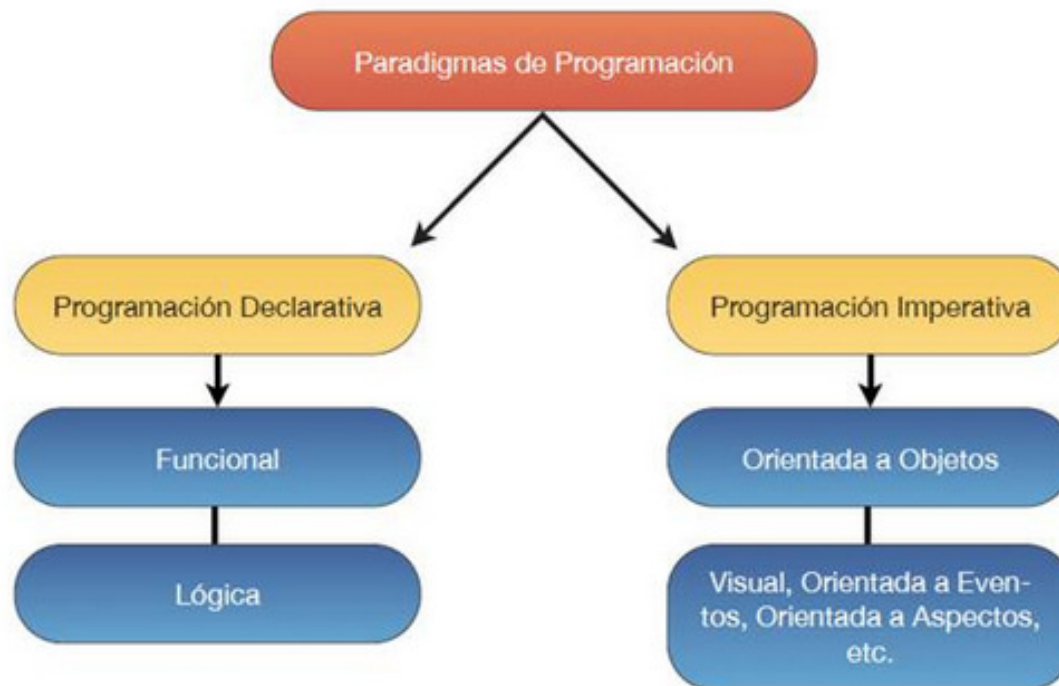
- SWI Prolog interpreter <http://www.swi-prolog.org/>
- Entorno de desarrollo: Eclipse <http://www.eclipse.org> con el plugin <http://prodevtools.sourceforge.net/>
- Learn Prolog Now! <http://www.learnprolognow.org>
- <http://comp.mq.edu.au/units/comp248/resources/brna-prolog-book.pdf>
- <http://www.ida.liu.se/~ulfni/lpp/>
- <http://staff.science.uva.nl/~ulle/teaching/prolog/>
- W. Clocksin, C. Mellish, “Programming in PROLOG”, Springer–Verlag. Libro que describe, de facto, el estándar de Prolog.
- W. Clocksin, “Clauses and Effects”, Springer–Verlag.
- L. Sterling, E. Shapiro, “The Art of Prolog”, 3th Ed, MIT Press, 1999.

Introducción

- Primer intento de diseñar un lenguaje de programación que posibilitara al programador **especificar sus problemas en lógica**.
- Se basa en el lenguaje de la **Lógica de Primer Orden (lógica de predicados)**.
- Se utiliza para resolver problemas en los que entran en juego **objetos y relaciones** entre ellos.
- Solemos usar **reglas** para describir relaciones: "dos personas son hermanas si ambos son hembras y tienen los mismos padres".
- **“Pepe tiene una casa”**
 - Estamos expresando una **relación (tiene)** entre un **objeto (Pepe)** y otro **objeto (una casa)**.
 - Estas relaciones tienen un **orden específico** (Pepe posee la casa y no al contrario).
 - **¿Tiene Pepe una casa?** → estamos indagando acerca de una relación.

Introducción

- En la programación lógica se especifica **qué** se tiene que hacer (**programación declarativa**), y no **cómo** se debe hacer (**programación imperativa**).
- Se describe un problema, no una solución.



Introducción



Diferencia entre imperativo y declarativo [\[editar \]](#)

