

# SWI Prolog



SwiProlog es una implementación portable del lenguaje de programación Prolog. Guía su desarrollo la compatibilidad, portabilidad, escalabilidad y estabilidad; proporcionando una implementación robusta que soporta una amplia gama de aplicaciones.

SwiProlog contiene una base de librerías con interfaces a otros lenguajes, base de datos, gráficos y redes. Provee un extenso soporte para manejar documentos HTML/SGML/XML y RDF. El sistema es particularmente adecuado para aplicaciones server debido a un fuerte apoyo de las librerías para multihilo y servidor HTTP.

Además de atenerse al estándar que fija la norma ISO-Prolog, es en gran medida compatible con Quinto, SICStus y YAP Prolog. SwiProlog ofrece un marco de compatibilidad desarrollado en cooperación con YAP y crea instancias para YAP, SICStus y IF/Prolog.

SwiProlog tiene como objetivo proporcionar un buen entorno de desarrollo, incluyendo un editor con un amplio soporte, depurador gráfico a nivel de fuente, cargador automático y mucho más.

Este documento ofrece una visión resumida de las características del sistema.

## 1) Introducción

Este documento tiene por objetivo introducir al lector en la instalación y uso de SwiProlog, poniéndolo en relación con los conceptos del paradigma lógico que abarca la asignatura Paradigmas de Lenguajes.

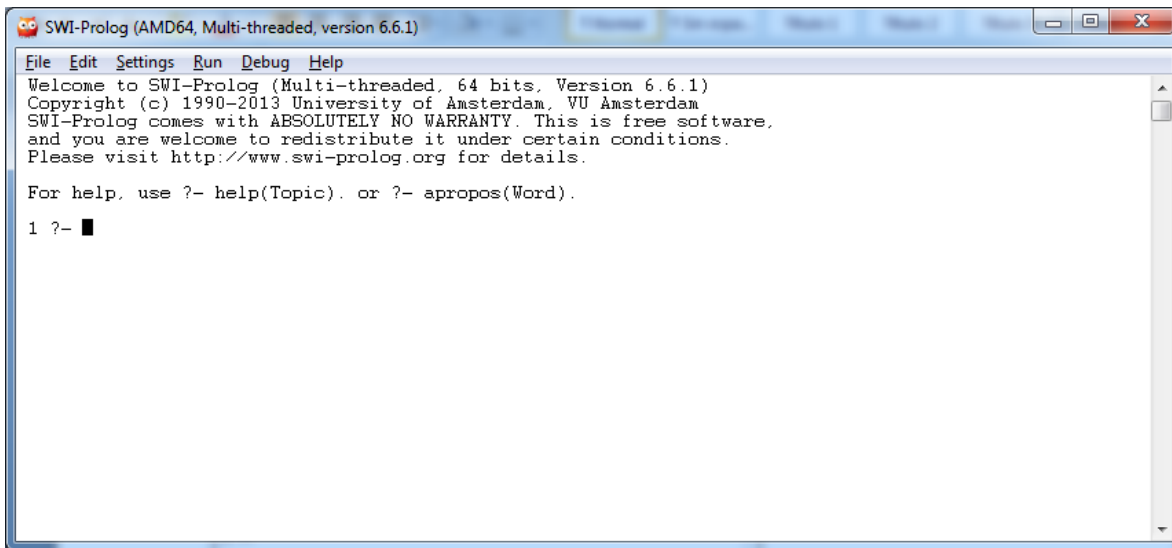
## 2) Comandos básicos

Descargar los instaladores desde el sitio oficial:

<http://www.swi-prolog.org/download/stable> y seguir los pasos de instalación del sistema.

