

**UNIVERSIDAD DON BOSCO
FACULTAD DE INGENIERIA
ESCUELA DE COMPUTACIÓN
SISTEMAS EXPERTOS DE INTELIGENCIA ARTIFICIAL**



“INVESTIGACIÓN SOBRE PROLOG”

CATEDRÁTICO:
ING. CRUZ ANTONIO GALDAMEZ

PRESENTADO POR:	
NANCY CAROLINA MONTERROSA CHEVEZ	MC000306
GUILLERMO ARMANDO PORTILLO TREJO	PT000041

CIUDADELA DON BOSCO, 23 DE JUNIO DE 2004

INTRODUCCION

El presente trabajo sobre el lenguaje de programación PROLOG, ha sido elaborado por alumnos que cursan la materia de Sistemas Expertos, de la Facultad de Ingeniería, en la Universidad Don Bosco.

El objetivo principal del estudio del tema, es conocer la forma en que se debe manipular el programa y los comandos utilizados por este, así como también, la manera en que se manejan las estructuras de control, de datos, las operaciones básicas y matemáticas.

Se considera de importancia el estudio del mismo, ya que ésta es una herramienta muy utilizada para la elaboración de Sistemas de Inteligencia Artificial, pero más que todo aplicado al área de los Sistemas Expertos, por la facilidad de elementos de lógica y heurística que maneja.

El trabajo consta de las siguientes partes:

Objetivos: Que son los que se pretenden alcanzar con el estudio del tema.

Desarrollo del trabajo: Aquí se describen las partes importantes de PROLOG como, la historia de esta aplicación, el tipo de datos que utiliza, los elementos del lenguaje, la estructura de los programas, entre otras cosas.

Código de Ejemplos: Aquí se muestra el código de los ejemplos para: estructura de datos, operaciones básicas, estructuras de control, operaciones matemáticas.

Conclusiones: Que son a las que se llegaron después de finalizada la investigación.

Para la elaboración del presente, se puso en práctica la investigación por Internet, así como también la aplicación PROLOG, esta última para poder elaborar los ejemplos requeridos. Y además se necesitó de la colaboración de los integrantes del grupo.

OBJETIVOS

General

- ✚ Investigar y conocer el lenguaje de programación PROLOG, para poder aplicarlo en la creación de programas básicos, y así poder familiarizarse con el uso de este en el ámbito de los Sistemas Expertos.

Específicos

- ✚ Detallar los tipos de datos que se pueden utilizar en PROLOG.
- ✚ Elaborar programas orientados a: estructura de datos, operaciones básicas, estructuras de control y operaciones matemáticas.
- ✚ Describir operadores y comandos básicos que son muy frecuentemente utilizados en PROLOG.
- ✚ Comprender la manera en que un lenguaje de programación como PROLOG interpreta los conocimientos y la lógica que sigue para detallar soluciones a través del motor de inferencia del mismo.

INTRODUCCIÓN A PROLOG

Prolog es un lenguaje de programación hecho para representar y utilizar el conocimiento que se tiene sobre un determinado dominio. Más exactamente, el dominio es un conjunto de objetos y el conocimiento se representa por un conjunto de relaciones que describen las propiedades de los objetos y sus interrelaciones. Un conjunto de reglas que describa estas propiedades y estas relaciones es un programa Prolog.

Prolog es un lenguaje de programación que es usado para resolver problemas que envuelven objetos y las relaciones entre ellos.

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.

En cuanto a la escasa utilidad práctica de Prolog podemos citar:

- Generación de CGI's.
- Acceso a bases de datos desde páginas Web.
- Paralelización automática de programas.
- Programación distribuida y multiagente.
- Sistemas expertos e inteligencia artificial.
- Validación automática de programas.
- Procesamiento de lenguaje natural.
- Prototipado rápido de aplicaciones.
- Bases de datos deductivas.
- Interfacing con otros lenguajes como Java y Tcl/Tk.

Prolog es aproximadamente diez veces más lento que el lenguaje C. Pero también, un programa en Prolog ocupa aproximadamente diez veces menos, en líneas de código y tiempo de desarrollo, que el mismo programa escrito en C. Además las técnicas de optimización de código en Prolog apenas están emergiendo en estos momentos. Algunos experimentos (optimistas) hacen pensar que la velocidad de ejecución de Prolog podría aproximarse a la de C en esta década.