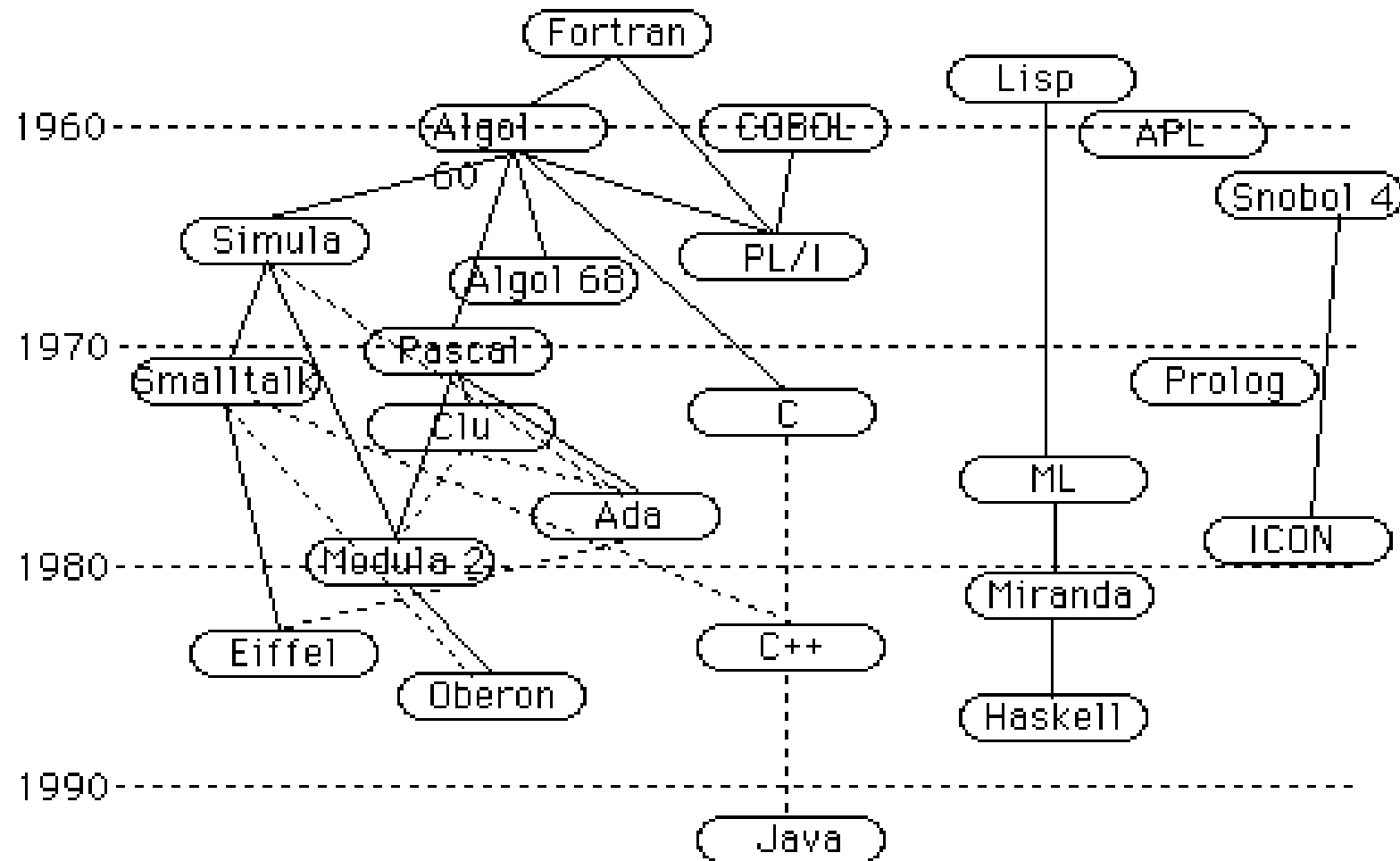
En la [programación imperativa](#) se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un [algoritmo](#) en el que se **describen los pasos necesarios para solucionar el problema**.

En la programación declarativa las sentencias que se utilizan lo que hacen es **describir el problema** que se quiere solucionar; se programa diciendo lo que se quiere resolver a nivel de usuario, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de [inferencia](#) de información a partir de la descripción realizada.

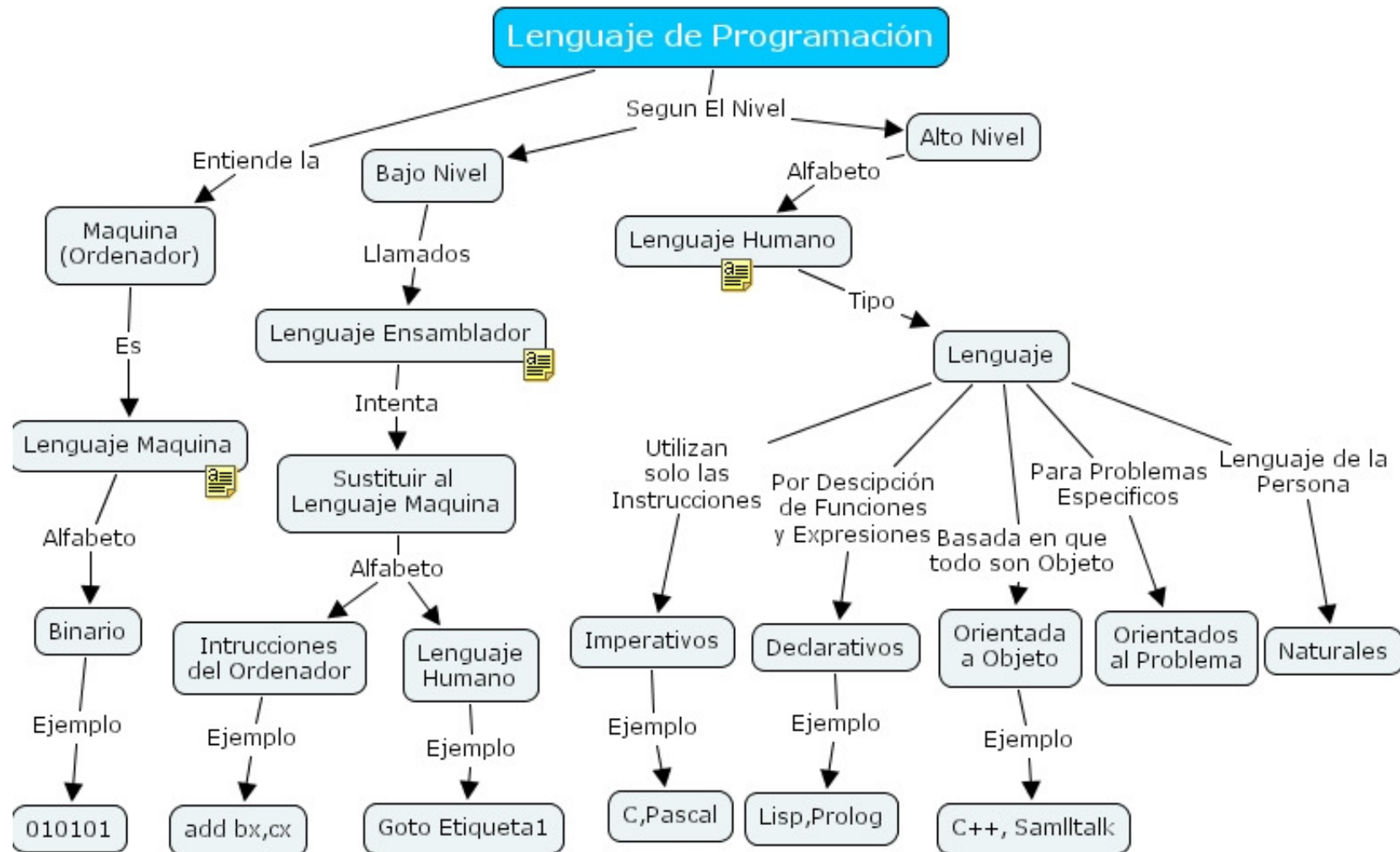
Introducción

- **PROLOG es un lenguaje de programación declarativa basado en la lógica de predicados de primer orden.**
- Hay otros lenguajes , de programación declarativa como por ejemplo:
 - *Haskell (Programación funcional)*
 - *ML (Programación funcional)*
 - *Lisp (Programación funcional)*
 - *F-Prolog (Programación lógica difusa)*
 - *Curry (Programación lógico-funcional)*
 - *ABSET*
 - *Lustre*
 - *MetaPost*
 - *QML*
 - ...

