Guía de Uso Básico de Prolog

1. Prolog es un lenguaje conversacional

Al contrario que la mayoría de los lenguajes de programación, Prolog es un lenguaje conversacional; es decir, el sistema Prolog mantiene un diálogo continuo con el programador desde el inicio de la sesión hasta el final de la misma. Este diálogo toma generalmente la forma de un interrogatorio, a lo largo del cual el programador planteará preguntas al sistema Prolog. Por su parte, el sistema Prolog responderá cada una de las preguntas formuladas por el programador en la medida en que esto sea posible.

Prolog le indica al programador que está esperando a que éste le formule una pregunta mostrando en pantalla el siguiente símbolo

```
?-
```

Tras este símbolo, el programador puede teclear una pregunta (terminada en un punto) y pulsar el retorno de carro. Con ello, el programador solicita al sistema Prolog que responda a la pregunta recién formulada. Una vez procesada la pregunta el sistema Prolog mostrará en pantalla la respuesta correspondiente.

Por ejemplo, si queremos preguntar a Prolog si 5 es igual a 2+3 podemos teclear la pregunta

```
?- 5 is 2+3.
Yes
```

Después de pulsar el retorno de carro, Prolog comprobará que efectivamente 2 y 3 suman 5 y, por lo tanto, responderá afirmativamente (`Yes`). Prolog puede dar también respuestas negativas a las preguntas

```
?- 1 is 1+1.
No
```

Es importante recordar que **todas las preguntas formuladas a Prolog deben terminar en un punto**. Si se olvida incluir el punto, por más veces que se presione retorno de carro, Prolog considerará que la pregunta no está formulada en su totalidad y, por lo tanto, seguirá esperando a que se termine de formular la pregunta. Por ejemplo, si olvidamos teclear el punto en la pregunta

```
?- 5 is 2+3
|
```

Prolog mostrará el símbolo |, indicando que está esperando a que se termine de formular la pregunta, para lo que basta teclear un punto seguido de un retorno de carro

```
?- 5 is 2+3
|.
Yes
```

También es posible que cometamos algún error al teclear una pregunta. Como veremos más

adelante, **las preguntas son realmente términos Prolog y deben ajustarse a una sintaxis formal concreta**. Si la pregunta en cuestión no es un término Prolog correcto, habremos cometido un error sintáctico. Afortunadamente, Prolog es capaz de detectar tales errores y nos avisará que no entiende la pregunta formulada. Por ejemplo, si al formular la pregunta anterior olvidamos teclear el operador de suma (+)

```
?- 5 is 2 3.
ERROR: Syntax error: Operator expected
ERROR: 5 is 2
ERROR: ** here **
ERROR: 3 .
```

el sistema Prolog nos advierte de que hay un error sintáctico (`syntax error`), mostrando la pregunta recién formulada y el punto en que se encuentra el error (`** here **`).

2. Base de conocimiento de Prolog

Para responder a las preguntas formuladas por el programador, Prolog consulta una base de conocimiento. Al iniciar una sesión Prolog, esta base de conocimiento almacena un conocimiento básico que incluye, entre otras cosas, conceptos y definiciones de la aritmética de los números naturales. Este conocimiento permite a Prolog responder correctamente las siguientes preguntas:

```
?- 5 is 2+3.
Yes
?- 1 is 1+1.
No
```

Obviamente, Prolog no es capaz de responder cualquier pregunta que le formulemos. Por ejemplo, si le preguntamos a Prolog si el pato Lucas es un pato

```
?- esPato(lucas).
ERROR Undefined predicate `esPato/1'
```

Prolog nos responderá que no sabe determinar si algo es o no un pato, pues su base de conocimiento no incluye información acerca de los patos. Formalmente, lo que ocurre es que el predicado lógico '`esPato/1`' no está definido (`undefined predicate`).

