Tutorial SWI-Prolog

Toda la documentación de SWI-Prolog se encuentra disponible en http://www.swi-prolog.org, y se puede acceder a una versión limitada online del intérprete en http://swish.swi-prolog.org.

Para **iniciar** una sesión con el intérprete de Prolog en un sistema Unix, basta teclear 'swipl' (o 'pl' en algunas distribuciones) después del prompt ('%'):

```
% swipl
```

En sistemas Windows se accederá al intérprete a través de su icono. Inmediatamente después de la invocación del intérprete, aparece un mensaje como el siguiente y un **prompt** ('?-') donde introducir los **objetivos** Prolog:

```
Welcome to SWI-Prolog (Multi-threaded, 64 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).

?-
```

Podéis acceder a la **ayuda** general tecleando:

```
?- help(help).
```

Para **cargar un fichero** con un programa Prolog se pueden usar indistintamente:

```
?- consult('file.pl').
?- consult(file).
?- ['file.pl'].
?- [file].
```

Puesto que el fichero en los ejemplos anteriores es relativo, deberá encontrarse en el mismo directorio donde se ha invocado al intérprete. En un sistema Windows, se pueden cargar programas en otras rutas, como en el siguiente ejemplo:

```
?- ['C:/Program Files/pl/demo/likes'].
```

o cambiar el directorio de trabajo con:

```
?- working_directory(_, 'C:/Program Files/pl/demo').
```

Para **terminar** la sesión con el intérprete se pulsará Ctrl-D o se tecleará ‘halt.’:

```
?- halt.
```

Un **ejemplo** de interacción con el intérprete usando el predicado predefinido ‘member’ podría ser el siguiente:

```
?- member(1, [1,2,3]).   /* El usuario teclea este objetivo. */
true ;                   /* Prolog encuentra una solución a este objetivo y devuelve ‘true’. El usuario
                            teclea ‘;’ indicando que quiere saber si hay más soluciones. */
false.                  /*  Al no haber más soluciones el intérprete devuelve ‘false’.

?- member(4, [1,2,3]).   /* El usuario teclea un nuevo objetivo */
false.                   /* que resulta ser insatisfactible. */

?- member(X, [1,2,3]).  /* Este nuevo objetivo contiene una variable en el primer argumento. */
X = 1 ;                  /* Una solución que satisface el objetivo es ‘X = 1’, pero se pulsa ‘;’ */
X = 2 ;                 /* para obtener otra solución, y se vuelve a pulsar ‘;’ */
X = 3.                   /* para obtener la última solución posible que satisface el objetivo. */
```

Prolog permite **depurar** programas de varias maneras. La más sencilla consiste en **trazar** toda la ejecución de un programa. Para **activar las trazas**, se invoca al predicado ‘trace’:

```
?- trace.
true.

[trace] ?-
```

que cambia el prompt por ‘[trace ?-]’, entrando al modo de traza. Modo del que se sale con ‘notrace’, seguido de ‘nodebug’, o directamente ‘nodebug’:

```
[trace] ?- nodebug.
true.

?-
```

También es posible trazar la ejecución de **predicados específicos** mediante **puntos espías** con el predicado ‘spy’, que introduce al intérprete en modo depuración, como indica el cambio de prompt:

```
?- spy(nat_max).
```

```
% Spy point on nat_max/2
true.

[debug] ?- nat_max(X, 2).
 * Call: (7) nat_max(_G330, 2) ? creep
 * Exit: (7) nat_max(1, 2) ? creep
X = 1 ;
 * Redo: (7) nat_max(_G330, 2) ? creep
   Call: (8) _G409 is 2+ -1 ? creep
   Exit: (8) 1 is 2+ -1 ? creep
 * Call: (8) nat_max(_G409, 1) ? creep
   Call: (9) fail ? creep
   Fail: (9) fail ? creep
 * Fail: (8) nat_max(_G409, 1) ? creep
 * Fail: (7) nat_max(_G330, 2) ? creep
false.
```

Estos puntos espía pueden **deshabilitarse** con ‘nospy’ y se puede salir del modo depuración con ‘nodebug’:

```
[debug] ?- nospy(nat_max).

[]
```