Introducción



Introducción



Introducción

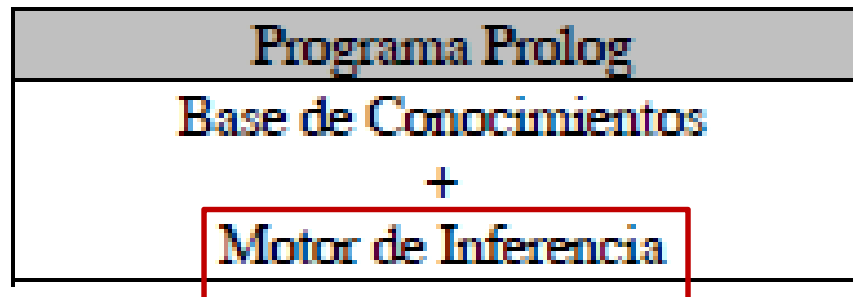
- Un programa Prolog es un conjunto de hechos y reglas que expresan cierto conocimiento mediante lógica de primer orden.
- La sintaxis del lenguaje consiste en lo siguiente:
 - Declarar **hechos** sobre **objetos** y sus **relaciones**.
 - Definir **reglas** sobre objetos y sus relaciones.
 - Hacer **preguntas** sobre objetos y sus relaciones.

Introducción

- ¿Cómo responde Prolog a las preguntas formuladas por el programador?
→ Prolog consulta la **base de conocimiento (hechos + reglas)**.
- Al iniciar una sesión Prolog → la base de conocimiento almacena un conocimiento básico que incluye, entre otras cosas, conceptos y definiciones de la aritmética de los números naturales.
- Es posible ampliar esta base de conocimiento añadiendo definiciones de conceptos sobre los que Prolog no sabe nada.
- **La base de conocimiento (hechos y reglas Prolog) es una representación sintáctica concreta de cláusulas de Horn de primer orden.**

Introducción

- Declarar algunos HECHOS sobre los objetos y sus relaciones,
- Definir algunas REGLAS sobre los objetos y sus relaciones, y
- Hacer PREGUNTAS sobre los objetos y sus relaciones.



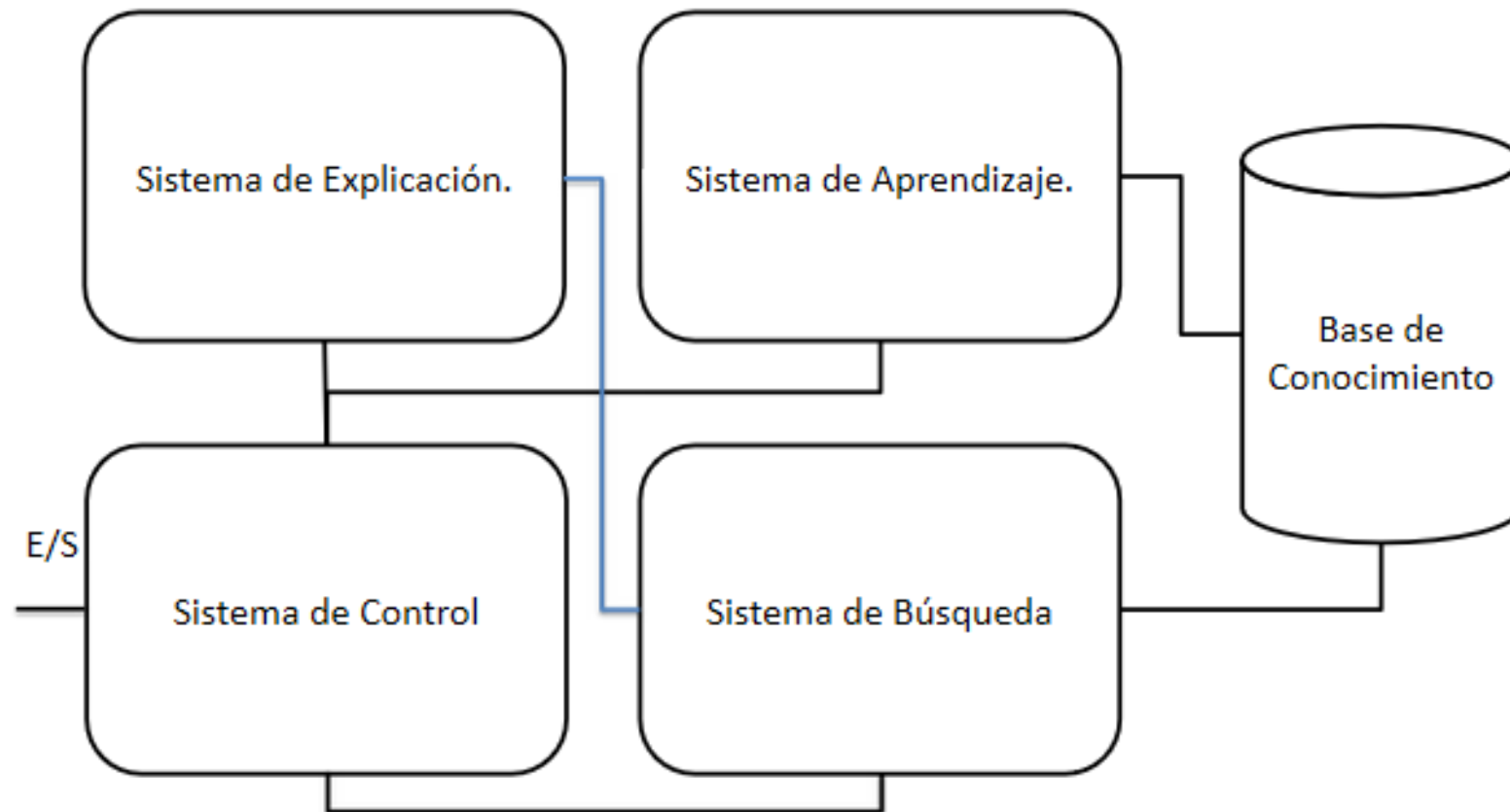
Introducción

Motor de Inferencia


- *De manera general, podemos definirlo como el componente encargado de administrar y controlar de forma lógica el uso y utilización del conocimiento (que viene expresado en la base de conocimiento).*
- *Distintos motores de inferencia implementan distintos modelos, esto es, distintas estrategias de búsqueda sobre la base de conocimiento para obtener el conocimiento buscado (responder a la pregunta planteada).*
- *Tareas:*
 - *Examina los hechos y reglas.*
 - *Añade, si es posible, nuevos hechos.*
 - *Decide orden de inferencias.*
 - *Verifica consistencias (datos, hechos, ...).*
 - *Actualiza base de conocimiento.*