Historia y desarrollo de Prolog

Una de las preocupaciones más tempranas de la computación de los años cincuenta fue la posibilidad de hacer programas que llevaran a cabo demostraciones automáticas de teoremas. Así empezaron los primeros trabajos de inteligencia artificial que más de veinte años después dieron lugar al primer lenguaje de programación que contempla, como parte del intérprete, los mecanismos de inferencia necesarios para la demostración automática. Este primer lenguaje está basado en el formalismo matemático de la Lógica de Primer Orden y ha dado inicio a un nuevo y activo campo de investigación entre las matemáticas y la computación que se ha denominado la Programación Lógica.

Estos mecanismos de prueba fueron trabajados con mucho entusiasmo durante una época, pero, por su ineficiencia, fueron relegados hasta el nacimiento de PROLOG, ocurrido en 1970 en la Universidad de Marsella, Francia, en el seno de un grupo de investigación en el campo de la Inteligencia Artificial.

La Programación Lógica tiene sus orígenes más cercanos en los trabajos de prueba automática de teoremas de los años sesenta. J. A. Robinson propone en 1965 una regla de inferencia a la que llama resolución, mediante la cual la demostración de un teorema puede ser llevada a cabo de manera automática.

La resolución es una regla que se aplica sobre cierto tipo de fórmulas del Cálculo de Predicados de Primer Orden, llamadas cláusulas y la demostración de teoremas bajo esta regla de inferencia se lleva a cabo por reducción al absurdo.

Actualmente, la programación lógica ha despertado un creciente interés que va mucho más allá del campo de la Inteligencia Artificial(IA) y sus aplicaciones. Los japoneses, con su proyecto de máquinas de la quinta generación, dieron un gran impulso a este paradigma de programación. Sin embargo, antes que ellos existían ya en Estados Unidos y en Europa grupos de investigación en este campo, en países como Inglaterra, Holanda, Suecia y, desde luego, Francia.

A principios de los años ochentas los japoneses comienzan a invertir recursos en un proyecto que denominan la Quinta Generación, para lucrar con la buena fama de los 4GL. Con este ambicioso proyecto Japón busca obtener el liderazgo en computación, usando como base la Programación Lógica y la Inteligencia Artificial.

La programación lógica tiene sus raíces en el cálculo de predicados, que es una teoría matemática que permite, entre otras cosas, lograr que un computador pueda realizar inferencias, capacidad que es requisito para que un computador sea una "máquina inteligente". La realización del paradigma de la programación lógica es el lenguaje Prolog.

El Prolog estuvo un tiempo diseñado para ejecutarse en minicomputadoras o estaciones de trabajo, actualmente hay versiones en Prolog que pueden instalarse en computadores personales como IBM-PC y PC-Compatible.

Un programa escrito en PROLOG puro, es un conjunto de cláusulas. Sin embargo, PROLOG, como lenguaje de programación moderno, incorpora más cosas, como instrucciones de Entrada/Salida, etc.

Una cláusula puede ser una conjunción de hechos positivos o una implicación con un único consecuente (un único termino a la derecha). La negación no tiene representación en PROLOG, y se asocia con la falta de una afirmación

(negación por fallo), según el modelo de suposición de un mundo cerrado solo es cierto lo que aparece en la base de conocimiento o bien se deriva de esta.

Las diferencias sintácticas entre las representaciones lógicas y las representaciones PROLOG son las siguientes:

1. En PROLOG todas las variables están implícitamente cuantificadas universalmente.
2. En PROLOG existe un símbolo explícito para la conjunción "y" (.), pero no existe uno para la disyunción "o", que se expresa como una lista de sentencias alternativas.
3. En PROLOG, las implicaciones $p \rightarrow q$ se escriben al revés $q :- p$, ya que el interprete siempre trabaja hacia atrás sobre un objetivo.

