Introducción a Prolog

Existen varias versiones de prolog, a continuación están las instrucciones para instalar SWI-Prolog, que es una versión estable y multiplataforma.

Para instalar en Ubuntu:

1. sudo apt-add-repository ppa:swi-prolog/stable
2. sudo apt-get update
3. sudo apt-get install swi-prolog

Para instalar en otra versión de linux, windows o Mac se pueden seguir las instrucciones del siguiente enlace:

https://www.swi-prolog.org/download/stable

Revisemos algunos aspectos importantes del lenguaje prolog.

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 tipos de términos:

Constantes

- **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').
```

- **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).
```

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 ‘_’. A continuación se muestran algunos ejemplos:

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

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

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

nombre_átomo(termino1, termino2, ..., terminoN).

Donde esos terminos del 1 al N, se les llama argumentos . Además, al nombre del átomo también 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.

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.

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 is 10 + 2
=:=	Igualdad	10 + 2 =:= 5 + 7

=\=	Desigualdad	10 + 2 =\= 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 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)	term ∈ 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.

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

Lo cual sería equivalente a tener en Prolog:

p :- p1, p2, …, pm.

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

Ejemplos:

1. planeta(marte) % Marte es un planeta.
2. hombre('Felipe') % Felipe es un hombre.
3. mato(hombre(_), X) % hombre(_) mató a X
4. come(A,B) :- carnivoro(A), animal(B), masDebil(B, A); herbivoro(A), plantaComestible(B).

Reescrita 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.

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. padre(juan). %Juan es padre - Denota la propiedad que tiene juan y es la de ser padre.
4. 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. 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
4. 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. Por ejemplo:

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

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

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

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 calcúla 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.

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

Hecho:

1. amigos(pedro, antonio).

Consulta:

```
?- amigos(pedro, antonio).
   True
```

Resolución de 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.

Ejemplo 1: Pablo es padre de Juan y de Andrés, ¿Juan es hermano de Andrés?

1. padre(pablo, juan).
2. padre(pablo, andres).
3. hermano(A, B) :- padre(C, A), padre(C, B).

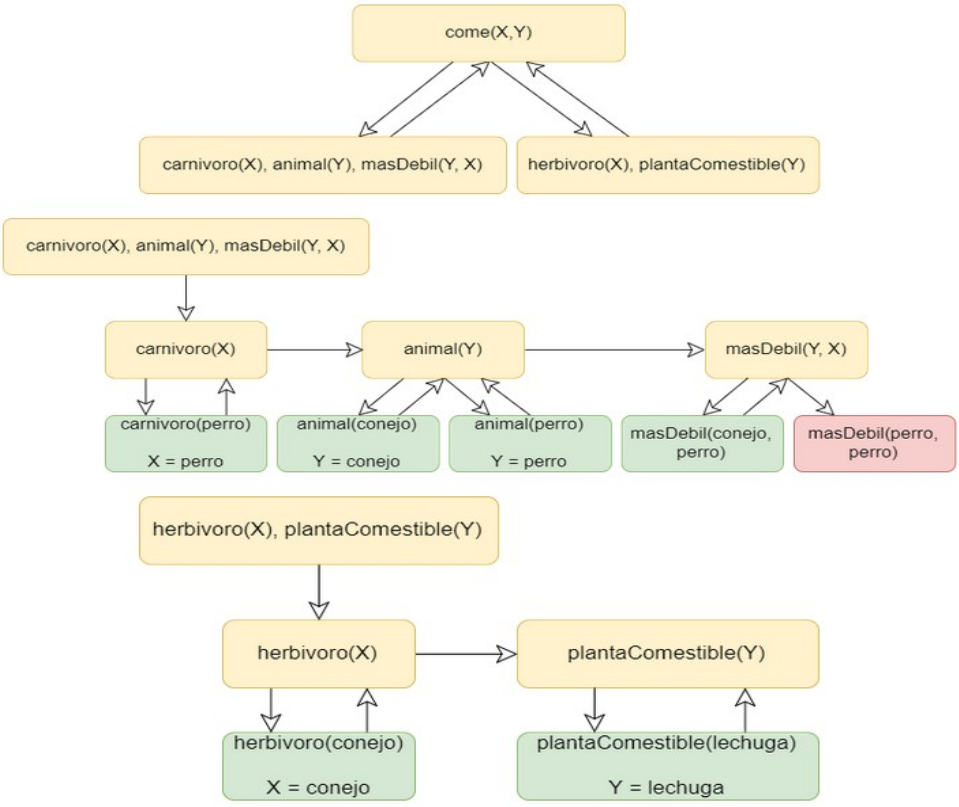
Si consultamos hermano(juan, andres), el proceso es el siguiente:

