

Prolog

PROGRAMACIÓN DECLARATIVA

SU PRIORIDAD ES RESPONDER A LA SIGUIENTE PREGUNTA

¿QUÉ PROBLEMA HAY QUE RESOLVER?

NO SE PREOCUPA DE ¿CÓMO HAY QUE RESOLVER EL PROBLEMA?

Prolog

Prog. Imperativa vs. Declarativa

La programación lógica, junto con la funcional, forma parte de lo que se conoce como programación declarativa.

En los lenguajes imperativos, la programación consiste en indicar cómo resolver un problema mediante sentencias;

En la programación lógica, se trabaja de una forma descriptiva, estableciendo relaciones entre entidades, indicando no cómo, sino qué hacer.

Base de conocimiento



Base de conocimiento

Las cláusulas o enunciados que conforman la base de conocimientos declaran la forma de demostrar o deducir constructivamente un objetivo a partir del programa.

Estas cláusulas se denominan 'Cláusulas de Horn'

Base de conocimiento

Para responder a las preguntas o consultas formuladas por el programador, Prolog consulta una base de conocimiento. Ésta base de conocimiento representa el programa como tal, programa que se compone únicamente de cláusulas, que con el uso de la lógica, expresan el conocimiento deseado por el programa.

La base de conocimiento o el programa se guarda en un archivo con la extensión **'*pl*'**, archivo que puede ser abierto y a partir de esto poderle hacer consultas a mi programa.

A continuación se muestra el comando para abrir un programa desde la consola de consultas:

```
≡ ?- consult('nombre_archivo.pl').
```

Prolog

Cláusula de Horn

Es un predicado con una sola conclusión y un conjunto de premisas de cuyo valor de verdad se deriva o deduce el valor de verdad de la conclusión. $c:- pr1, pr2, pr3, pr4$.

Una conclusión es verdadera si lo son todas sus premisas.

Prolog

Carácter declarativo:

Un programa lógico NO tiene un algoritmo que indique los pasos a seguir para llegar al resultado, sino que está formado por expresiones lógicas que declaran o describen la solución y es el sistema interno quien proporciona la secuencia de control en que se utilizan esas expresiones.

Prolog

Aplicaciones:

- ▶ Inteligencia Artificial
- ▶ Sistemas expertos
- ▶ Procesamiento de Lenguaje Natural
- ▶ Compiladores
- ▶ Bases de datos deductivas
- ▶ Acceso a bases de datos desde páginas web

Prolog - Usos

Asignación de Recursos Limitados

IBM desarrollo **EI – IA** para asignación de aviones en líneas aéreas

Verificación de Circuitos digitales

CVE desarrollado por Siemens para verificar de forma automática la salida de un circuito digital

Aplicaciones

A continuación se listan algunas aplicaciones de Prolog: Prolog puede resolver básicamente cualquier tipo de problema.

- ▶ Investigación en inteligencia artificial
- ▶ Gestión de juegos.
- ▶ Sistemas expertos que emulan la habilidad de un humano para la toma de decisiones.
- ▶ Software "clarissa" construido por la **NASA** para la **ISS**. Es una interfaz de voz que busca los procesos de la estación.
- ▶ **SICStus Prolog** se encarga de la logística de la reservación de boletos de aerolíneas y ayudar a los ferrocarriles a operar mejor.
- ▶ Académico.
- ▶ Investigación en inteligencia artificial

Prolog

PROLOG PROgrammation en LOGique

Este lenguaje es el principal representante del paradigma.

La base conceptual de la lógica proposicional es desarrollada por Alfred Horn en los años 50.

Philippe Roussel y Alain Colmerauer (Universidad de Aix- Marsella) lo crearon en 1972, y su base teórica se debe en gran parte a Kowalski

Prolog

Estructuras básicas del lenguaje

Prolog cuenta con dos tipos de estructuras: términos y sentencias.

Los términos pueden ser constantes, variables o funtores:

Las constantes, representadas por una cadena de caracteres, pueden ser números o cualquier cadena que comience en minúscula.