[El entorno de desarrollo Prolog](#)

Un entorno de desarrollo Prolog se compone de:

- **Un compilador.** Transforma el código fuente en código de byte. A diferencia de Java, no existe un standard al respecto. Por eso, el código de byte generado por un entorno de desarrollo no tiene por que funcionar en el intérprete de otro entorno.
- **Un intérprete.** Ejecuta el código de byte.
- **Un shell o top-level.** Se trata de una utilidad que permite probar los programas, depurarlos, etc. Su funcionamiento es similar a los interfaces de línea de comando de los sistemas operativos.
- **Una biblioteca de utilidades.** Estas bibliotecas son, en general, muy amplias. Muchos entornos incluyen (afortunadamente) unas bibliotecas standard-ISO que permiten funcionalidades básicas como manipular cadenas, entrada/salida, etc.

Generalmente, los entornos de desarrollo ofrecen extensiones al lenguaje como pueden ser la programación con restricciones, concurrente, orientada a objetos, etc.

SICStus, CIAO Prolog, y posiblemente otros más, ofrecen entornos integrados generalmente basados en Emacs que resultan muy fáciles de usar. CIAO Prolog además ofrece un autodocumentador similar al existente para Java además de un preprocesador de programas.

ELEMENTOS DEL LENGUAJE

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 se olvida 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 se cometa algún error al teclear una pregunta. **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, se habrá cometido un error sintáctico. Afortunadamente, Prolog es capaz de detectar tales errores y 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 advierte de que hay un error sintáctico (syntax error), mostrando la pregunta recién formulada y el punto en que se encuentra el error (** here **).

Comentarios

Los comentarios en Prolog se escriben comenzando la línea con un símbolo de porcentaje. Ejemplo:

% Hola, esto es un comentario.

% Y esto también.

Variables lógicas

Las variables en Prolog no son variables en el sentido habitual, por eso las llamamos variables lógicas. Se escriben como una secuencia de caracteres alfabéticos comenzando siempre por mayúscula o subrayado. Ejemplos de variables:

Variable

_Hola

—

Pero no son variables:

variable

\$Hola

p__

El hecho de que los nombres de variables comiencen por mayúscula (o subrayado) evita la necesidad de declarar previamente y de manera explícita las variables, tal y como ocurre en otros lenguajes.

La variable anónima

Existen variables sin nombre, y todas ellas se representan mediante el símbolo de subrayado _. Pero cuidado, aunque todas las variables anónimas se escriben igual, son todas **distintas**. Es decir, mientras que dos apariciones de la

secuencia de caracteres Hola se refieren a la misma variable, dos apariciones de la secuencia _ se refieren a variables distintas.

Tipos de datos en Prolog

Symbol

Hay dos tipos de símbolos:

1. Un grupo de caracteres consecutivos (letras, números y signos de subrayado) que comienzan con un carácter en minúscula

Ejemplo: `Alto,Alto_edificio,El_alto_edificio_en_la_ciudad`

2. Un grupo de caracteres consecutivos(letras y números) que comienzan y terminan con dobles comillas(""). Este tipo es útil cuando se quiere comenzar el símbolo con un carácter en mayúscula o si se quiere agregar espacios entre los caracteres del símbolo.

Ejemplo:`"alto","alto edificio"`

String

Cualquier grupo de caracteres consecutivos (letras y números) que comience y termine con dobles comillas(""). Es igual a símbolo pero Prolog los trata de forma distinta.

Ejemplo:`"alto","alto edificio"`

Integer

Cualquier numero comprendido entre (-32.768 y 32.768). El limite esta determinado porque los enteros se almacenan como valores de 16 bits, este limite puede variar según la versión de Prolog.

Ejemplo:`4,-300,3004`

Real

Cualquier número real en el rango +/- 1E-307 a +/-1E+308. El formato incluye estas opciones: signo, número, punto decimal, fracción, E(exponente), signo para el exponente, exponente.

Ejemplo:3,3.1415

Char

Cualquier carácter de la lista ASCII estándar, posicionado entre dos comillas sencillas(').

Ejemplos:'t','X','f'

ESTRUCTURA DE UN PROGRAMA PROLOG

Un programa Prolog está formado por una secuencia de enunciados: hechos, reglas y comentarios.

Una relación puede estar especificada por hechos, simplemente estableciendo objetos que satisfacen la relación o por reglas establecidas acerca de la relación. Cada regla está formada por un primer miembro (o la cabeza de la regla), un segundo miembro (o cola de la regla) ligados por " :- " y termina con el carácter " . ".

código del programa

**** Hechos ****

mujer(maria).

hombre(pedro).

hombre(manuel).

hombre(arturo).