Introducción

Motor de Inferencia



Entorno

 SWI-Prolog (Multi-threaded, version 7.2.3)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit <http://www.swi-prolog.org> for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-

Entorno

- Prolog es un lenguaje “conversacional”
 - *Diálogo con el usuario/programador con el sistema.*
 - *En forma interrogativa.*
- ?- pregunta. <Enter>
- ?- respuesta

Entorno

- En una sesión de trabajo, PROLOG asume que todo lo que no se declare explícitamente (hechos o conclusiones que puedan demostrarse usando las reglas) es falso.
- Formalmente, PROLOG va buscando hechos que se unifiquen con el objetivo.
- En PROLOG, la coma representa el conectivo \wedge (and).
- En general, una pregunta [query] puede ser más compleja que una simple consulta a la base de datos. Por ejemplo, puede ser una conjunción de átomos:
?- esHombre(juan), esHombre(pedro) (*¿son Juan y Pedro hombre?*)
yes
- PROLOG intentará demostrar los objetivos en el orden especificado. Si no puede demostrar alguno de ellos, devolverá un "no". Si puede demostrarlos todos, devolverá un "yes".

Entorno

Tipos de preguntas

Sin variables

- La respuesta es, simplemente, yes o no.
?- esHombre(juan), esHombre(pedro).
yes

Con variables

- Cuando estamos interesados en obtener todos los objetos que cumplen un objetivo.

?-esHombre(**Persona**).

Persona = juan ;

Persona = pedro ;

no

El punto y coma (;), introducido por el usuario, le indica al intérprete de PROLOG que siga buscando respuestas. El punto (.) se emplea cuando ya no nos interesa obtener más respuestas

Entorno

Tipos de preguntas

Con variable anónima

- En el objetivo, la variable anónima hay que verla como una variable libre, sin ninguna cuantificación, que forzará una única sustitución: la primera que se encuentre.
- **Haremos preguntas con la variable anónima (_), cuando estamos interesados en saber si existe algún objeto que cumpla un objetivo.**
- La respuesta de PROLOG es, simplemente, yes o no

?- esHombre(_).

yes

Entorno

- Algunos ejemplos de uso
 - ?- 5 is 3+2. (". AL FINAL OBLIGATORIO)
 - ?- true / yes
 - ?- 5 is 3+3.
 - ?- false / no
 - ?- 5 is 3+2
 - ?- | (FALTA ". AL FINAL → QUEDA ESPERANDO)

Entorno

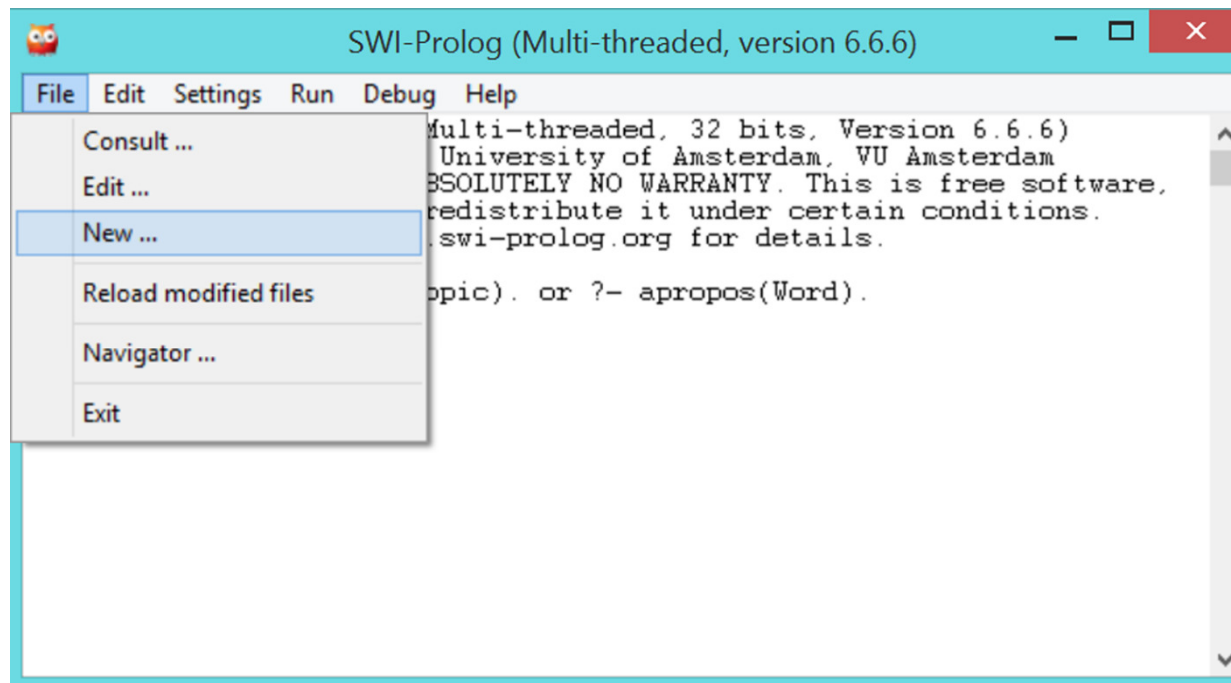
- Algunos ejemplos de uso
 - ?- 5 is 3 R 2.
 - ?- ERROR: Syntax error: Operator expected
 - ?- ERROR: 5
 - ?- ERROR: ** here **
 - ?- ERROR: IS 3 r 2 .
 - ?- **<ctrl + d> (abortar)**