Una vez instalado SwiProlog, desde la ventana de comandos ejecutar:

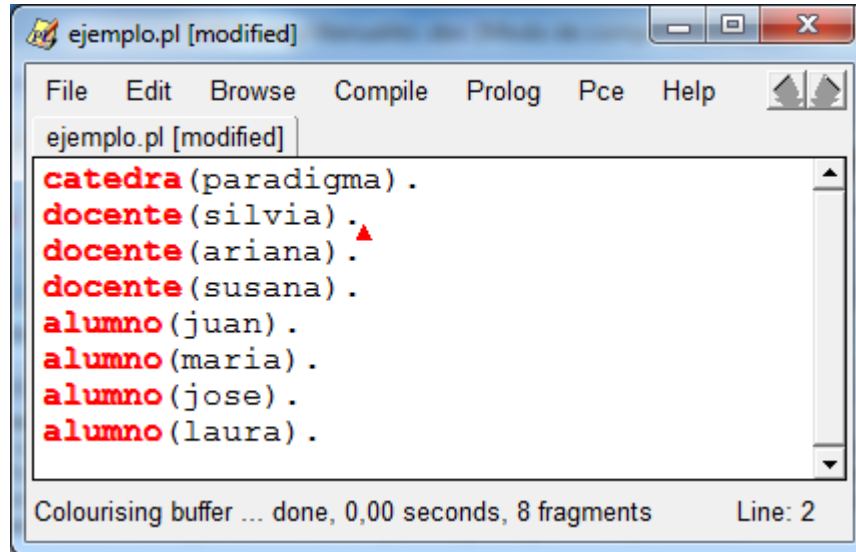
**> swipl**



Desde este entorno es posible cargar un programa para realizar las consultas necesarias o bien, editar o crear un nuevo programa. A continuación, se detallan estas operaciones:

1. Crear un programa: Cualquier programa en SwiProlog tiene que estar escrito en un fichero de texto plano (sin formato), y tener la extensión **.pl**. Puede usarse cualquier editor de texto, incluso el Block de Notas, sin embargo, aquí seguiremos los pasos del editor que propone este entorno:
  - a. **?- emacs.**

- b. Seleccionar del menú File -> New y luego pedirá el nombre del programa. Ahora se puede tipear el programa. Por ejemplo:

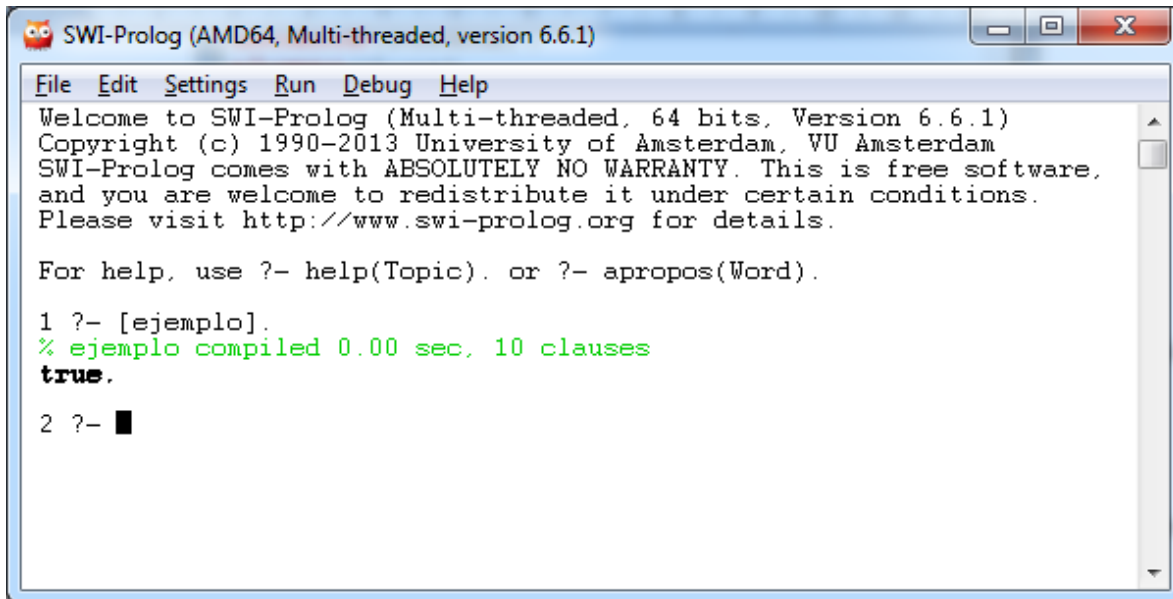


The screenshot shows a window titled 'ejemplo.pl [modified]' with a menu bar (File, Edit, Browse, Compile, Prolog, Pce, Help) and a toolbar. The text area contains the following Prolog code:

```
catedra(paradigma).  
docente(silvia).  
docente(ariana).  
docente(susana).  
alumno(juan).  
alumno(maria).  
alumno(jose).  
alumno(laura).
```