A lo largo de la conversación mantenida con el programador, Prolog no sólo es capaz de responder a ciertas preguntas que se le formulen, sino que es también capaz de aprender sobre aquello que no sabe. Durante la sesión, es posible ampliar esta base de conocimiento añadiendo definiciones de conceptos sobre los que Prolog no sabe nada (como por ejemplo una definición de pato), o bien modificando y extendiendo definiciones sobre conceptos que Prolog ya conoce (por ejemplo, añadiendo nuevos operadores aritméticos sobre los naturales).

La base de conocimiento se expresa mediante hechos y reglas Prolog, que no son otra cosa que una representación sintáctica concreta de cláusulas de Horn de primer orden. Por lo tanto, todo el conocimiento Prolog queda expresado empleando (casi) exclusivamente lógica de primer orden. Por ello se dice que Prolog es un lenguaje lógico y que la programación en Prolog es programación lógica. **Un programa Prolog no es por tanto más que un conjunto de hechos y reglas que expresan cierto conocimiento mediante lógica de primer orden.**

3. Predicados sobre directorios y ficheros

Los programas Prolog se almacenarán en ficheros de texto (con extensión '.pl'). Prolog adquiere nuevos conocimientos consultando (es decir, leyendo) estos programas. Para facilitar al programador el acceso a los programas almacenados en los ficheros, Prolog define un conjunto de predicados especiales que permiten navegar por el sistema de ficheros y visualizar los directorios. Un detalle importante a tener en cuenta es que **Prolog utiliza notación diferente a MS-DOS para representar las rutas de los ficheros**. Mientras que en MS-DOS los directorios que forman parte de una ruta se separan entre sí por el carácter '\', en Prolog se emplea el carácter '/' con el mismo propósito.

Así, el directorio MS-DOS `c:\juegos\comecoco` se escribe en notación Prolog `c:/juegos/comecoco`

El predicado pwd/0.

El predicado `pwd` imprime el directorio de trabajo actual (es equivalente al comando MS-DOS `cd` sin parámetros). Por ejemplo, si nuestro directorio de trabajo es `c:\prolog\marisol` (en notación MS-DOS), al preguntar a Prolog por el directorio actual obtenemos

```
?- pwd.
c:/prolog/marisol
```

es decir, el directorio actual en notación Prolog.

El predicado ls/0.

El predicado `ls` lista el contenido del directorio de trabajo actual (es equivalente al comando MS-DOS `dir`). Por ejemplo, si el directorio actual contiene los ficheros 'patos.pl' y 'familia.pl' al ejecutar `ls` obtenemos

```
?- ls.
patos.pl familia.pl
```

El predicado cd/1.

Finalmente, es posible cambiar el directorio actual mediante el predicado `cd` (equivalente al comando MS-DOS `cd`). El nombre del nuevo directorio debe ser una ruta (absoluta o relativa) en notación Prolog, encerrada entre comillas simples. Por ejemplo

```
?- cd('../pablo').
```

establecerá `c:\prolog\pablo` como nuevo directorio de trabajo, mientras que

```
?- cd('c:/prolog/marisol').
```

Reestablece `c:/prolog/marisol` como directorio de trabajo.