Entorno: directorios y ficheros

- Programas Prolog → ficheros de texto con **extensión '.pl'**.
- Notación diferente a sistemas tipo Win/DOS para representar las rutas de los ficheros. Win/DOS → carácter '\'. Prolog → carácter '/'.
- Conjunto de **predicados especiales para navegar** por el sistema de ficheros y visualizar los directorios.
- **pwd.** muestra directorio actual
- **ls.** visualiza contenido directorio actual
- **cd('x').** permite cambiar de directorio
 - **cd('..').** (cambia al directorio anterior del actual)
 - **cd('c:/trabajo/prolog').** (cambia a este directorio)
- **delete('fichero').** borra el fichero especificado. Entre ' '. Debe tener ext .pl
- **exists_file('fichero').** borra el fichero especificado.
- **rename_file('fichero1', 'fichero2').** Renombra fichero

Entorno: programas

- Cualquier programa en Prolog tiene que estar escrito en un archivo de texto plano sin formato (Bloc de Notas, ...).
- El archivo debe poseer la extensión ".pl" → código fuente de Prolog.
- Un programa Prolog está formado con un conjunto de hechos y de reglas junto con un objetivo. Puede ser lanzado desde el shell de SWI-Prolog.



Entorno: programas

- Comandos básicos




Comando	Explicación
?- halt.	Sale del entorno Prolog.
?- edit('archivo').	Invoca al editor predeterminado.
?- consult('archivo'). ['archivo'].	Consulta el fichero fuente (B. conocimiento).
?- help. help('ayuda').	Solicita ayuda al entorno.
?- make.	Consulta los ficheros que cambiaron desde la última consulta.
?- listing.	Muestra todos los predicados de la base de conocimiento.
?- listing('predicado').	Muestra el predicado especificado.
?- apropos('palabra').	Busca información sobre lo introducido.
?- trace.	Activamos el modo de traza.

Entorno: programas

- **Consultar:** cargar y compilar un programa en Prolog.
 - Desde menú → file/consult
 - Desde el shell: varias formas
 - `?- [p1].` `?- ['c:/trabajo/p1.pl'].` `?- consult(p1).` `?- consult('c:/tr/p1.pl').`

`?- consult(<nombre de fichero>).`

Añade todas las construcciones (hechos y reglas) del fichero a las que ya existen en memoria (correspondientes al fichero actual). Se añaden al final, tal cual.

 `reconsult(<nombre de fichero>).`

Añade todas las construcciones (hechos y reglas) del fichero a las que ya existen en memoria. Se añaden al final, pero se borran todas aquellas reglas antiguas cuya cabecera coincide con alguna de las nuevas. Ídem con los hechos.

NOTA: Éste es realmente la acción realizada con *File > Consult* en SWI-Prolog.

 `?- [<fichero_1> , ... , <fichero_n>].`

Equivale a realizar `reconsult` sobre los ficheros indicados.

Entorno: programas

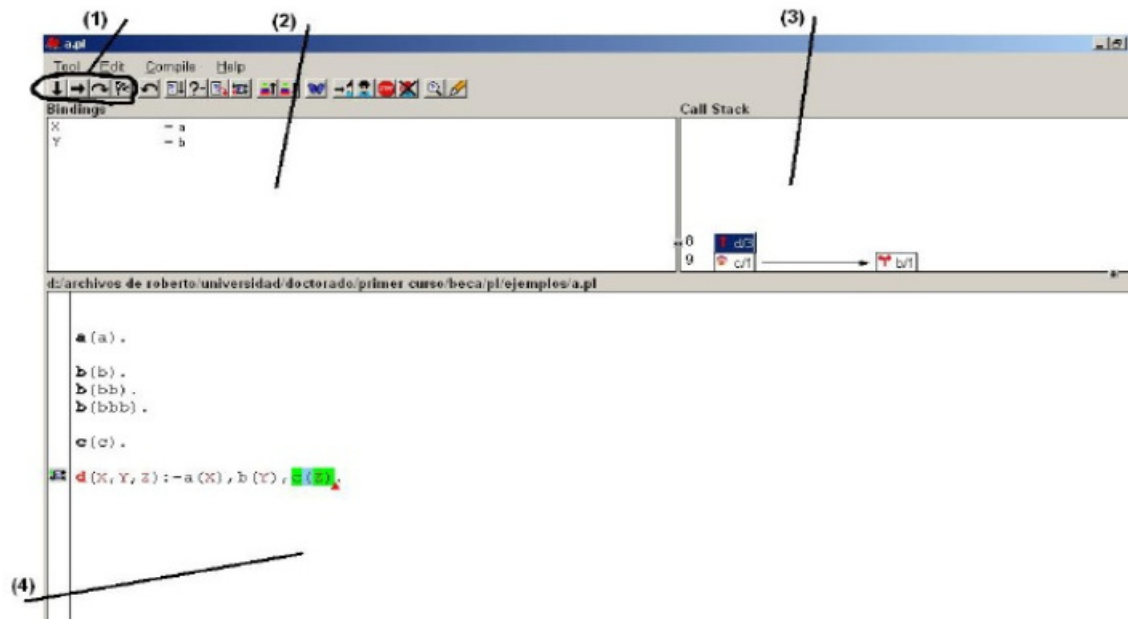
- **Errores y avisos** comunes al cargar y compilar un programa en Prolog:
 - Error sintáctico: el intérprete mostrará el error que se ha producido, donde ha sido y por qué se ha parado.
 - Error de archivo no encontrado: hay que fijarse en el nombre del archivo, la ruta que se ha puesto explícitamente y los directorios por defecto en donde busca, como puede ser el directorio de trabajo.
 - Aviso de variable *singleton*: variable que solamente aparece una vez en una cláusula. El compilador interpreta que puede haber un error al escribir esa variable pero, por no tratarse de un error, da un aviso.
 - Aviso de que las cláusulas de un mismo predicado no están juntas: este aviso salta cuando se escriben cláusulas de un predicado entre cláusulas de otro.

Entorno: programas

- **Ayuda:** desde el shell “help.”
- **Depuración:** para entrar en el modo trace hay que escribir “trace.”. Al lado del shell aparecerá “[trace]”. Para salir de este modo hay que escribir “notrace.” y “nodebug.”
 - Existe la posibilidad de utilizar un depurador gráfico escribiendo “guitracer.”