**** Relaciones ****

padre(pedro,manuel).

padre(pedro,arturo).

padre(pedro,maria).

**** Reglas ****

nino(X,Y):- padre(Y,X)

hijo(X,Y):-nino(X,Y),hombre(X).

hija(X,Y):-nino(X,Y),mujer(X).

hermano_o_hermana(X,Y):-padre(Z,X),padre(Z,Y).

hermano(X,Y):-hermano_o_hermana(X,Y),hombre(X).

hermana(X,Y):-hermano_o_hermana(X,Y),mujer(X).

Hechos

Expresan relaciones entre objetos. Supongamos que queremos expresar el hecho de que "un coche tiene ruedas". Este hecho, consta de dos objetos, "coche" y "ruedas", y de una relación llamada "tiene". La forma de representarlo en PROLOG es:

tiene(coche,ruedas).

* Los nombres de objetos y relaciones deben comenzar con una letra minúscula.

* Primero se escribe la relación, y luego los objetos separados por comas y encerrados entre paréntesis.

*Al final de un hecho debe ir un punto (el carácter ".").

El orden de los objetos dentro de la relación es arbitrario, pero deben ser coherentes a lo largo de la base de hechos.

Variables

Representan objetos que el mismo PROLOG determina. Una variable puede estar instanciada o no instanciada. Esta instanciada cuando existe un objeto

determinado representado por la variable. De este modo, cuando preguntamos "Un coche tiene X ?", PROLOG busca en los hechos cosas que tiene un coche y respondería:

X = ruedas.

instanciando la variable X con el objeto ruedas.

* Los nombres de variables comienzan siempre por una letra mayúscula.

Un caso particular es la variable anónima, representada por el carácter subrayado ("_"). Es una especie de comodín que utilizaremos en aquellos lugares que debería aparecer una variable, pero no nos interesa darle un nombre concreto ya que no vamos a utilizarla posteriormente.

Reglas

Las reglas se utilizan en PROLOG para significar que un hecho depende de uno o más hechos. Son la representación de las implicaciones lógicas del tipo $p \rightarrow q$ (p implica q).

* Una regla consiste en una cabeza y un cuerpo, unidos por el signo ":-".

* La cabeza esta formada por un único hecho.

* El cuerpo puede ser uno o más hechos (conjunción de hechos), separados por una coma (","), que actúa como el "y" lógico.

* Las reglas finalizan con un punto (".").

La cabeza en una regla PROLOG corresponde al consecuente de una implicación lógica, y el cuerpo al antecedente. Este hecho puede conducir a errores de representación. Supongamos el siguiente razonamiento lógico:

tiempo(lluvioso) ----> suelo(mojado)

suelo(mojado)

Que el suelo esta mojado, es una condición suficiente de que el tiempo sea lluvioso, pero no necesaria. Por lo tanto, a partir de ese hecho, no podemos deducir mediante la implicación, que esta, lloviendo (pueden haber regado las calles). La representación *correcta* en PROLOG, sería:

suelo(mojado) :- tiempo(lluvioso).
suelo(mojado).

Cabe señalar que la regla esta "al revés". Esto es así por el mecanismo de deducción hacia atrás que emplea PROLOG. Si cometiéramos el *error* de representarla como:

tiempo(lluvioso) :- suelo(mojado).
suelo(mojado).

PROLOG, partiendo del hecho de que el suelo esta mojado, deduciría incorrectamente que el tiempo es lluvioso.

Para generalizar una relación entre objetos mediante una regla, utilizaremos variables. Por ejemplo:

Representación lógica | Representación PROLOG

Es un coche(X) ----> | tiene(X,ruedas) :
tiene(X,ruedas) | es un coche(X).

Con esta regla generalizamos el hecho de que cualquier objeto que sea un coche, tendrá ruedas. Al igual que antes, el hecho de que un objeto tenga ruedas, no es una condición suficiente de que sea un coche. Por lo tanto la representación inversa sería incorrecta.

El ámbito de las variables.