Las variables son cadenas que comienzan con una letra mayúscula.

Los funtores son identificadores que empiezan con minúscula, seguidos de una lista de parámetros (términos) entre paréntesis, separados por comas.

Términos

Los términos en prolog son los componentes que conforman el lenguaje, y en este caso éstos van a ser los únicos elementos que componen un programa.

Existen tres (3) tipos de términos:

Constantes: Átomo y Número

Átomo o Functor: Son nombres de objetos, propiedades, o relaciones. Estos deben empezar en minúscula.

```
1 atomo(luis).  
2 atomo(color).  
3 atomo(padre).  
4 atomo('pedro').
```

Términos

Número: Valores que solo pueden ser entero o reales, pueden llevar el signo.

Ejemplos de las diferentes formas de expresar un número:

```
1 numero(2).  
2 numero(216565).  
3 numero(1.54521).  
4 numero(-5).  
5 numero(-5.0).  
6 numero(2e10).
```

Términos

Variables: Se representan mediante cadenas representadas por **letras**, **números** o por el símbolo '_', para que Prolog las tome como variables, éstas deben empezar en mayúscula o con '_'.

Una **variable anónima** se representa por el nombre '_' con la cual en cada instancia de ésta variable se refiere a una variable distinta.

A continuación se muestran algunos ejemplos:

```
1 variable(X).  
2 variable(Variable).  
3 variable(_).  
4 variable(_var).
```

Términos

Estructuras: Estos son términos compuestos por otros términos, donde la sintaxis que se tiene es la siguiente:

nombre_atomo(termino1, termino2, ..., terminoN).

Donde esos terminos del 1 al N, se les llama **argumentos** . Además, al nombre del atomo tambien se le llama **predicado**.

A continuación se muestran algunos ejemplos de estructuras:

```
1 padre(luis). % Estructura que toma un solo argumento.  
2 edad(luis, 30). % Estructura que ya se compone por más de un argumento.  
3 color(X). % Estructura con atomo llamado color y con un argumento que es una variable.
```

Prolog

HECHOS

- Propositiones:

“Juan es un programador”

“El león es un mamífero”

Se expresa en Prolog:

`programador(juan).`

`mamifero(leon).`

- Nótese que se anota primero el predicado y entre paréntesis el sujeto de la proposición

Prolog

RELACIONES

“Juan es el padre de Miguel”

Se expresa en prolog como:

padre(juan, miguel).

REGLAS

Sentencias condicionales

“Si el león come carne, entonces es carnívoro”

Se expresa en prolog como:

carnivoro(leon):- comecarne(leon)

Prolog

CONJUNCIONES

Emplea el operador lógico AND

Se utiliza la coma (,)

`tia(X,Y):-hermana(X,Z),padre(Z,Y).`

DISYUNCIONES

Emplea el operador lógico OR

Se utiliza el punto y coma (;)

`hijo(X,Y):-padre(Y,X);madre(Y,X).`

Prolog

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.

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 puede dar también respuestas negativas a las preguntas

?- 1 is $1+1$.

No

Prolog

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

Prolog

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

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`

Prolog

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
```

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
```


Prolog

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`

Prolog

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 tío Gilito son patos.

Podemos expresar este conocimiento en Prolog mediante 3 hechos (verdades incondicionales) tal y como sigue