At the bottom of the window, a status bar displays 'Colourising buffer ... done, 0,00 seconds, 8 fragments' and 'Line: 2'.

- c. Luego compilar para depurar posibles errores. En esta instancia pedirá grabar el archivo en la carpeta por defecto.
- d. Para efectuar las consultas es necesario cargar el archivo.
2. Realizar consultas: Para cargar y ejecutar (o hacer consultas) un programa se puede seguir dos caminos:
- a. Dar doble clic al archivo **ejemplo.pl** en el administrador de archivos, abre el SWI y carga el programa, o bien.
  - b. En Windows: Abrir el entorno SWI, y luego tipear el nombre del archivo entre corchetes seguido de punto.

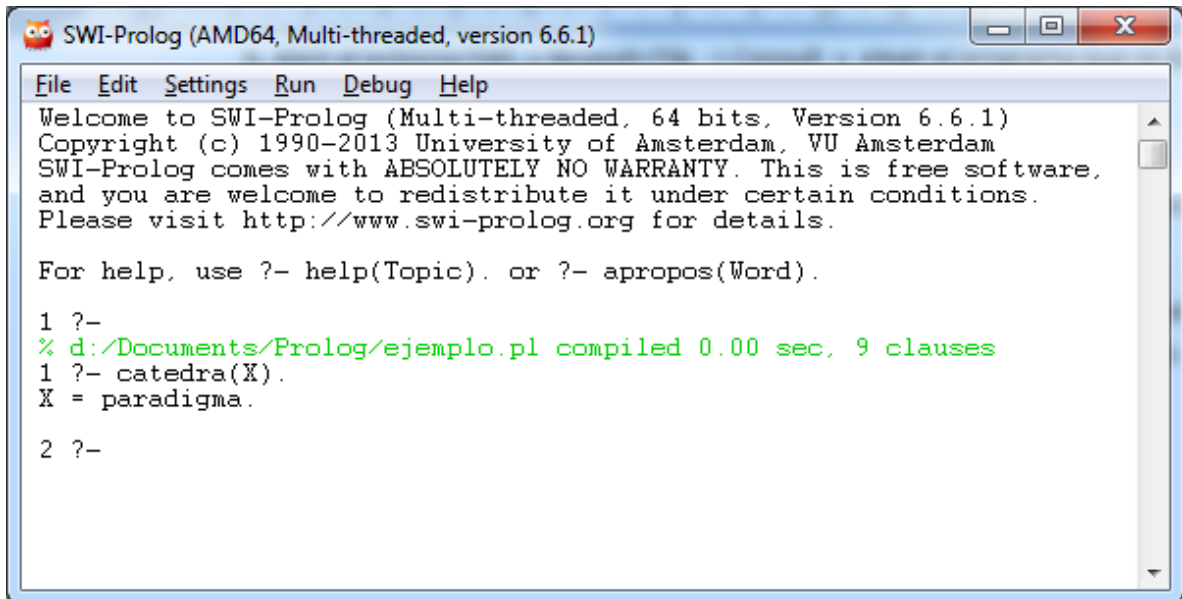


c. Si no recuerdo el nombre: abrir el entorno SWI y después File -> Consult y elegir el programa que se desea cargar.

Si hay errores en el programa, aparecen en la ventana de SWI indicando la línea de cada error. Marca también los errores warnings, que, aunque permitan hacer la consulta deben ser resueltos.

**Nota:** Si se modifica y graba un programa en la ventana del editor y SWI no se actualiza, entonces se deberá tipear: ?- **make.** en la ventana del SWI; esto hace que el sistema levante de nuevo el programa a partir del archivo .pl.