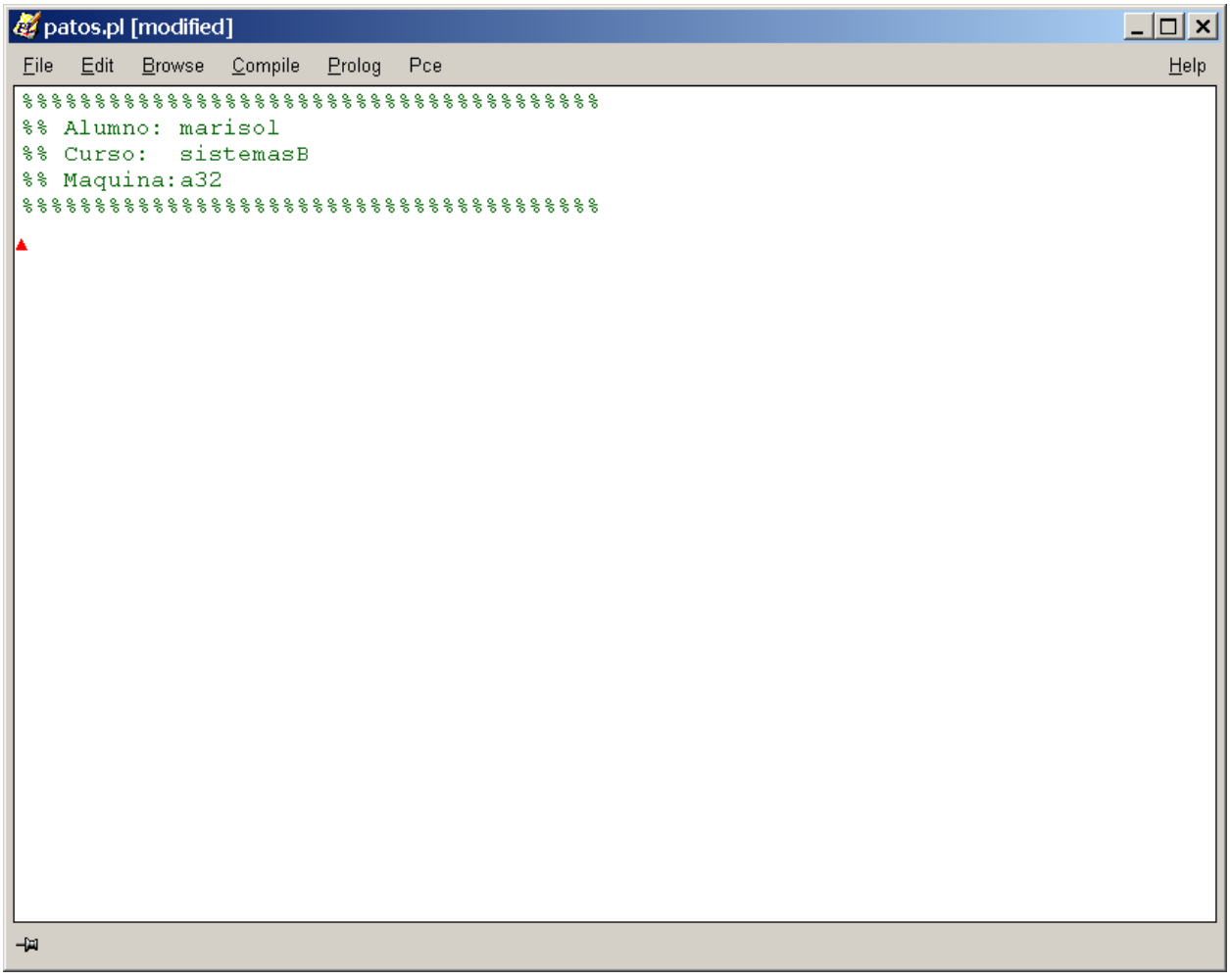
4. Creación, ejecución, modificación y consulta de programas Prolog

El predicado crea/4.

El primer paso para escribir un programa Prolog consiste en crear el fichero que lo almacenará. Esto se hace mediante el predicado `crea`, que recibe cuatro parámetros: el nombre del fichero a crear, el nombre del autor, el curso y el código de la máquina en que se está trabajando. Por ejemplo,

```
?- crea(patos,marisol,sistemasB,a32).
```

creará el fichero `'patos.pl'` en el directorio actual, le añadirá como comentario una cabecera en que figurarán el nombre del autor, el curso y el código de la máquina y, finalmente, editará el fichero para que podamos escribir nuestro programa Prolog. Es importante recordar que **todos los parámetros facilitados al predicado `crea` deben comenzar por una letra minúscula y no deben contener espacios**. Si el fichero mencionado en el predicado `crea` ya existiese, entonces simplemente se editará el fichero sin añadir cabecera ni realizar modificación alguna sobre el mismo. Tras ejecutar el predicado anterior, y suponiendo que el fichero `'patos.pl'` no existía en el directorio actual, aparecerá el editor con el siguiente texto