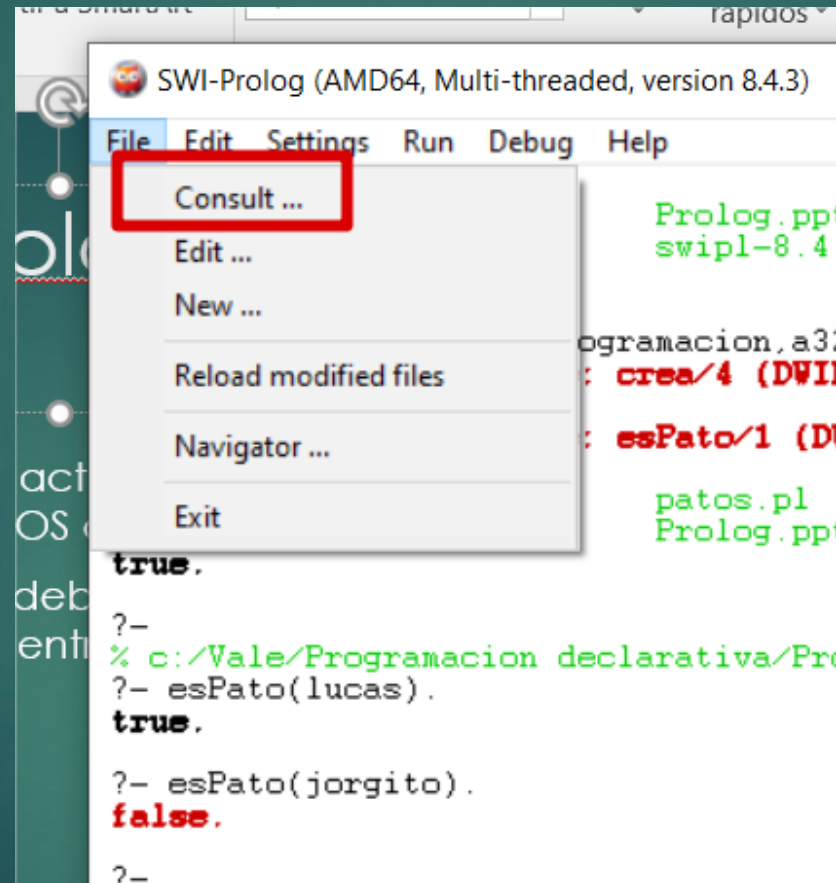
```
esPato(lucas).
```

```
esPato(donald).
```

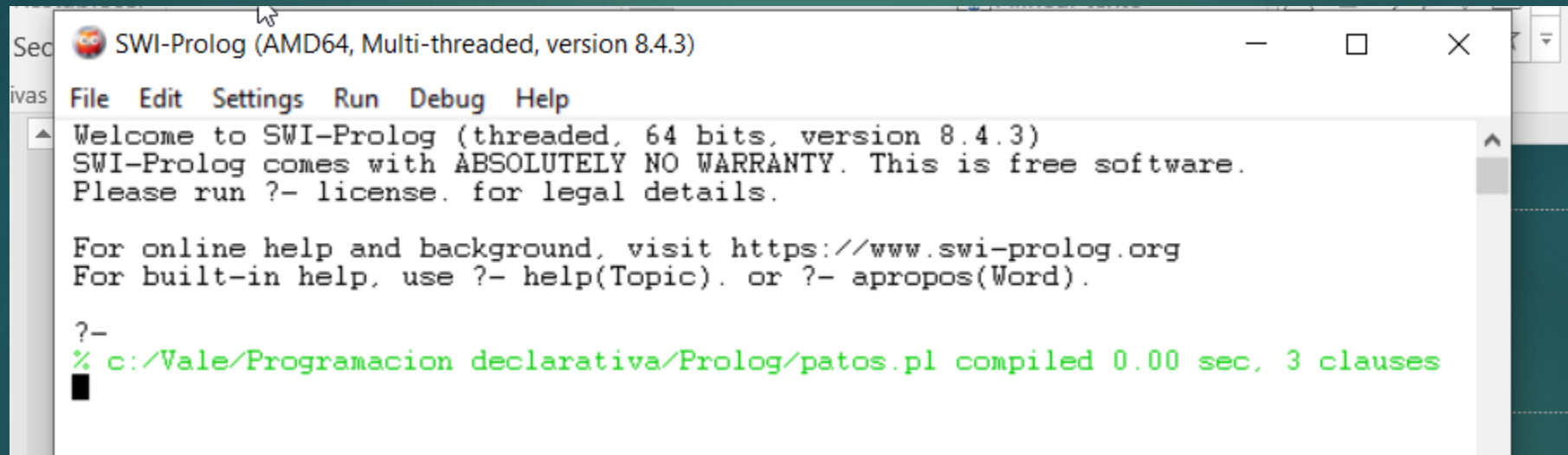
```
esPato(gilito).
```

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'