Cuando en una regla aparece una variable, el ámbito de esa variable es únicamente esa regla. Supongamos las siguientes reglas:

(1) `hermana_de(X,Y) :- hembra(X), padres(X,M,P), padres(Y,M,P).`

(2) `puede_robar(X,P) :- ladron(X), le_gusta_a(X,P), valioso(P).`

Aunque en ambas aparece la variable X (y la variable P), no tiene nada que ver la X de la regla (1) con la de la regla (2), y por lo tanto, la instanciación de la X en (1) no implica la instanciación en (2). Sin embargo todas las X de *una misma regla* si que se instanciaran con el mismo valor.

ESTRUCTURAS DE DATOS EN PROLOG

Listas:

La lista es una estructura de datos muy común en la programación no numérica. Es una secuencia ordenada de elementos que puede tener cualquier longitud. Ordenada significa que el orden de cada elemento es significativo. Un elemento puede ser cualquier término e incluso otra lista. Se representa como una serie de elementos separados por comas y encerrados entre corchetes.

El valor de las listas en un programa Prolog disminuye si no es posible identificar los elementos individuales que habrán de integrarlas. Debido a ello, es necesario tener en cuenta el concepto de su división en dos partes: cabeza y cola. La cabeza de la lista es el primer elemento de la misma. La cola es el resto de la lista, sin importar lo que pueda contener.

Para hacer uso practico de la capacidad de dividir listas en cabeza y cola, el Prolog proporciona una notación especial con la que se definen las listas en los programas.

Existen dos símbolos especiales que se utilizan:

1. El corchete abierto/cerrado. Se usa para denotar el inicio y el final de una lista
2. El separador. Su símbolo es | y se usa para permitir que una lista se represente como una cabeza y una cola.

Para procesar una lista, la dividimos en dos partes: la cabeza y la cola. Por ejemplo:

Lista Cabeza Cola

[a,b,c,d] a [b,c,d]

[a] a [] (lista vacia)

[] no tiene no tiene

[[a,b],c] [a,b] [c]

[a,[b,c]] a [[b,c]]

[a,b,[c,d]] a [b,[c,d]]

Para dividir una lista, utilizamos el símbolo "|". Una expresión con la forma [X | Y] instanciar X a la cabeza de una lista e Y a la cola. Por ejemplo:

p([1,2,3]).

p([el,gato,estaba,[en,la,alfombra]]).

?-p([X|Y]).

X = 1,

Y = [2,3]

X = el,

Y = [gato,estaba,[en,la,alfombra]]

Lista de listas, se pueden hacer estructuras tan complejas como se quiera en Prolog. Se pueden poner listas dentro de listas.

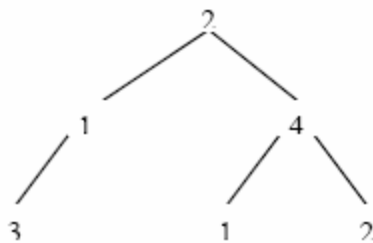
Ejemplo:

Animales([[mono,bufalo,rata], [serpiente,iguana,cocodrilo], [paloma,pingüino]])

Árboles:

Es posible implementar otros tipos de datos con estructuras. Un ejemplo común son los árboles binarios. Para representar un árbol podemos utilizar el functor árbol con tres argumentos: valor, árbol izquierdo y árbol derecho. El árbol nulo lo representaremos como a_nulo.

Ej.



Este árbol se representará de la siguiente manera:

```
arbol(2, arbol(1, arbol(3, a_nulo, a_nulo), a_nulo),  
        arbol(4, arbol(1, a_nulo, a_nulo), arbol(2, a_nulo, a_nulo) ) )
```

ESTRUCTURAS DE CONTROL

En Prolog, a diferencia de lenguajes procedurales como Pascal o C, no existen estructuras de control para bucles. Éstos se implementan mediante predicados recursivos. En cambio existen estructuras de control nuevas que no existen en otros lenguajes y que describiremos a continuación.

- **AND**

El “AND” o Y lógico se representa en Prolog mediante la coma ‘,’. Este operador ya se ha visto en detalle en todos los ejemplos utilizados hasta ahora.

- **OR**

El “OR” o O lógico se puede realizar en Prolog de dos formas distintas: mediante varias cláusulas para un mismo predicado o mediante el operador ‘;’. Mediante varias cláusulas se consigue poniendo cada una de las opciones en una cláusula distinta del predicado. Entre las distintas cláusulas de un mismo predicado se puede considerar que existe un “OR”. Esto de hecho ya se ha utilizado por ejemplo en el factorial (se define mediante dos cláusulas) o en el predicado padre/2.