debajo del cual escribiremos ahora un programa Prolog en el que reflejemos nuestro conocimiento sobre los patos. Para ello, todo lo que debemos hacer es definir un predicado `'esPato(X)'`, que nos responda si un `X` dado es realmente un pato. Por ejemplo, sabemos que es un hecho que Lucas, Donald y el tio Gilito son patos. Podemos expresar este conocimiento en Prolog mediante 3 **hechos** (*verdades incondicionales*) tal y como sigue

```
esPato(lucas).
esPato(donald).
esPato(gilito).
```

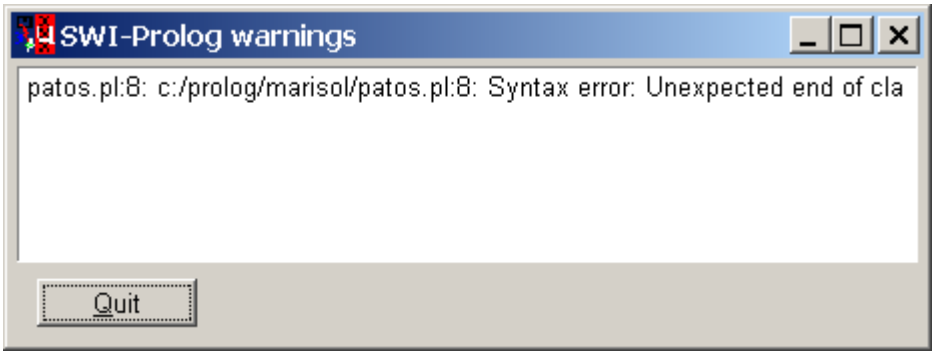
La primera línea del programa anterior puede entonces leerse como "es un hecho que Lucas es un pato". Obsérvese que **los nombres propios se escriben comenzando con minúsculas para distinguirlos de las variables que comienzan siempre con mayúsculas**. Una vez escritos estos 3 hechos, debemos guardar el fichero `'patos.pl'` e incluirlo en el sistema Prolog. Para guardar el fichero seleccione la opción de menú `File->Save Buffer`. Para incluir nuestras declaraciones debemos seleccionar la opción de menú `Prolog->Compile Buffer` que consultará (es decir, leerá) el fichero `'patos.pl'`, adquiriendo el conocimiento expresado en el mismo; es decir, que Lucas, Donald y Gilito son patos.

La opción de menú `Prolog->Compile Buffer` antes de incluir nuestras declaraciones en Prolog se cerciora de que el fichero guardado en disco y el editado coincidan de manera que si el editado tiene algunas modificaciones, nos avisa por si queremos guardarlo.

Otra cosa importante que hace esta opción es analizar el contenido del fichero guardado para encontrar errores. Si es así, lo muestra en una ventana. Por ejemplo, si el conocimiento que hemos puesto contiene un error (falta un paréntesis en la última línea),

```
esPato(lucas).
esPato(donald).
esPato(gilito.
```

al utilizar la opción `Prolog->Compile Buffer` mostrará la pantalla



en la que se indica el error (o errores en caso de que haya varios). Pulsando dos veces sobre el error podemos ir directamente al editor en la línea en la que se encuentra dicho error.

Si todo va bien tras pulsar `Prolog->Compile Buffer` aparecerá en el sistema Prolog la siguiente información

```
% c:/prolog/marisol/patos.pl compiled 0.00 sec, 56 bytes
```

A partir de este momento, tenemos disponibles nuestras definiciones para realizar consultas a Prolog.