Compilar y cargar un archivo



Compilar y cargar un archivo



The screenshot shows a window titled "SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)". The window has a menu bar with "File", "Edit", "Settings", "Run", "Debug", and "Help". The main text area displays the following content:

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Vale/Programacion declarativa/Prolog/patos.pl compiled 0.00 sec, 3 clauses
```

Prolog se ejecuta en un ciclo REPL (Read-Eval-Print Loop), lo que significa que una vez ejecutado, el sistema está a la espera de una instrucción para evaluarla e imprimir el resultado. Eso se indica con el prompt `?-`. Asumiendo que existe el siguiente programa bajo el nombre `abuelo.pl`:

```
1  %%%padre(X,Y): X es padre de Y
2
3  padre(luis,ana).
4  padre(juan,luis).
5
6  %%%abuelo(X,Y): X es abuelo de Y
7
8  abuelo(X,Y) :-
9      padre(X,Z), padre(Z,Y).
```

el programa puede cargarse en Prolog de la siguiente manera:

```
1  ?- [abuelo].
2  true.
```

Una vez cargado un programa, se le pueden plantear metas al mismo. Por ejemplo:

```
1  ?- padre(luis,ana).
2  true.
3
4  ?- padre(luis,X).
5  X = ana.
6
7  ?- abuelo(X,ana).
8  X = juan.
```

Si el programa se encuentra en otro directorio, puede ser cargado mediante una cadena de texto que indique el camino completo hasta el programa:

```
1  ?- ["Dropbox/code/prolog/ia2/cap03/abuelo.pl"].
2  true.
3
4  ?- abuelo(X,ana).
5  X = juan.
```

Para salir de Prolog, use el comando `halt`:

```
1  ?- halt.
```

Comentarios

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

Ejemplo:

% Hola, esto es un comentario.

% Y esto también.

También como en C: /* Comentario */

Ejecución de un programa

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, ya 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

Edición de programas

Ahora vamos a aumentar el conocimiento de Prolog acerca de los patos. Para ello será necesario que volvamos a editar el fichero 'patos.pl'.

En esta ocasión editaremos el fichero mediante el predicado edit, que recibe como parámetro el nombre del fichero a editar

?- edit(patos).

Agregaremos

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

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).

Edición de un programa

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).
```

Para terminar una sesión Prolog, basta ejecutar el predicado halt

```
?- halt.
```


Prolog - Entrada /Salida

Se usa el comando write para desplegar un texto o una variable en la pantalla

```
write('Hola...').
```

Se usa el comando read para capturar desde el teclado

```
write('Anote su nombre:'), read(Nombre), nl, write('Hola  
' ),write(Nombre).
```

Nótese que la variable Nombre inicia con mayúscula.

Para limpiar la pantalla se puede escribir: `write('\33\[2J')`.

Operadores Aritméticos

Los operadores nos permiten manipular diferentes tipos de datos.

Operadores aritméticos.

Con estos podemos llevar a cabo operaciones aritméticas entre números de tipo entero o real, sin embargo se tuvieron en cuenta sólo los básicos, pero existen para las funciones trigonométricas, valor absoluto, piso, techo, entre otros muchas más.

La notación de Positivo y Negativo es prefijo, el resto es infijo

Operadores Aritméticos

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/ y //	División real y entera
^ y **	Potencia
+	Positivo
-	Negativo

Operadores Relacionales

Las operaciones relacionales nos permiten establecer relaciones de orden.

- Operadores relacionales con evaluación.

Este tipo de operadores recibe valores numéricos y/o expresiones antes de realizar unificación o comparaciones evalúa el valor de la expresión.