El operador ‘;’ se utiliza igual que el operador ‘,’. Únicamente hay que prestar especial atención a la prioridad de los operadores, ya que ‘;’ tiene menor prioridad que ‘,’ lo que significa que normalmente habrá que encerrar entre paréntesis la operación. A este respecto hay que recordar que el “OR” se considera una suma lógica y el “AND” una multiplicación lógica.

Ej.

Escribir un predicado numero/1 que sea cierto para los números enteros o reales y falso en caso contrario.

Varias cláusulas Operador ‘;’

```
numero(X):- numero(X):-  
integer(X). integer(X);  
numero(X):- real(X).  
real(X).
```

Normalmente se utiliza la disyunción mediante cláusulas puesto que resulta más claro.

- **NOT**

La negación se realiza mediante el operador not. El operador 'not' antes de la llamada a un predicado P cambia su valor de verdad, es decir, si el predicado P tiene éxito, not P fallará y si el predicado P falla, not P tendrá éxito.

Ej.

no_entero(X):-

not integer(X).

Sin embargo hay que tener la precaución de aplicar la negación únicamente en llamadas a predicados donde todas las variables existentes estén ya instanciadas, ya que si no el comportamiento no es el esperado.

Ej.

padre(pepe, juan).

huerfano(X):-

not padre(Z, X).

Para la pregunta huerfano(pepe) funcionará bien y dirá que sí, pero si preguntamos quién es huérfano con huerfano(X) dirá que no, es decir, que no hay ningún huérfano.

- **Corte**

El operador de corte '!' lo que hace es limitar el backtracking. Cuando se ejecuta elimina todos los puntos de backtracking anteriores dentro del predicado donde está definido (incluido el propio predicado)

Se puede considerar que lo que hace es cortar la entrada al puerto de REDO.

Existen dos razones fundamentales para utilizar el corte:

1. Para aumentar la eficiencia. Se eliminan puntos de backtracking que se sabe que no pueden dar ninguna solución. Por ejemplo, cuando hemos encontrado una solución y sabemos que no existen más.

2. Para modificar el comportamiento del programa. Se eliminan puntos de backtracking que podían dar soluciones válidas. Esto es muy peligroso pues se está modificando el significado del programa. Este tipo de corte debe utilizarse lo menos posible.

- **Fallo, Cierto**

El predicado de fallo, *fail*, se utiliza para obligar al Prolog a dar un fallo.

El predicado cierto, *true*, se utiliza como instrucción nula, es decir, cuando se tiene que escribir una instrucción pero no se quiere que haga nada.

Un ejemplo típico es para imitar las instrucciones *if-then*:

cero(X):-

(X == 0, write('Cero');

true

).

- **Repeat**

El predicado repeat se utiliza para crear un punto de backtracking infinito. Se puede considerar que está definido de la siguiente manera:

repeat.

repeat:- repeat.

Se utiliza típicamente para menús o en entradas de datos.

Ej.

menu(X):-

```
repeat,
write('Opcion:'),
read(X),
(X == 1; X == 2; X == 3).
```

Resumiendo lo antes mencionado, presentamos el siguiente cuadro, donde se detalla cada una de las estructuras de control utilizadas por PROLOG

Estructuras de control

,	Es un "and" entre objetivos. Si una cláusula tiene una secuencia de objetivos separados por ",", todos los objetivos deben cumplirse para que la cláusula sea verdad.
;	Es un "or" entre objetivos. Todos los objetivos de un sólo lado de la disyunción deben ser ciertos para que la disyunción sea cierta.
!	"Cut" o "corte" es un objetivo que siempre se cumple. El corte limita la vuelta atrás ("backtracking"), de forma que si en la vuelta atrás se encuentra un corte, entonces el predicado entero que contiene el corte falla, no sólo la cláusula.
not P	El término P es interpretado como un objetivo. Si P tiene éxito, entonces not P falla y si P falla, not P tiene éxito.
true	Es un objetivo que siempre tiene éxito.
fail	Es un objetivo que siempre falla. Útil para provocar backtracking.
repeat	Este objetivo siempre es cierto y siempre deja un punto de backtracking.
(P -> Q)	If...Then. Si el objetivo P tiene éxito, se ejecuta el predicado Q. P no debe contener ningún corte (!). No se realiza backtracking sobre P.
(P -> Q ; R)	If...Then...Else. Si el objetivo P tiene éxito, se ejecuta el término Q. Si P falla, se ejecuta el término R. En ningún caso se realizará backtracking sobre P. P no debe contener ningún corte (!).

