Nombre: Jonathan Atancuri

Prolog

Introducción

Prolog , proveniente del francés PROgrammation en LOGique, es un lenguaje de programación lógico e interpretado usado habitualmente en el campo de la Inteligencia artificial.

Inicialmente se trataba de un lenguaje interpretado hasta que, una década más tarde, se creó un compilador capaz de traducir Prolog en un conjunto de instrucciones de una máquina abstracta denominada WAM ("Warren Abstract Machine"), lo que lo convirtió desde entonces en un lenguaje semi-interpretado.

Prolog nació de un desafío crear un lenguaje de muy alto nivel, aun cuando fuera ineficiente para los informáticos de la época.

Definición Prolog

Un programa Prolog se compone de un conjunto de hechos (afirmaciones simples) y de reglas (que sirven para afirmar la veracidad de un hecho en base a otros).

Está basado en los siguientes mecanismos básicos: unificación, estructuras de datos basadas en árboles y backtracking automático.

Características

- Basado en lógica y programación declarativa.
- No se especifica cómo debe hacerse, sino qué debe lograrse.
- Una característica importante en ProLog y que lo diferencia de otros lenguajes de programación, es que una variable sólo puede tener un valor mientras se cumple el objetivo.
- El programador se concentra más en el conocimiento que en los algoritmos.

  -¿Qué es conocido? (hechos, reglas)

  -¿Qué preguntar? (Cómo resolverlo)

Funciones

Los programas en ProLog responden preguntas sobre el tema del cual tienen conocimiento. ProLog es un lenguaje de programación especialmente indicado para modelar problemas que impliquen objetos y las relaciones entre ellos.

La programación en ProLog consiste en:

- Declarar algunos HECHOS sobre los objetos y sus relaciones.
- Definir algunas REGLAS sobre los objetos y sus relaciones, y
- Hacer PREGUNTAS sobre los objetos y sus relaciones.

Uso de la Herramienta SWI-Prolog y el IDE

Base de conocimiento

Para responder a las preguntas o consultas formuladas por el programador, Prolog consulta una base de conocimiento. Ésta base conocimiento representa el programa como tal, programa que

se compone unicamente de clausulas, que con el uso de la lógica, me expresan el conomiento deseado por el programa.

La base de conociento o el programa se guarda en un archivo con la extención '.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').
```

Expresiones

- **Operadores Aritmeticos**

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

- **Operadores Relacionales**

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**

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]).

Clausulas

Las cláusulas en Prolog están basadas en cláusulas de Horn.

P1 ^ P2 ^ … ^ Pm => 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:

"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."

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

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:

```
color(azul).% azul es color - Denota la propiedad del azul de ser un color
color(verde). % verde es color

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

```
padrede('juan', 'maria'). % Juan es padre de maria - Este hecho expresa una relacion de padre-hijo
padrede('pablo', 'juan'). % Pablo es padre de juan

edad(juan, 30). % Juan tiene la edad de 30 años - Este hecho está relacionando a juan con un edad de 30 a
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:

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.

```
%A es hijo de B si B es padre de A
hijode(A,B) :- padrede(B,A).

% A es abuelo de B si A es padre de C y C es padre B
abuelode(A,B) :- padrede(A,C), padrede(C,B).
```

Las reglas se dividen en :

- **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.
  Ejemplo:
  ```
  % X es hermano de Y si existe algún padre Z que sea padre de X y Y
  hermano(X, Y) :- padre(Z), padrede(Z, X), padrede(Z, 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.
  ```
  % A y B son familiares si A es padre de B o A es hijo de B o A es hermano de B
  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:

```
antecesor_de(X,Y) :- padrede(X, Y). % Padre
antecesor_de(X,Y) :- padrede(X, Z), padrede(Z, Y). % Abuelo
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:

```
antecesor_de(X, Y) :- padrede(X, Y).                    % Paso base
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:

```
factorial(0, 1). % paso base.
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:

```
amigos(pedro, antonio).
```

Consulta

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

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.

