Elementos Básicos de Programación en PROLOG

EDWIN INSUASTY PORTILLA

Prolog, lenguaje de programación cuyo nombre nació de la expresión **PROgramming in LOGic**, fue inventado en Francia al rededor de 1970 por investigadores de la Universidad de Marsella para ser usado en experimentación de Inteligencia Artificial. En la actualidad a conquistado el liderazgo por encima del lenguaje americano LISP dedicado también a investigaciones similares.

Existen en el mundo diversidad de lenguajes de programación, algunos de propósito específico y otros de propósito general como por ejemplo COBOL para desarrollo de aplicaciones comerciales y C para cualquier tipo de aplicación, incluyendo la Inteligencia Artificial. Prolog está dedicado más al tipo de problemas lógicos, donde la toma de decisiones y el orden son importantísimos. La ventaja que PROLOG tiene sobre LISP, es su escritura simple como en los ejemplos se notará.

Dos aspectos importantes que se trabajan en Inteligencia Artificial son los Sistemas Expertos y el Procesamiento del Lenguaje Natural (lenguaje humano). Los Sistemas expertos son paquetes de programas cuya función es la de emular a un experto humano, tal como un médico quien observando unos síntomas y con la información de unos exámenes, puede diagnosticar una enfermedad. Es obvio que el sistema experto debe estar provisto de una base de datos y código que permita en ella analizar las preguntas que se le

formulen con el fin de encontrar las respuestas correspondientes, esto se conoce como **Motor de Inferencia** (procesamiento por razonamiento lógico de la información). El procesamiento del Lenguaje Natural consiste en el desarrollo de software que haga capaz a la máquina entender el lenguaje del hombre por ejemplo "haga una copia del archivo command.com para guardarlo en el directorio notas y póngale como nuevo nombre ensayo.sss" y que con esta orden, la máquina ejecute las acciones que en el momento actual las daríamos así:

```
c:\> copy  command.com  \notas
c:\> ren  \notas\command.com  ensayo.sss
```

Ahora empecemos con los conceptos iniciales de programación en Prolog. Se deben construir las llamadas cláusulas; en el ejemplo siguiente se observará la manera de traducir hechos de la vida normal a la sintaxis de este lenguaje.

ORACION EN LENGUAJE NATURAL	SINTAXIS LOGICA
carlos es un deportista	deportista(carlos)
a lucy le gusta un mazda si el mazda es bonito	le_gusta(lucy,mazda) if es_bonito(mazda)
susy es prima de luis	prima(susy, luis)

Un objeto es algo que se puede definir en términos computacionales. Hay que definir objetos y relaciones como por ejemplo:

Objetos : luis, diana, dora, josé, paola
Relaciones: hijo, padre, mujer, hombre, esposa

Las relaciones aplicadas a los objetos forman los Hechos : hijo(luis,josé) esposa(diana, josé) mujer(paola). Claro que si usted le define un hecho simple como hombre(dora), Prolog lo tomará como cierto pues allí no tiene otra forma de razonar sino aceptarlo plenamente.

En el siguiente ejemplo se tratará lo correspondiente a la escritura de predicados, hechos y reglas.

```
predicates
  le_gusta(symbol,symbol)   Nota : aquí se define qué tipo de datos se aceptan

clauses
  le_gusta(luisa, tenis).
  le_gusta(john, futbol).
  le_gusta(lola, beisbol).
  le_gusta(paty, nadar).
  le_gusta(carlos, tenis).
  le_gusta(ruth, Actividad) if le_gusta(luisa, Actividad).
```

Nótese que la mayoría de palabras están escritas en minúsculas. La palabra **Actividad** está con mayúscula para identificar que es una variable.

Al correr el programa podemos dar como objetivo los siguientes y obtener a continuación las respuestas que se señalan:

```
Goal: le_gusta(john, tenis)
No (respuesta del programa)

Goal: le_gusta(lola, beisbol)
Yes  (respuesta del programa)

Goal: le_gusta( _ , tenis)
Yes
```

La consulta anterior significaría: Hay alguien a quien le guste tenis?

La regla: le_gusta(ruth, Actividad) if le_gusta(luisa, Actividad). combina el hecho: le_gusta(luisa, tenis) para decidir que: le_gusta(ruth, tenis). Por lo tanto se obtendrán las siguientes respuestas a las consultas:

```
Goal: le_gusta(ruth, beisbol)
No

Goal: le_gusta(ruth, tenis)
Yes
```

El siguiente ejemplo introduce en una regla, el uso de la conjunción **and** que puede ser remplazada por una coma (,) y el condicional **si** por **:-**

Examine las siguientes consultas y sus resultados:

```
predicates
  le_gusta(symbol,symbol).

clauses
  le_gusta(carlos, mazda).
  le_gusta(pepe, r4).
  le_gusta(judy, pizza).
  tiene_dinero(carlos)
  puede_comprar(Quien, Cosa) :- le_gusta(Quien, Cosa) , tiene_dinero(Quien)

Goal: puede_comprar(carlos, r4)
No

Goal: le_gusta(carlos, mazda)
Yes

Goal: puede_comprar(carlos, mazda)
Yes

Goal: puede_comprar(Alguien, mazda)
Alguien = carlos
1 solution

Goal: le_gusta(judy, Que)
Que=pizza
1 solution

Goal : puede_comprar(judy, pizza)
No
```