Ejecución de un programa Prolog

Ejecutar un programa Prolog consiste realmente en formular una pregunta a la que Prolog intentará responder haciendo uso del programa (base de conocimientos). Por ejemplo, para "ejecutar" el programa `'patos.pl'` podemos plantear a Prolog la siguiente cuestión

```
?- esPato(lucas).
Yes
```

Prolog responderá afirmativamente, pues es cierto que Lucas es un pato. Debe entenderse que Prolog sabe todo aquello que le hemos enseñado y sólo aquello que le hemos enseñado acerca de los patos. Es decir, Prolog no lo sabe todo acerca de los patos. Es más, ni siquiera sabe lo mismo que nosotros acerca de los patos. Así, aunque nosotros sabemos que Jorgito, el sobrino de Donald, es también un pato, no hemos facilitado a Prolog conocimiento suficiente como para inferir tal información. Si preguntamos si Jorgito es un pato

```
?- esPato(jorgito).
No
```

Prolog responderá negativamente.

Prolog sólo no es capaz de contestar si alguien en particular (Lucas, Donald, etc.) es o no un pato; es también capaz de enumerarnos todos los patos que conoce. Para ello, basta formular la siguiente pregunta

```
?- esPato(X).
```

donde `X` (en mayúscula) es una variable que representa a algún pato por determinar. La pregunta anterior puede leerse entonces como "dime los valores posibles de X tales que son patos". Esta pregunta tiene tres respuestas posibles, pues Prolog conoce tres patos: Lucas, Donald y Gilito. Cuando Prolog reconoce que una pregunta tiene varias respuestas, se detiene cada vez que muestra una de éstas y espera a que el programador le indique si desea obtener más respuestas. Por ejemplo, a la pregunta

```
?- esPato(X).
X = lucas
```

Prolog responde que conoce al pato Lucas, y se queda a la espera de que el programador le indique si quiere obtener o no más respuestas. Si pulsamos retorno de carro, le estaremos indicando a Prolog que nos damos satisfechos con la respuesta obtenida hasta el momento. Por el contrario, si queremos que Prolog nos muestre más respuestas, debemos teclear un punto y coma después de cada respuesta

```
?- esPato(X).
X = lucas;
X = donald;
X = gilito;
No
```

El predicado edit/1.

Ahora vamos a aumentar el conocimiento de Prolog acerca de los patos. Para ello será necesario que volvamos a editar el fichero `'patos.pl'`. Puesto que el fichero ya existe, no es necesario que empleemos el predicado `crea`. En esta ocasión editaremos el fichero mediante el predicado `edit`, que recibe como parámetro el nombre del fichero a editar

```
?- edit(patos).
```

Salvo cruces extraños y aberraciones genéticas, podemos suponer que algo es un pato si es sobrino de otro pato. Este conocimiento lo expresaremos mediante la siguiente **regla** (*verdad condicional*) Prolog

```
esPato(S) :- sobrino(S,T), esPato(T).
```

que añadiremos al fichero `'patos.pl'`. Podemos leer la anterior regla Prolog como "S es un pato *si* S es sobrino de T y T es un pato".

Para completar la anterior definición de pato, es necesario definir qué es un sobrino. Esto lo haremos añadiendo al fichero `'patos.pl'` los siguientes 3 hechos

```
sobrino(jorgito,donald).
sobrino(jaimito,donald).
sobrino(juanito,donald).
```

indicando con ello que Jorgito, Jaimito y Juanito son sobrinos de Donald. Una vez guardado el fichero y cerrado el editor, Prolog consultará de nuevo el programa almacenado en `'patos.pl'`, aprendiendo nuevas definiciones sobre lo que es un pato. Podemos comprobar si Prolog reconoce a Jorgito como pato, pues es sobrino del pato Donald

```
?- esPato(jorgito).
Yes
```

y también podemos pedirle a Prolog que nos indique qué patos conoce ahora

```
?- esPato(X).
...
```

El predicado `edit` nos permite editar la definición de un predicado en concreto. Para ello, basta facilitar a `edit` el nombre del predicado que deseamos editar, por ejemplo

```
?- edit(sobrino).
```

editará el fichero `'patos.pl'` (pues Prolog sabe que la definición de `sobrino` se encuentra en este fichero) y colocará el cursor del editor sobre la primera definición de `sobrino` que aparezca en `'patos.pl'`.