Entorno: programas

- **Depurador gráfico**
 - (1) avanzar en la ejecución del programa
 - (2) unificaciones que llevamos en este momento en esta cláusula
 - (3) pila de objetivos para la recursión
 - (4) que cláusula se esta ejecutando ahora y en color verde a qué objetivo se va a llamar. En color rojo aparecen las clausulas que fallan.



Entorno: programas

- **asserta(P)**: introduce P en la base de hechos, al comienzo de la definición de P.
- **assertz(P)**: introduce P en la base de hechos, al final de la definición de P.
- **retract(P)**: elimina P de la base de hechos.
- **listing**: muestra todos los hechos y reglas introducidos en la base de conocimiento.
- **listing(P)**: muestra todos los hechos y reglas sobre el predicado P.

Sintaxis

- **Comentarios**

- /* esto es un comentario de 1 a n líneas */
- % comentario de 1 línea

- **Operadores**

- ^ mod * / + - = \= =< >= == <>

- **Funciones**

- El nombre la primera letra debe ser minúscula. Deberá estar seguido de un conjunto de términos (otras funciones, variables o constantes) encerrados entre paréntesis. Ej: f(c,X), arbol(Hijo Izq, Raiz, Hijo Der).

- **funciones aritméticas predefinidas** (abs, sig, min, max, random, round, integer, float, sqrt, sin, cos, tan, log, log10, exp, ...).

- ?- X is abs(-7.8).
- X=7.8

Sintaxis

- **Términos:** pueden ser Variables y Constantes.
 - **Constantes**
 - **Primera letra min:** c, pi
 - Dos clases:
 - Átomos:
 - » Cadenas de letras, dígitos y subrayado (_) empezando por letra minúscula.
 - » Cualquier cadena de caracteres encerrada entre comillas simples (').
 - » Combinaciones especiales de signos: "?-", ":-", ...
 - Números:
 - » Enteros: en la implementación de Prolog-2 puede utilizarse cualquier entero que el intervalo $[-223, 223-1] = [-8.388.608, 8.388.607]$.
 - » Reales: decimales en coma flotante, consistentes en al menos un dígito, opcionalmente un punto decimal y más dígitos, opcionalmente E, un «+» o «-» y más dígitos.
 - **Variables**
 - Primera letra May: X, Valor, Suma.
 - **Cualquier identificador que empiece por mayúscula, será tomado por Prolog como una variable.**

Sintaxis

- **Conectivas lógicas**

- **+Goal1, +Goal2 → Conjunción.** Cuando se le da a Prolog una secuencia de objetivos separados por comas, intentará satisfacerlos por orden, buscando objetivos coincidentes en la Base de Conocimiento.
- **+Goal1; +Goal2 → Disyunción.** También la podemos representar mediante un conjunto de sentencias alternativas, es decir, poniendo cada miembro de la disyunción en una cláusula aparte.
- La negación lógica no puede ser representada explícitamente en Prolog, sino que se representa implícitamente por la falta de aserción. No es una verdadera negación, en el sentido de la Lógica, sino una negación “por fallo”.
- La implicación o condicional, sirve para significar que un hecho depende de un grupo de otros hechos. En castellano solemos utilizar las palabras “si ... entonces ...”. En Prolog se usa el símbolo “:-” para representar lo que llamamos una regla:

cabeza_de_la_regla :- cuerpo_de_la_regla

La cabeza describe el hecho que se intenta definir; el cuerpo describe los objetivos que deben satisfacerse para que la cabeza sea cierta.

Sintaxis

- **Hechos y predicados**

- Los **hechos** son Clausulas de Horn $P \leftarrow$ que se escriben $?- p$.
- Los hechos representan una relación entre objetos.
- En general, la sintaxis es: **relación (objeto, objeto, ...)**.
- La relación se conoce como el predicado y los objetos como los argumentos.
predicado(argumento, argumento, ...).
- Los nombres de todos los predicados y argumentos deben de comenzar con una letra minúscula. Al final del hecho debe ir un punto.
- Llamaremos **cláusulas** de un predicado tanto a los hechos como a las reglas. Una colección de cláusulas forma una Base de Conocimientos.

/ Conjunción de predicados */*

`le_gusta_a(clara,jorge), le_gusta_a(clara,chocolate).`

/ Disyunción de predicados */*

`le_gusta_a(clara,jorge); le_gusta_a(jorge,clara).`

/ Condicional: REGLAS */*

`novios(X,Y) :- le_gusta_a(X,Y),`

`le_gusta_a(Y,X).`

Sintaxis

- Hechos

Especificación de hechos

- Átomos sin cuantificación:

```
gusta(juan,maria).
```

- Átomos con cuantificación existencial (uso de variables de Skolem):

```
gusta(alguien_maria,maria).  /*  $\exists x$  (gusta(x,maria)) */  
gusta(alguien_paula,paula).  /*  $\exists x$  (gusta(x,paula)) */
```

- Átomos con cuantificación universal,
dependiendo del compilador de PROLOG:

```
gusta(X,maria).  
gusta(_,maria).           /*  $\forall x$  (gusta(x,maria)) */
```

Sintaxis

- Hechos

```
arbolgenealogico.pl
File Edit Browse Compile Prolog Pce Help
arbolgenealogico.pl
progenitor(joseLuis,joseLuisJ).
progenitor(joseLuis,joaquin).
progenitor(isabel,joseLuisJ).
progenitor(isabel,joaquin).
progenitor(joseLuisJ,marta).
progenitor(joseLuisJ,daniel).
progenitor(lucia,marta).
progenitor(lucia,daniel).
progenitor(joaquin,sonia).
progenitor(teresa,sonia).
Buffer saved in file `arbolgenealogico.pl' Line:
```



Sintaxis

- Hechos

```
1 ?- [arbolgenealogico].  
true.  
2 ?- progenitor(joseLuis,joaquin).  
false.  
3 ?- progenitor(joseLuis,joaquin).  
true.  
4 ?- progenitor(joseLuis,lucia).  
false.  
5 ?- progenitor(joseLuis,daniel).  
false.  
6 ?- progenitor(X,daniel).  
X = joseLuisJ .  
7 ?- progenitor(isabel,Y).  
Y = joseLuisJ .  
8 ?- progenitor(isabel,Y).  
Y = joseLuisJ ;  
Y = joaquin.  
9 ?-
```