El ejemplo siguiente, se hará uso de una variable anónima para realizar búsquedas de soluciones.

```
predicates
  le_gusta(symbol,symbol)

clauses
  le_gusta(elena, leer).
  le_gusta(john, computadores).
  le_gusta(john, pinpon).
  le_gusta(leonardo, pinpon).
  le_gusta(eric, nadar).
  le_gusta(eric, leer).

Goal: le_gusta(Quien, leer)
Quien = elena
Quien = eric
2 solutions
```

Si se quiere saber si la consulta que se hará tiene o no por lo menos una solución se usa para ello la variable anónima "_"

```
Goal: le_gusta( _ , leer)
Yes
```

Con el siguiente ejemplo, se observará otras formas de usar la variable anónima.

```
predicates
  hombre(symbol)
  mujer(symbol)
  familiar(symbol, symbol)

clauses
  hombre(bill).
  hombre(joe).
  mujer(sue).
  mujer(tatiana).
  familiar(bill, joe).
  familiar(sue, joe).
  familiar(joe, tatiana).
```

En caso de querer saber quienes son parientes de alguien, se preguntaria:

```
Goal: familiar(Quienes, _ )
Quienes = bill
Quienes = sue
Quienes = joe
3 solutions
```

COMPOSICIÓN DE GOALS CON CONJUNCIONES Y DISYUNCIONES

A continuación se usarán otros tipos de argumentos admitidos por turbo prolog

```
predicates
  carro(symbol,real,integer,symbol,integer)
  camion(symbol,real,integer,symbol,integer)

clauses
  carro(chrysler, 130000, 3, rojo, 12000).
  carro(ford, 90000, 4, gris, 25000).
  carro(datsun, 8000, 1, rojo, 30000).
  camion(ford, 80000, 6, azul, 8000).
```

```
camion(datsun, 50000, 5, naranja, 20000).
camion(toyota, 25000, 2, negro, 25000).
```

Los argumentos de los predicados carro y camion son en su orden:

carro(fábrica, kilometraje, años de uso, color, precio en miles)

Para preguntar si hay carros que cuesten 25000 se consultaría:

```
Goal: carro( Marca, Kilometros, Años, Color, 25000)
Marca = ford
Kilometros = 90000
Años = 4
Color = gris
1 solution
```

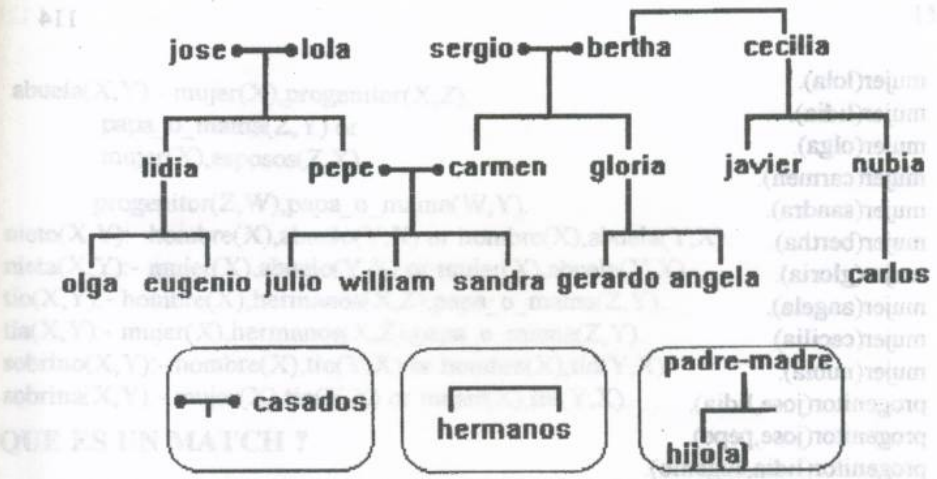
Se puede usar la conjunción para generar búsquedas como en el siguiente ejemplo:

```
Goal: carro(A, B, C, D, Precio ) and Precio < 26000
A = chrysler                A = ford
B = 130000                  B = 90000
C = 3                       C = 4
D =rojo                     D = gris
Precio = 12000                  Precio = 25000
2 solutions
```

Si deseo preguntar: Hay un carro con costo menor de 20000 o un camión con costo menor de 15000 ? podré usar la disyunción **or** así:

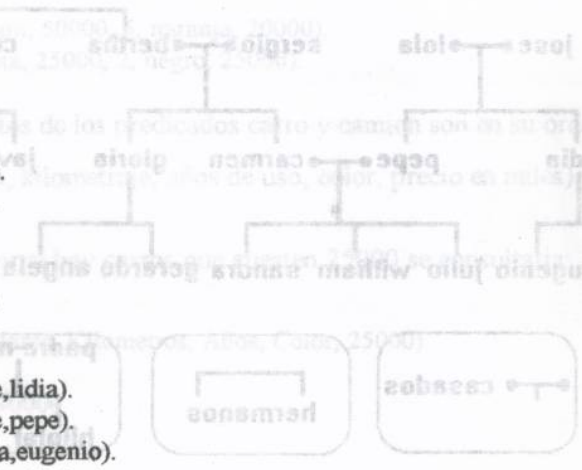
```
Goal: carro(A, B, C, D, Precio) and Precio < 20000 or camion(A, B, C, D, Precio)
and Precio < 15000
A = chrysler                A = ford
B = 130000                  B = 80000
C = 3                       C = 6
D =rojo                     D = azul
Precio = 12000              Precio = 8000
2 solutions
```

El siguiente ejemplo muestra la manera de consultar parentescos en el siguiente árbol.