Operaciones Matemáticas

Son predicados predefinidos en PROLOG para las operaciones matemáticas básicas. Su sintaxis depende de la posición que ocupen, pudiendo ser infijos o prefijos. Por ejemplo el operador suma ("+"), podemos encontrarlo en forma prefija `+(2,5)` o bien infija, `'2 + 5'`.

También dispone de predicados de igualdad y desigualdad.

`X = Y` igual

`X \= Y` distinto

`X < Y` menor

`X > Y` mayor

`X =< Y` menor o igual

`X >= Y` mayor o igual

Al igual que en otros lenguajes de programación es necesario tener en cuenta la precedencia y la asociatividad de los operadores antes de trabajar con ellos.

En cuanto a precedencia, es la típica. Por ejemplo, `3+2*6` se evalúa como `3+(2*6)`. En lo referente a la asociatividad, PROLOG es asociativo por la izquierda. Así, `8/4/4` se interpreta como `(8/4)/4`. De igual forma, `5+8/2/2` significa `5+((8/2)/2)`.

El operador 'is'.

Es un operador infijo, que en su parte derecha lleva un término que se interpreta como una expresión aritmética, contrastándose con el término de su izquierda.

Por ejemplo, la expresión `'6 is 4+3.'` es falsa. Por otra parte, si la expresión es `'X is 4+3.'`, el resultado será la instanciación de X:

$X = 7$

Una regla PROLOG puede ser esta:

$\text{densidad}(X,Y) \text{ :- población}(X,P), \text{área}(X,A), Y \text{ is } P/A.$

Operadores y predicados aritméticos

Operador	Explicación	Operador	Explicación
+	Suma	-	Resta
*	Producto	/	División
//	División entera	^	Exponenciación
mod	Resto de la división	abs()	Valor absoluto
sin()	Seno	cos()	Coseno
tan()	Tangente	asin()	Arco seno
acos()	Arco coseno	atan()	Arco tangente
ln()	Logaritmo neperiano	exp()	Potencia neperiana
sqrt()	Raíz cuadrada	fix()	Trunca a entero

Operadores aritméticos de evaluación

Operador	Explicación	Operador	Explicación
>	Mayor que	<	Menor que
>=	Mayor o igual que	=<	Menor igual que
=:=	Igual que	!=	Distinto de

Ejemplo 1: Árbol Genealógico

```
% Este es un ejemplo de un programa en Prolog

% Se trata de un árbol genealógico muy simple

% Primero definimos los parentescos básicos de la familia.

% padre(A,B) significa que B es el padre de A...

padre(juan,alberto).

padre(luis,alberto).

padre(alberto,leoncio).

padre(geronimo,leoncio).

padre(luisa,geronimo).


% Ahora definimos las condiciones para que

% dos individuos sean hermanos

% hermano(A,B) significa que A es hermano de B...

hermano(A,B) :-
    padre(A,P),
    padre(B,P),
    A \== B.

% Ahora definimos el parentesco abuelo-nieto.

% nieto(A,B) significa que A es nieto de B...

nieto(A,B) :-
    padre(A,P),
    padre(P,B).
```

Ejemplo2: Cálculo de un Factorial

%Ejemplo del Cálculo del Factorial en PROLOG

factorial(0,1).

factorial(N,F) :-

 N>0,

 N1 is N-1,

 factorial(N1,F1),

 F is N * F1.

CONCLUSIONES

PROLOG es un lenguaje de programación fácil de utilizar, y rápido de entender, ya que la lógica a utilizar se asemeja mucho a lo que pensamos o tratamos de decir. Además es un lenguaje muy comúnmente utilizado para el desarrollo de Sistemas Expertos, por lo que es importante conocerlo y aprender a usarlo.

El desarrollo de los ejemplos sirvió para la mejor comprensión de la forma de manipulación del programa, así como también, las sintaxis que se utilizan y los comandos que nos pueden servir para el desarrollo de una mejor aplicación.