Sintaxis

- Hechos

```
12 ?- progenitor(A,B).  
A = joseLuis,  
B = joseLuisJ ;  
A = joseLuis,  
B = joaquin ;  
A = isabel,  
B = joseLuisJ ;  
A = isabel,  
B = joaquin ;  
A = joseLuisJ,  
B = marta ;  
A = joseLuisJ,  
B = daniel ;  
A = lucia,  
B = marta ;  
A = lucia,  
B = daniel ;  
A = joaquin,  
B = sonia ;  
A = teresa,  
B = sonia.
```



Sintaxis

- Hechos

¿Quién es abuelo/a de Sonia?

```
14 ?- progenitor(X,sonia),progenitor(Abuelo,X) .  
X = joaquin,  
Abuelo = joseLuis ;  
X = joaquin,  
Abuelo = isabel .
```



Sintaxis

- Hechos

¿Quién es nieto/a de Isabel?

```
16 ?- progenitor(isabel,H),progenitor(H,Nieto).  
H = joseLuisJ,  
Nieto = marta ;  
H = joseLuisJ,  
Nieto = daniel ;  
H = joaquin,  
Nieto = sonia.
```



Sintaxis

- Reglas

- El nombre la primera letra min
- Deberá estar seguido de un conjunto de términos (otras funciones, variables o constantes) encerrados entre paréntesis. Ej: $f(c,X)$, $\text{arbol}(\text{Hijo Izq}, \text{Raiz}, \text{Hijo Der})$

Para todos X e Y,

X es descendiente de Y SI

Y es progenitor de X.

$\text{descendiente}(X,Y) \text{ :- progenitor}(Y,X).$



Sintaxis

- Reglas: simples

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

La regla equivale a: “*Belén cuida a Pedro* si *belén* está en *paro* y *Pedro* es bueno”. En lógica de predicados sería:

$paro(belen) \wedge bueno(pedro) \rightarrow cuida(belen, pedro)$

El símbolo $:-$ podría interpretarse como: \leftarrow (si)

Sintaxis

- 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).
```

Equivale a:

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

✓ En lógica de predicados:

$\forall x \forall y (mujer(x) \wedge progenitor(x,y) \rightarrow madre(x,y))$

Sinta

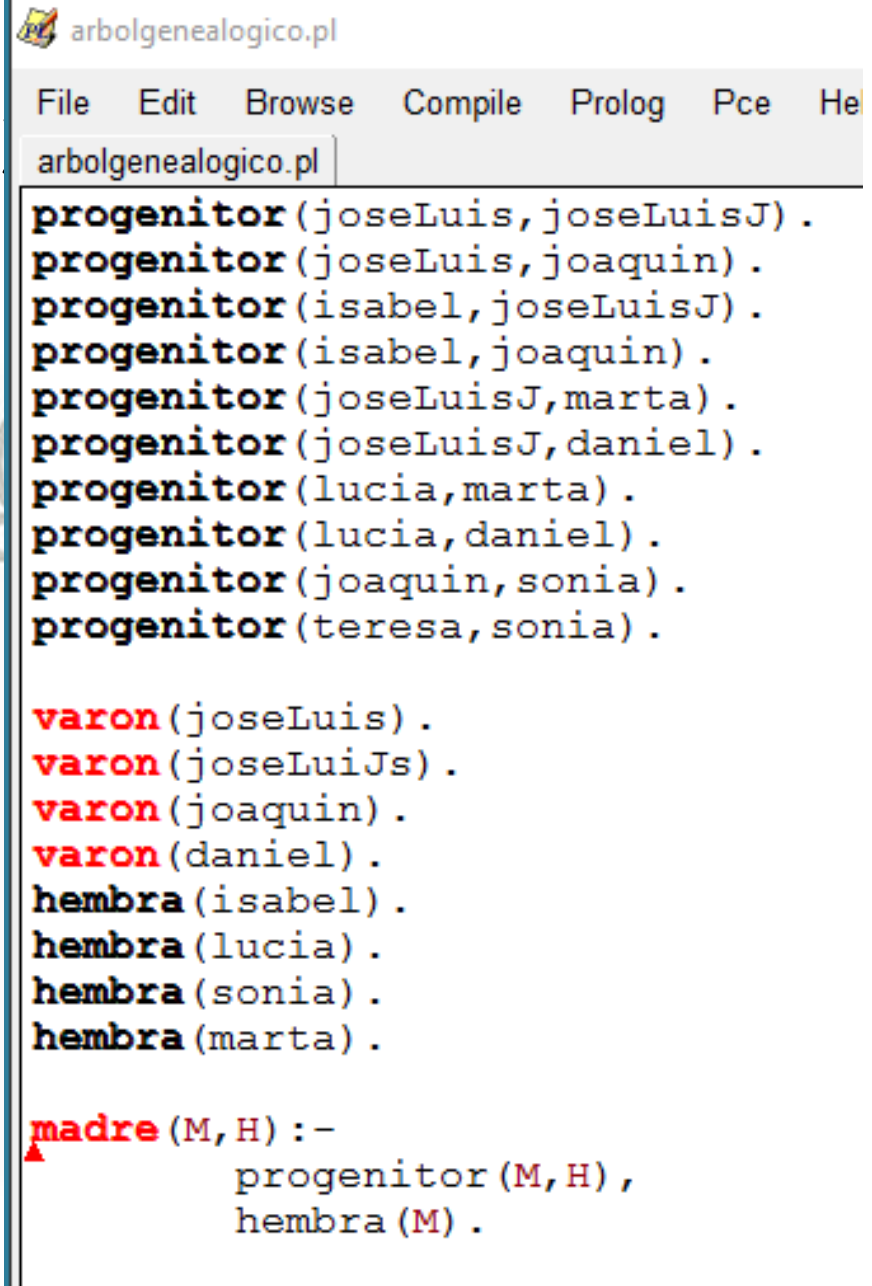
- Reglas

madre (M, H) :-

progenitor(M, H),

hembra(M).