Operador	Significado	Ejemplo
is	Unificación	$X \text{ is } 10 + 2$
$=:=$	Igualdad	$10 + 2 =:= 5 + 7$
\neq	Desigualdad	$10 + 2 \neq 5 + 8$
$>$	Mayor que	$11 * 3 > 3 ^ 2$
$<$	Menor que	$2 ** 10 < 5 * 2$
$>=$	Mayor o igual que	$99.0 >= 0$
$=<$	Igual o menor que	$-15 = < 15$

Operadores Relacionales

Operadores de listas.

Las operaciones en listas nos permiten consultar alguna propiedad de una lista, así como realizar modificaciones.

Operador	Significado	Ejemplos
=	Unificación	$[X, Y, Z] = [a, 1, 2.0]$ $[X, Y Z] = [b, 2, 3.0]$
member(term, list)	$\text{term} \in \text{list}$	member(4.0, [c, 3, 4.0]). member(X, [c, 3, 4.0]).
append(list1, list2, result)	Une list1 con list2	append([h, o], [l, a], X). append([h, o], X, [h, o, l, a]). append(X, [l, a], [h, o, l, a]). append(X, Y, [h, o, l, a]).
length(list, result)	Calcula la longitud de la lista	length([3, 0.0, x], X).
sort(list, result)	Ordena la lista	sort([4, a, 3], X).
is_list(term)	Comprueba si term es lista	is_list([a, list]).

Cláusulas

Las cláusulas en Prolog están basadas en cláusulas de Horn.

$$p_1 \wedge p_2 \wedge \dots \wedge p_m \Rightarrow p$$

Lo cual sería equivalente a tener en Prolog:

$p :- p_1, p_2, \dots, p_m.$

Donde p es la Cabeza y todos los p_i son el Cuerpo, y cada uno de ellos son Functores.

Ejemplos:

$\text{planeta}(\text{marte}). : \text{Marte es un planeta.}$

$\text{hombre}(\text{'Felipe'}). : \text{Felipe es un hombre.}$

$\text{mato}(\text{hombre}(_), X). : \text{hombre}(_) \text{ mató a } X$

Cláusulas

```
1 come(A,B) :-  
2     carnivoro(A), animal(B), masDebil(B, A);  
3     herbivoro(A), plantaComestible(B).
```

Reescritas a lenguaje natural:

"A come a B si, A es carnívoro y B es animal y B es más débil que A, o si A es herbívoro y B es una planta comestible."

Tipos de cláusulas:

- Una cláusula con cabeza y cuerpo es llamada **Regla**.
- Sin cuerpo es un **Hecho** o **Afirmación**.
- Sin cabeza es una **Pregunta** o **Consulta**.

Hechos

Un hecho es un mecanismo para representar propiedades o relaciones de los objetos que se están representando. Los hechos declaran los valores que van a ser **verdaderos o afirmativos** para un predicado en todo el programa.

Los hechos siguen la siguiente sintaxis: **nombre_predicado(argumentos)**.

Los hechos se dividen en 2 tipos: **propiedades** y **relaciones**.

Hechos

- **Propiedades:** las propiedades se caracterizan por llevar un solo argumento y de esta manera expresan una propiedad de los objetos. Por ejemplo:

```
1 color(azul).% azul es color - Denota la propiedad del azul de ser un color
2 color(verde). % verde es color
3
4 padre(juan).%Juan es padre - Denota la propiedad que tiene juan y es la de ser padre.
5 padre(pablo). % Pablo es padre
```

- **Relaciones:** las relaciones se caracterizan por llevar más de un argumento y de esta manera expresan la relación entre varios objetos. Por ejemplo:

```
1 padrede('juan', 'maria'). % Juan es padre de maria - Este hecho expresa una relacion de padre-hijo
2 padrede('pablo', 'juan'). % Pablo es padre de juan
3
4 edad(juan, 30). % Juan tiene la edad de 30 años - Este hecho está relacionando a juan con un edad de 30 años, expresando una verdad o afirmación
5 edad('pablo', 50).
```

Reglas

Cuando la verdad de un hecho depende de la verdad de otro hecho o de un grupo de hechos se usa una regla.