Una vez que se cargó el programa, las consultas se hacen escribiéndolas en la ventana del SWI, al lado del prompt ?- . Tener en cuenta que las consultas deben terminar con punto. Por ejemplo:



```
SWI-Prolog (AMD64, Multi-threaded, version 6.6.1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.1)
Copyright (c) 1990-2013 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.

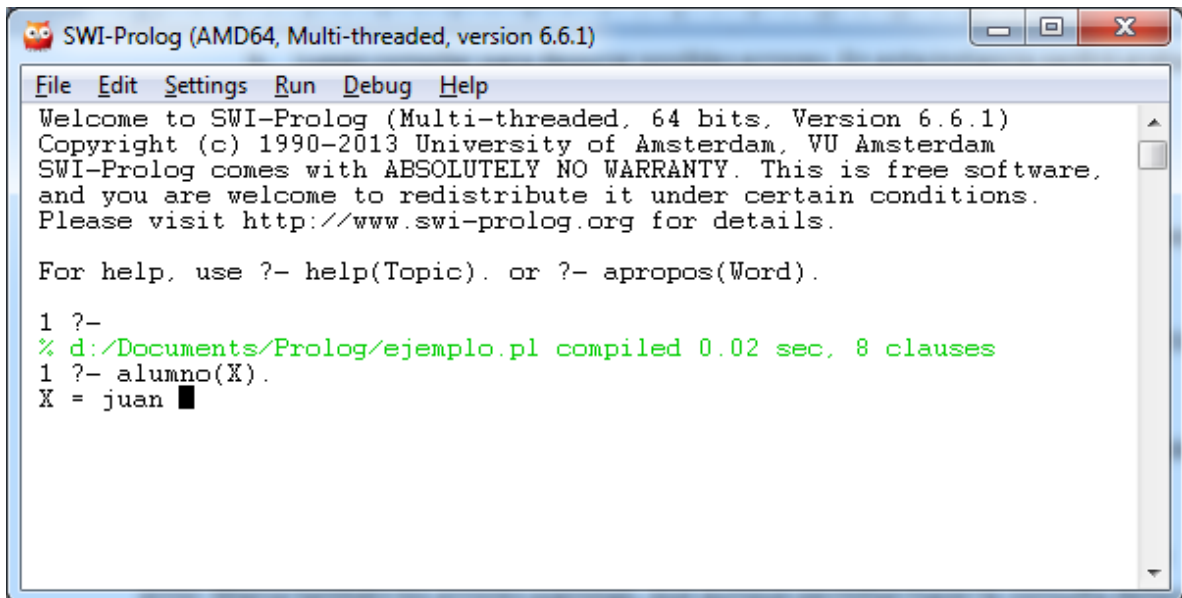
For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% d:/Documents/Prolog/ejemplo.pl compiled 0.00 sec, 9 clauses
1 ?- catedra(X).
X = paradigma.

2 ?-
```

Esta consulta tiene una sola solución que la muestra seguida por un punto.

Veamos otra consulta.



```
SWI-Prolog (AMD64, Multi-threaded, version 6.6.1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.1)
Copyright (c) 1990-2013 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 ?-
% d:/Documents/Prolog/ejemplo.pl compiled 0.02 sec, 8 clauses
1 ?- alumno(X).
X = juan ■
```

Como puede verse, muestra **la primera solución**.

Para **ver más** es necesario **ingresar** ; (punto y coma), de lo contrario **pulso Enter**.

La última solución la muestra seguida de punto.

Si **ingreso** ; y responde **False** es porque en la última rama de búsqueda no encontró otra solución.