```
20 ?- madre(isabel,joaquin).  
true.
```



```
arbolgenealogico.pl  
File Edit Browse Compile Prolog Pce He  
arbolgenealogico.pl  
progenitor(joseLuis,joseLuisJ).  
progenitor(joseLuis,joaquin).  
progenitor(isabel,joseLuisJ).  
progenitor(isabel,joaquin).  
progenitor(joseLuisJ,marta).  
progenitor(joseLuisJ,daniel).  
progenitor(lucia,marta).  
progenitor(lucia,daniel).  
progenitor(joaquin,sonia).  
progenitor(teresa,sonia).  
  
varon(joseLuis).  
varon(joseLuisJs).  
varon(joaquin).  
varon(daniel).  
hembra(isabel).  
hembra(lucia).  
hembra(sonia).  
hembra(marta).  
  
madre(M,H):-  
    progenitor(M,H),  
    hembra(M).
```

Sintaxis

- 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)

Sintaxis

- Reglas recursivas

**hermana(X, W) :-
 progenitor(Z, X),
 progenitor(Z, W),
 hembra(X).**

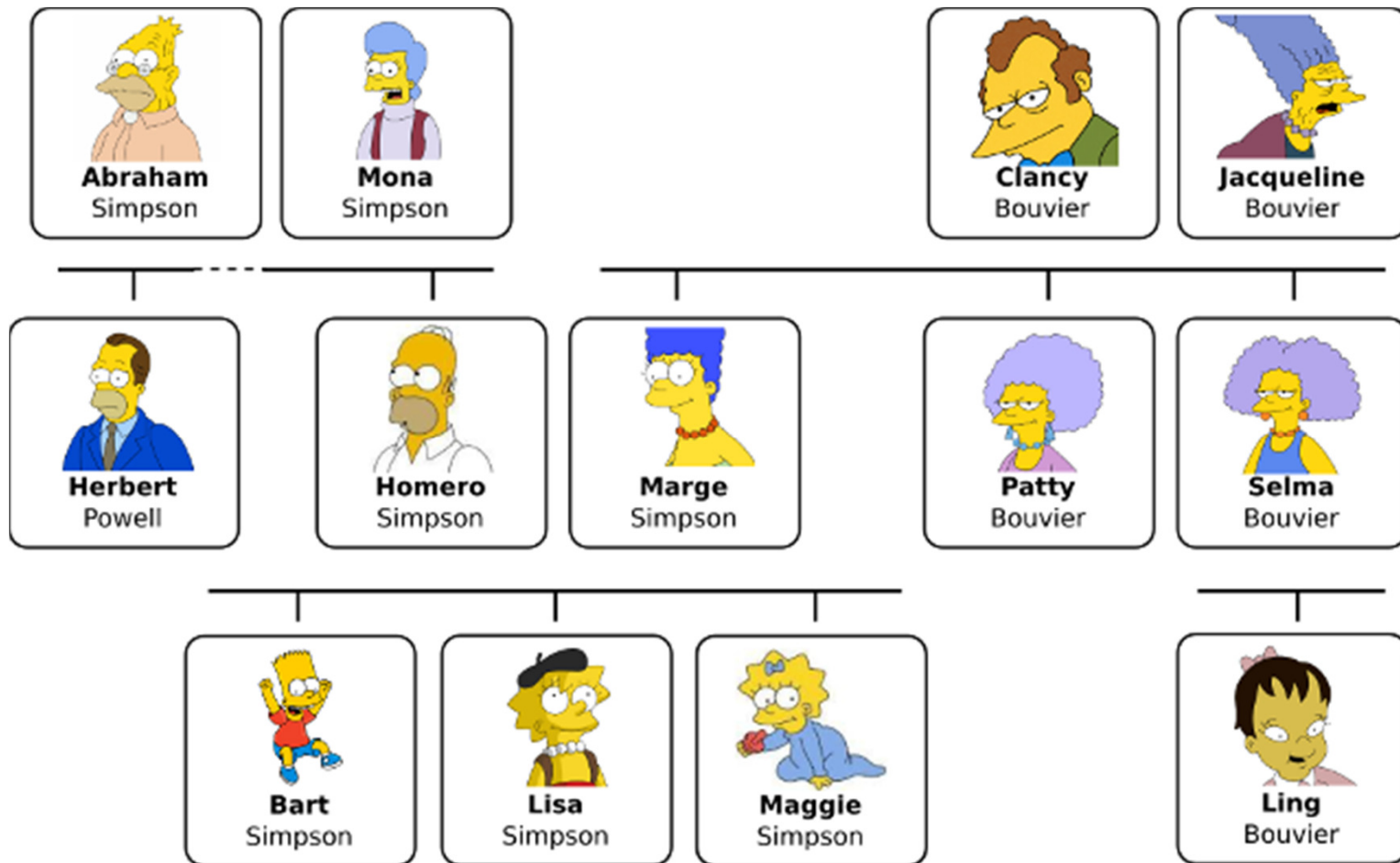
**antepasado(X, Z) :-
 progenitor(X, Z).**

**antepasado (X, Z) :-
 progenitor (X, W),
 antepasado (W, Z). (OJO CON MANEJAR BIEN LA RECURSIVIDAD)**



Sintaxis

Ejercicio: *realizar un programa Prolog lo más completo posible para el árbol siguiente.*



Sintaxis

Ejercicio: *realizar un programa Prolog que simule en sistema experto en detección de enfermedades a partir del código que ya se dispone en el fichero **doctor.pl.**, de tal manera que podamos preguntarle a partir de uno o varios síntomas de un paciente, y responda con la enfermedad que tiene, el especialista al que debe ir, así como alimentos que le pueden sentar bien y mal.*

Fuentes y bibliografía

- www.swi-prolog.org/
- ftp://www.cc.uah.es/pub/Alumnos/G_Ing_Informatica/Conocimiento_y_Razonamiento_Automatizado/
- Ejercicios de programación declarativa con Prolog. José A. Alonso Jiménez. Grupo de Lógica Computacional. Dpto. de Ciencias de la Computación e Inteligencia Artificial. Universidad de Sevilla.
- Prácticas de Lógica. Prolog. Faraón Llorens Largo, M^a Jesús Castel de Haro. Universidad de Alicante.
- Programación Práctica en Prolog. Jose E. Labra G.. Área de Lenguajes y Sistemas Informáticos. Departamento de Informática. Universidad de Oviedo.