A veces puede ocurrir que Prolog no este seguro de qué es lo que se desea editar. Por ejemplo, un predicado puede estar defiinido en más de un fichero o puede ocurrir que un predicado y un

fichero tengan el mismo nombre. En tales casos, Prolog mostrará un menú al usuario mostrándole todas las opciones de edición posibles para que éste decida qué es lo que desea editar.

El predicado halt/0.

Para terminar una sesión Prolog, basta ejecutar el predicado `halt`

```
?- halt.
```

El predicado consult/1.

Un punto importante a destacar es que Prolog no recuerda lo que aprendió en las sesiones anteriores. Cada vez que se inicia una nueva sesión, Prolog parte sólo del conocimiento básico, independientemente del conocimiento que hubiera adquirido en las sesiones anteriores. Así, si damos por terminada (mediante el predicado `halt`) la sesión en que enseñamos a Prolog acerca de los patos, iniciamos una nueva sesión, y preguntamos si Donald es un pato

```
?- esPato(donald).
[WARNING Undefined predicate `esPato/1']
No
```

Prolog nos responderá que no sabe nada acerca de los patos. Para que Prolog vuelva a aprender todo lo que le enseñamos acerca de los patos basta con consultar el fichero 'patos.pl' mediante el predicado `consult`

```
?- consult(patos).
```

Una vez consultado el fichero, Prolog habrá adquirido nuevamente todo el conocimiento que le dimos acerca de los patos Para ampliar aún más ese conocimiento, basta editar el fichero `'patos.pl'` y añadir más hechos y reglas

```
?- edit(patos).
```

Esta vez trataremos de determinar aquello que caracteriza a un pato. Sabemos que un pato es algo que tiene plumas y hace "cuac". Podemos expresar tal conocimiento en la regla Prolog

```
esPato(P) :- tienePlumas(P), haceCuac(P).
```

que se lee "P es un pato *si* tiene plumas y hace cuac".

Para completar esta definición de pato, añadiremos ciertos hechos indicando quién tiene plumas

```
tienePlumas(piolin).
tienePlumas(daisy).
```

y quién hace "cuac"

```
haceCuac(daisy).
```

Una vez realizadas todas estas modificaciones, guardaremos el fichero y cerraremos el editor, regresando al sistema Prolog. Ahora podemos preguntar, por ejemplo, si Piolín es un pato

```
?- esPato(piolin).
No
```

a lo que Prolog responderá negativamente, pues aunque Piolín tiene plumas no hace "cuac", y por lo tanto no es un pato. Por el contrario, Daisy si es un pato, tal y como nos confirma Prolog

```
?- esPato(daisy).
Yes
```

Finalmente, podemos solicitar a Prolog un listado de todos los patos que conoce

```
?- esPato(X).
...
```

5. Ayuda de SWI-Prolog

SWI-Prolog viene equipado con un potente sistema de ayuda. En realidad, **todo el manual del usuario de SWI-Prolog está disponible a través de dos predicados Prolog**: `help/1` y `apropos/1`.

El predicado `help/0` muestra un mensaje sobre el funcionamiento del sistema de ayuda de SWI-Prolog.

El predicado `help/1` muestra una descripción detallada acerca de un predicado concreto o tema concreto

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

Finalmente, el predicado `apropos/1` muestra una lista de todos los predicados y secciones del manual que contienen una palabra en concreto

```
?- apropos(palabra).
```

Es decir, `apropos/1` genera la entrada del índice de palabras correspondiente a `'palabra'`.

La mejor forma de familiarizarse con el sistema de ayuda de SWI-Prolog es haciendo uso del mismo.