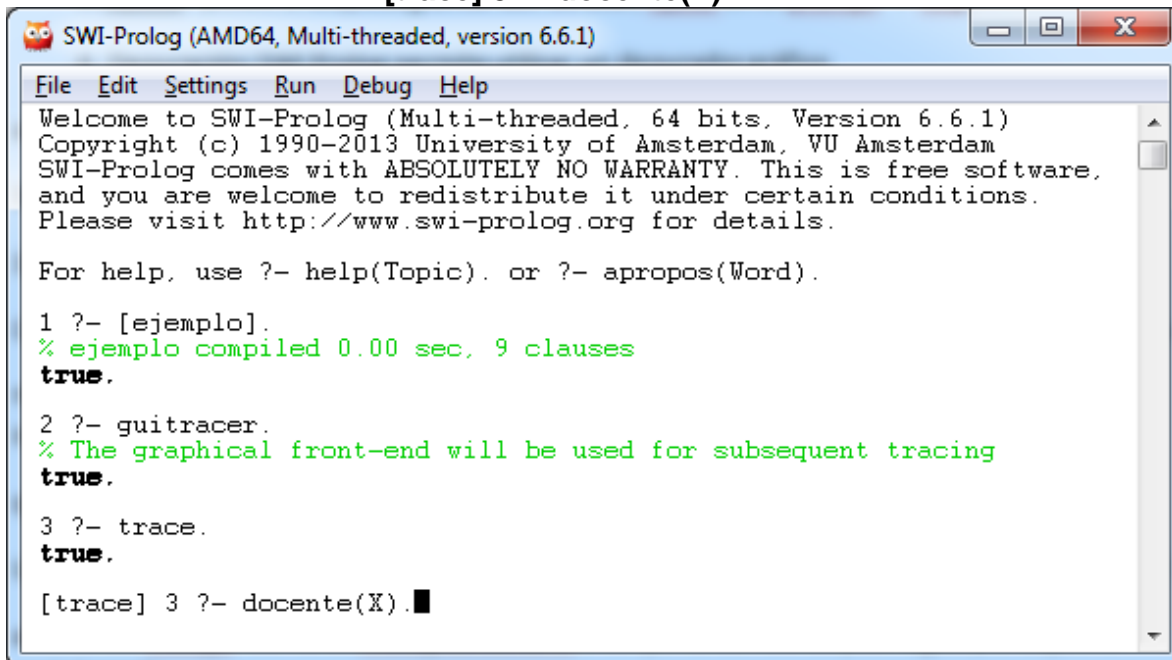
3. Ayuda: posee una ayuda grafica fácil de usar, tipear en el prompt **?- help**.