Permiten establecer relaciones más elaboradas entre objetos donde se declaran las condiciones para que un predicado sea cierto, combinando hechos para dar el valor de verdad del predicado.

La sintaxis base para una regla es la siguiente:

CABEZA :- CUERPO

La forma como se debe leer esta sintaxis es de la siguiente manera:

“La cabeza es verdad si el cuerpo es verdad”.

De esta manera se obtendrá el valor de verdad de la cabeza con el valor que se obtenga en el cuerpo, si el cuerpo resulta falso, la cabeza será falsa.

Reglas

```
1 %A es hijo de B si B es padre de A  
2 hijode(A,B) :- padrede(B,A).  
3  
4 % A es abuelo de B si A es padre de C y C es padre B  
5 abuelode(A,B) :- padrede(A,C), padrede(C,B).
```

Reglas

Las reglas se pueden dividir en 2 tipos, estos dependiendo de como se calcula el valor de verdad del cuerpo:

Conjunciones: Se usa una **coma** para separar los hechos del cuerpo de la regla. Este 'separador' se traduce como un **AND** lógico, concatenado cada hecho con un **AND**.

Por ejemplo:

```
1 % X es hermano de Y si existe algún padre Z que sea padre de X y Y
2 hermano(X, Y) :- padre(Z), padrede(Z, X), padrede(Z, Y).
```

Reglas

Disyunciones: Se usa un **punto y coma** para separar los hechos del cuerpo de la regla. Este 'separador' se traduce como un **OR** lógico, concatenado cada hecho con un **OR**.
Por ejemplo:

```
1 % A y B son familiares si A es padre de B o A es hijo de B o A es hermano de B
2 familiarde(A,B) :- padrede(A,B); hijode(A,B); hermanode(A,B).
```

Reglas Recursivas

Prolog permite el uso de la recursividad cuando se están definiendo reglas, esto es útil para definir reglas generales y más flexibles. Por ejemplo si se quiere definir la regla **predecesor_de** se puede realizar de forma iterativa como se muestra a continuación:

```
1 antecesor_de(X,Y) :- padrede(X, Y). % Padre
2 antecesor_de(X,Y) :- padrede(X, Z), padrede(Z, Y). % Abuelo
3 antecesor_de(X,Y) :- padrede(X, Z1), padrede(Z1, Z2), padrede(Z2, Y). %Bisabuelo
```

Reglas Recursivas

Como se puede ver en el anterior ejemplo, para encontrar el antecesor de una persona genera mucho código, y el código generado genera hasta el bisabuelo, pero si se quiere el 10 antecesor?

Para este caso se puede usar la recursividad para de esta manera generar de una forma general el antecesor. A continuación se muestra la forma de realizar esto:

```
1 antecesor_de(X, Y) :- padrede(X, Y). % Paso base
2 antecesor_de(X, Y) :- padrede(X, Z), antecesor_de(Z, Y). % Paso recursivo
```

A continuación se muestra otro ejemplo de Reglas recursivas en el que se calcula el factorial de un numero:

```
1 factorial(0, 1). % paso base.
2 factorial(N, F) :- N>0, N1 is N - 1, factorial(N1, F1), F is N * F1. % Paso recursivo.
```

```
≡ ?- factorial(5, RES)
```


Consultas

Es el mecanismo para extraer conocimiento del programa, donde una consulta está constituida por una o más metas que Prolog debe resolver.

Hecho:

`amigos(pedro, antonio).`

Consulta:

`? amig0s(pedro, antonio).`

Consultas

Para resolver consultas Prolog intenta unificar con algún Hecho o Regla con igual predicado, si es posible, se realiza lo mismo con el Cuerpo de la Regla sustituyendo en cada objetivo también lo que se logró unificar.