Ejemplo:

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).
```

Listas

Las listas son estructuras que contiene una secuencia ordenada de cualquier tipo de términos. Está formada recursivamente por una cabeza, que es el primer elemento de la lista y una cola, que es una lista del resto de elementos.

Ejemplo:

Hallar el último elemento de una lista.

```
1 ultimo([Result], Result).                    % Base
2 ultimo([_|L], Result) :- ultimo(L, Result).  % Recursividad
```

```
?- ultimo([a, [b,c], 2], Ultimo).
```

Ultimo = 2

Next | 10 | 100 | 1,000 | Stop

Backtracking

Prolog siempre está consultando “la base del conocimiento”, para verificar que hechos son verdaderos y que nos permitirá la construcción de las posibles reglas. Para aquello en problemas de recursión, prolog se devuelve hasta que encuentra que un hecho base es verdadero y de ahí construye la respuesta.

Aplicaciones

A continuación se listan algunas aplicaciones de Prolog:

- 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 par 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.
- Academia.
- Investigación en inteligencia artificial

Desventajas	Ventajas
Curva de aprendizaje larga	El código tiende a ser mucho mas corto, luego es fácil de modificar
Se debe establecer muy bien los hechos o la representación del conocimiento, porque pueden haber soluciones erroneas	Facilidad para programar

Forma de pensar en diferente a como estamos acostumbrados	

Ejemplo 1: Recetar una medicina

Vamos a ver cada parte del sistema y que funcion cumple.

La primera parte, la regla evaluar es la cual inicia el proceso, está a su vez llama a hipotesis, para luego de evaluarla escribir en consola la solución que considera correcta según los datos que tiene disponibles.

```
evaluar:-
hipotesis(Enfermedad),
write('Creo que el paciente tiene '),
write(Enfermedad),
nl,
write('Tener cuidado!'),
deshacer.
```

En la regla "hipotesis" se enumeran las enfermedades las cuales puede diagnosticar el sistema experto

```
/*Hipotesis que deberia ser probada*/
hipotesis(resfriado) :- resfriado.
hipotesis(gripe) :- gripe.
hipotesis(tifoidea) :- tifoidea.
hipotesis(sarampion) :- sarampion.
hipotesis(malaria) :- malaria.
hipotesis(desconocido). /* sin diagnostico*/
```

Cada enfermedad tiene una regla la cual liga los síntomas con la respectiva enfermedad, además de mostrar los respectivos medicamentos recomendados.

```
/*Reglas de identificacion*/
resfriado :-
verificar(dolor_cabeza),
verificar(nariz_moqueda),
verificar(estornudo),
verificar(dolor_garganta),
write('Consejos y Sugerencias:'),
```

```
nl,
write('1: Tylenol'),
nl,
write('2: Panadol'),
nl,
write('3: Aerosol nasal'),
nl,
write('Por favor use ropa de abrigo porque'),
nl.
```

Cada regla de enfermedad usa "verificar" para reconocer si se sabe si el usuario tiene o no un síntoma en específico, el hecho "si(sintoma)" no se encuentra en la base de conocimiento del sistema entonces se hace el llamado a la regla "preguntar(síntoma)