1. Se unifica con la regla hermano(A, B) y obtenemos:
2. hermano(juan, andres) :- padre(C, juan), padre(C, andres).
3. Ahora se tendrá que hallar C para completar el primer objetivo, para lo cual unificamos con el hecho 1. C = pablo.
4. Ya que C está definido debemos evaluar si padre(pablo, andres) es verdadero.
5. Se acaban los objetivos y todos fueron verdaderos, entonces, Juan es hermano de Andrés.

Ejemplo 2

1. animal(conejo).
2. animal(perro).
3. carnivoro(perro).
4. masDebil(conejo, perro).
5. herbivoro(conejo).
6. plantaComestible(lechuga).
7. come(A,B) :- carnivoro(A), animal(B), masDebil(B, A); herbivoro(A), plantaComestible(B).

Si consultamos come(X, Y), estaríamos preguntando para qué X e Y se cumple que X come a Y el proceso es el siguiente:



En prolog el proceso de llamadas sería el siguiente:

?- trace, come(X, Y).
1. Call:come(_4220, _4224)
2. Call:carnivoro(_4220)
3. Exit:carnivoro(perro)
4. Call:animal(_4224)
5. Exit:animal(conejo)
6. Call:masDebil(conejo, perro)
7. Exit:masDebil(conejo, perro)
8. Exit:come(perro, conejo)
9. X = perro,
10. Y = conejo

La traza de ejecución se obtiene con la instrucción trace y una coma antes de la consulta como esta escrito arriba.

Practica 8

Como el ejemplo 2 anterior que establece una relación según la alimentación, escriba un programa en prolog en el cual se pueda realizar la consulta come(X,Y) para saber que come cada animal. Debe incluir las tres clasificaciones de animales según su alimentación: herbívoro, carnívoro y omnívoro. Incluya también las plantas comestibles para los hervivoros.

En los hechos debe incluir al menos 20 animales para cada clasificación, así como 20 plantas comestibles.

Adjunto a este PDF encontraran un programa en prolog llamado Familia.pl, el cual establece la relación que existe entre personas. Para correr el programa debe estar situado en la carpeta donde se encuentra el archivo Familia.pl y luego ejecutar el siguiente comando:

swipl -s Familia.pl

Entrara al entorno de prolog, donde ya se habrán cargado todas las reglas y hechos del archivo Familia.pl. Observara algo similar a:

$ swipl -s Familia.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?-

En **?-** es donde podrá escribir sus consultas.

Por ejemplo:

?- abuelo(jim,X).
X = cathy ;
X = sharon ;
X = stephanie ;
X = danielle.

- La consulta es ¿De quien es abuelo jim? Como puede observar hay más de una respuesta.
- Cuando ejecute la consulta dando enter, obtendrá la primer respuesta que prolog encuentre, si vuelve a dar enter la consulta termina y solo verá la primer respuesta.
- Cuando ejecute la consulta dando enter, obtendrá la primer respuesta que prolog encuentre, si ahora presiona espacio o introduce un ; obtendrá la siguiente respuesta. Puede seguir presionando espacios o ; hasta que prolog termine de buscar todas las respuestas.

Para salir del entorno de prolog solo debe escribir:

?- halt.