4. Depuración: SWI Prolog permite utilizar un depurador gráfico tipeando:

**?- guitracer.**

**?- trace.** y luego la consulta que deseamos.

**[trace] 3 ?- docente(X).**



```
SWI-Prolog (AMD64, Multi-threaded, version 6.6.1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.1)
Copyright (c) 1990-2013 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).

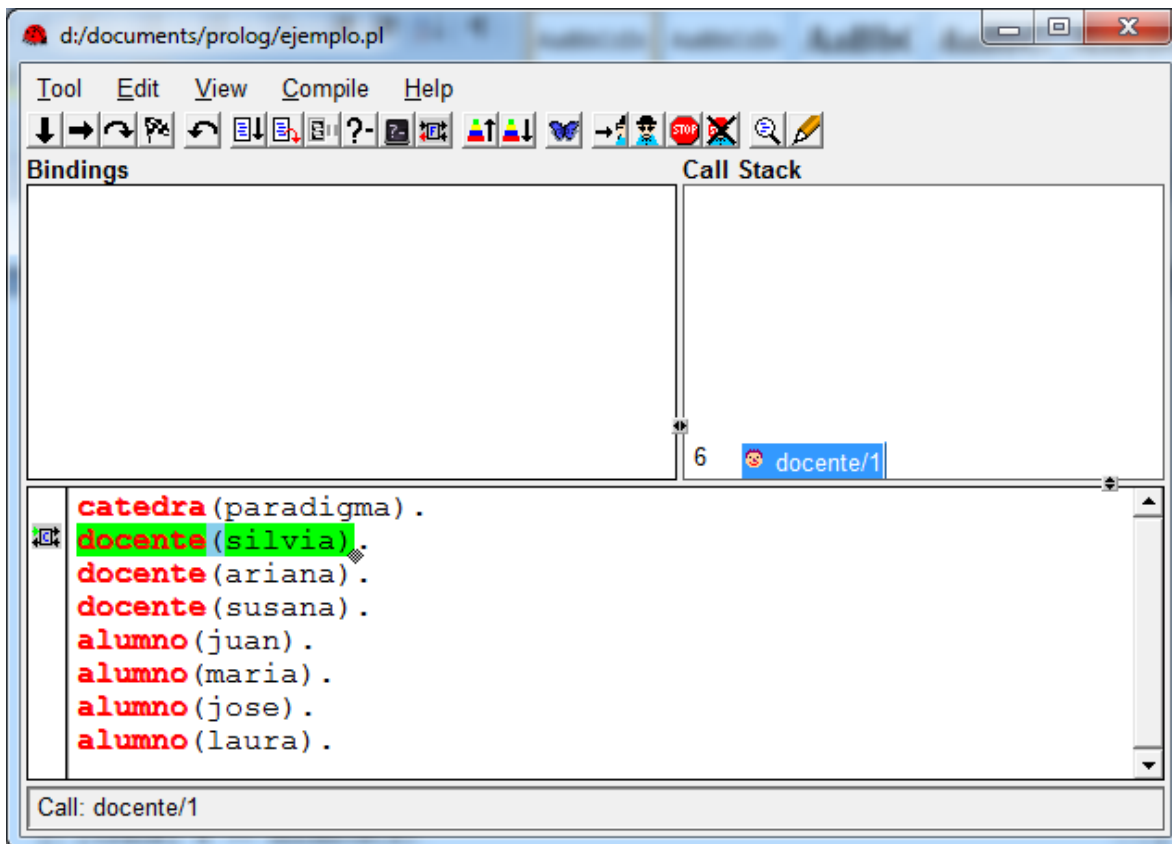
1 ?- [ejemplo].
% ejemplo compiled 0.00 sec, 9 clauses
true.

2 ?- guitracer.
% The graphical front-end will be used for subsequent tracing
true.

3 ?- trace.
true.

[trace] 3 ?- docente(X).
```

Se abrirá una ventana que irá mostrando paso a paso la secuencia de búsqueda y la pila de llamadas. Tener en cuenta que debe interactuar con la ventana del SWI para permitirle continuar con el árbol de búsqueda.



Para cerrar este modo se debe escribir: `?- notrace.` o bien `?- nodebug.`

### 3) Errores y avisos

Cuando se hacen las consultas pueden aparecer mensajes de error o de aviso. Los más comunes son:

- Error sintáctico: el intérprete mostrará el error que se ha producido, donde ha sido y por qué se ha detenido.
- Error de archivo no encontrado: Verificar el nombre del archivo, la ruta de acceso por defecto en donde busca, como puede ser la carpeta de trabajo.
- Aviso de variable singleton: una variable singleton es una variable que solamente aparece una vez en una cláusula. El compilador interpreta que puede haber un error al escribir esa variable, pero no se trata de un error, solo da un aviso. Si la cláusula es correcta y posee una variable singleton significa que el valor de esa variable no importa en dicha cláusula ya que va a unificar con cualquier cosa y no tiene relevancia posterior. Para que el

programa sea más claro y no salga el aviso podemos usar, en su lugar, una **variable anónima**.

- d) 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.

#### 4) Base de conocimiento

Para ver el contenido de la base de conocimiento usamos el predicado **listing**. Este predicado muestra todas las cláusulas que tenemos actualmente. Para ver solo aquellas cláusulas que pertenecen a un predicado usamos **listing(nombrePred)**, donde nombrePred es el predicado que buscamos mirar. También podemos escribir **listing(nombrePred/Arid)**, donde Arid es la aridad del predicado nombrePred.

#### 5) Modificar dinámicamente la base de conocimiento.

Es posible añadir y eliminar cláusulas, usando la familia de predicados **assert** y **retract**. Con **assert(nombrePred)** insertamos la cláusula al principio de la lista de cláusulas de ese predicado. Con **assertz(nombrePred)** hacemos lo mismo, pero al final.

Con **retract(nombrePred)** elimina el predicado con el que primero unifica.