Si no se puede resolver un objetivo, se retrocede mediante backtracking con otras alternativas para su resolución, y se resuelve con la siguiente alternativa. Si no existen alternativas disponibles, el objetivo de partida falla. Si se vacía la lista de objetivos, el objetivo queda resuelto.

Consultas

Ejemplo:

Pablo es padre de Juan y de Andrés, ¿Juan es hermano de Andrés?

Hechos:

`padre(pablo, juan).`

`padre(pablo, andres).`

`hermano(A, B) :- padre(C, A), padre(C, B).`

Si consultamos `hermano(juan, andres)`, el proceso es el siguiente: Se unifica con la regla `hermano(A, B)` y obtenemos:

`hermano(juan, andres) :- padre(C, juan), padre(C, andres).`

Ahora se tendrá que hallar C para completar el primer objetivo, para lo cual unificamos con el hecho 1). C = pablo.

Ya que C está definido debemos evaluar si `padre(pablo, andres)` es verdadero.

Se acaban los objetivos y todos fueron verdaderos, entonces, Juan es hermano de Andrés.

Prolog - Estructuras

ESTRUCTURAS • Se pueden utilizar varios datos a la vez:
nacimiento(pedro, fecha(23,ago,1970)).

?- nacimiento(pedro, X).

X=fecha(23, ago, 1970)

yes.

Consultas de todas las personas nacidas en Agosto:

?- nacimiento(X, fecha(Y, ago, Z)).

X=pedro

yes.

Prolog

Uso de operaciones aritméticas en predicados:

`suma(A, B, C):- C is A + B.`

`?- suma(3, 4, 7).`

`yes.`

`?- suma(3, 4, X).`

`X=7`

`yes.`

Prolog

En Prolog, casi no se usan ciclos, en lugar de ellos se aplica recursividad; sin embargo, se pueden implementar.

P. ejemplo Para imprimir los numeros del 1 al 10 se usa ...

```
lista(M, N):- M <N, nl, write(M), NuevoM is M+1, lista(NuevoM, N).
```

En Prolog no hay matrices, en su lugar se usan Listas.

```
[maria, javier, juan]
```

```
[] %lista vacía
```

Prolog - Listas

En prolog, una lista es una representación de un conjunto de elementos.

La notación es la siguiente: [manzana, pera, banana] lista vacia: []

Se pueden utilizar cómo elementos de la lista cualquier tipo de dato de prolog, incluyendo listas: [[a,b,c],[d, e, f]]

También estructuras prolog: [camino(tandil, bsas), camino(mardel, tandil), camino(bsas,junin)]

[vehiculo(ale, [bici, moto, auto]), vehiculo(ariel,[bici, auto, helicoptero])]

Prolog - Listas

Si se tiene la lista [a, b, c, d], la a es la cabeza y la cola es la lista [b, c, d]

Una lista cuya cabeza es A y cola es B se anota como
[A | B]

El predicado primer_elemento(X, [X | _]).

tiene éxito si X es el primer elemento de la lista.

Si la lista no está vacía, primero se imprime la cabeza y luego la cola:
imprimir([A | B]):- write(A), imprimir(B).

Hallar el último elemento de una lista.

% Base

% Recursividad

Hallar el elemento k de una lista.

% Base

% Recursividad

elemento_k(L, K1, Result).

Prolog - Listas

En su forma más básica, una lista se puede ver como un predicado que tiene 2 partes:

`lista(cabeza, cola)`

en prolog: `[Cabeza | Cola]`

Por ejemplo: `[Cabeza | Cola]=[1,2,3] [1] = [1 | []]`

Cabeza=1,

Cola=[2,3]

Ejercicio

1 Definir la relación `primero(?L,?X)` que se verifique si X es el primer elemento de la lista L.

```
| ?- primero([a,b,c],X).  
| X = a
```

Obtener las respuestas a las siguientes preguntas:

```
| ?- primero([X,b,c],a).  
| ?- primero([X,Y],a).  
| ?- primero(X,a).
```