```
/*Como se verifica */
verificar(S) :-
(si(S)
->
true ;
(no(S)
->
fail ;
preguntar(S))).
```

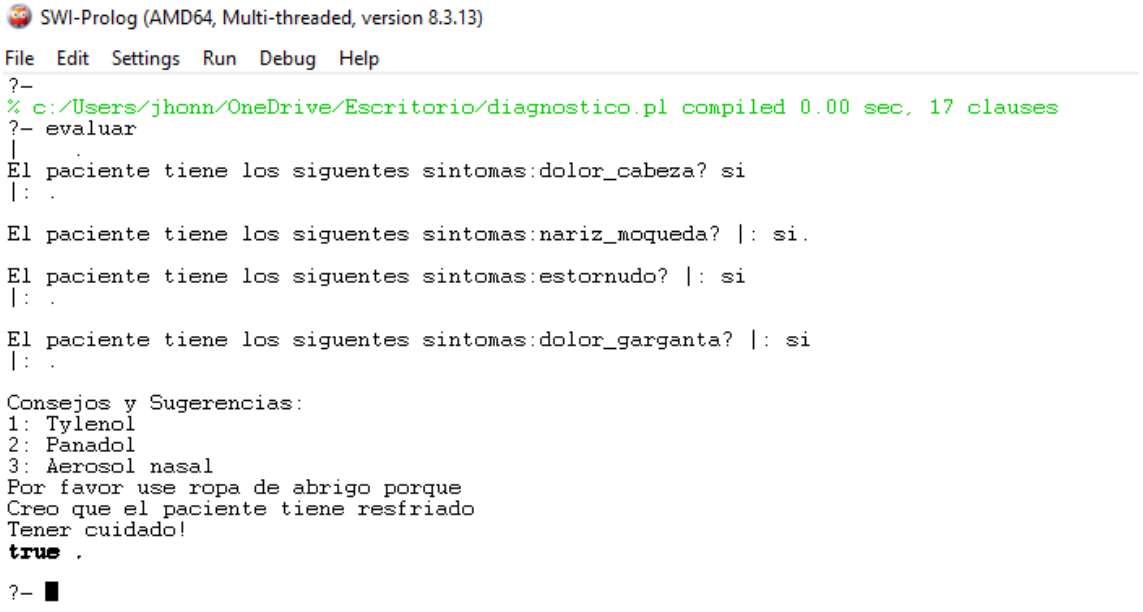
La regla auxiliar "preguntar" sirve para recibir los datos del usuario, esta hace uso de la regla assert que trae prolog por defecto para poder añadir nuevos hechos a la base de conocimiento del sistema.

```
/* Como se hacen las preguntas */
preguntar(Pregunta) :-
write('El paciente tiene los siguentes sintomas:'),
write(Pregunta),
write('? '),
read(Respuesta),
nl,
( (Respuesta == si)
```

->

assert(si(Pregunta)) ;

assert(no(Pregunta)), fail).



Ejemplo 2

Animales

hypothesis(cheetah):- cheetah, !.

hypothesis(tigre):- tigre, !.

hypothesis(jirafa):- jirafa, !.

hypothesis(zebra):- zebra, !.

hypothesis(animal_desconocido).

Base de echos identifica las reglas

cheetah:-

verify(mamifero),

verify(carnivoro),

verify(color_ambar),

verify(manchas_oscuras).

tigre :-

verify(mamifero),

verify(carnivoro),

```
    verify(color_ambar),
    verify(rayas_negras).
jirafa :-
    verify(mamifero),
    verify(pezunias),
    verify(cuello_largo),
    verify(manchas_oscuras).
zebra :-
    verify(mamifero),
    verify(rayas_negras),
    verify(pezunias).
```

Preguntas

```
ask(Question) :-
  write('El animal tiene la siguiente caracteristica?: '),
  write(Question),
  write('? '),
  read(Response),
  nl,
  ( (Response == yes ; Response == y)
   ->
    assert(yes(Question)) ;
    assert(no(Question)), fail).
```

Conclusiones:

Determinamos que es un herramienta bastante util , para poder realizar sistemas expertos , usado en casi todas las areas de la medicina , tecnologia ,inteligencia artificial,etc.

Esta herramienta nos facilita la solucion a muchos problemas que se tiene dia a dia partiendo de su base de conocimientos que seria mas o menos la base de datos que tendria el prolog en memoria , en donde nos devolveria un resultado con hechos y relaciones y ademas de implementar las reglas para poder preguntar al robot y que nos devuelva una respuesta.