El programa es el siguiente:

```
predicates
        hombre(symbol)
        mujer(symbol)
        progenitor(symbol,symbol)
        esposos(symbol,symbol)
        hermanos(symbol,symbol)
        hijo(symbol,symbol)
        hija(symbol,symbol)
        prima(symbol,symbol)
        primo(symbol,symbol)
        papa(symbol,symbol)
        mama_soltera(symbol,symbol)
        mama(symbol,symbol)
        papa_o_mama(symbol,symbol)
        abuelo(symbol,symbol)
        abuela(symbol,symbol)
        nieto(symbol,symbol)
        nieta(symbol,symbol)
        tio(symbol,symbol)
        tia(symbol,symbol)
        sobrino(symbol,symbol)
        sobrina(symbol,symbol)
clauses
        hombre(jose).
        hombre(pepe).
        hombre(eugenio).
        hombre(julio).
        hombre(william).
        hombre(sergio).
        hombre(gerardo).
        hombre(javier).
        hombre(carlos).
```



```
mujer(lola).
mujer(lidia).
mujer(olga).
mujer(carmen).
mujer(sandra).
mujer(bertha).
mujer(gloria).
mujer(angela).
mujer(cecilia).
mujer(nubia).
progenitor(jose,lidia).
progenitor(jose,pepe).
progenitor(lidia,eugenio).
progenitor(lidia,olga).
progenitor(pepe,julio).
progenitor(pepe,sandra).
progenitor(pepe,william).
progenitor(sergio,carmen).
progenitor(sergio,gloria).
progenitor(gloria,gerardo).
progenitor(gloria,angela).
progenitor(cecilia,javier).
progenitor(cecilia,nubia).
progenitor(nubia,carlos).
esposos(jose,lola).
esposos(pepe,carmen).
esposos(sergio,bertha).
hermanos(bertha,cecilia).
hermanos(cecilia,bertha).
hermanos(X,Y):- progenitor(Alguien,X),
      progenitor(Alguien,Y),X<>Y.
hijo(X,Y):- hombre(X), progenitor(Y,X) or
     hombre(X), progenitor(Z,X),
     esposos(Z,Y).
hija(X,Y):- mujer(X), progenitor(Y,X) or
     mujer(X), progenitor(Z,X),
     esposos(Z,Y).
papa(X,Y):- hombre(X), progenitor(X,Y).
mama_soltera(X,Y):- mujer(X),progenitor(X,Y).
mama(X,Y):- mujer(X), progenitor(Z,Y),esposos(Z,X)
     or mama_soltera(X,Y).
papa_o_mama(X,Y):- papa(X,Y) or mama(X,Y).
primo(X,Y):- hombre(X), hijo(X,B),
      hermanos(B,A), papa_o_mama(A,Y).
prima(X,Y):- mujer(X), hija(X,B),
      hermanos(B,A),papa_o_mama(A,Y).
abuelo(X,Y):- hombre(X),progenitor(X,Z),
      papa_o_mama(Z,Y).
```

```
abuela(X,Y):- mujer(X),progenitor(X,Z),
        papa_o_mama(Z,Y) or
        mujer(X),esposos(Z,X),
        progenitor(Z,W),papa_o_mama(W,Y).
nieto(X,Y):- hombre(X),abuelo(Y,X) or hombre(X),abuela(Y,X).
nieta(X,Y):- mujer(X),abuelo(Y,X) or mujer(X),abuela(Y,X).
tio(X,Y):- hombre(X),hermanos(X,Z),papa_o_mama(Z,Y).
tia(X,Y):- mujer(X),hermanos(X,Z),papa_o_mama(Z,Y).
sobrino(X,Y):- hombre(X),tio(Y,X) or hombre(X),tia(Y,X).
sobrina(X,Y):- mujer(X),tio(Y,X) or mujer(X),tia(Y,X).
```

QUE ES UN MATCH ?

Los match involucran una o más variables libres. Suponiendo que se presenta el siguiente Goal: pariente(lola, X), Prolog empieza a buscar en la sección clauses de arriba a abajo y de izquierda a derecha hasta encontrar un predicado idéntico al del goal, luego verifica el número de argumentos de lo encontrado ya que pueden existir predicados de igual escritura pero con diferente número de parámetros. Si las condiciones no se satisfacen, Prolog deja la primera cláusula y toma la segunda, así hasta encontrar un predicado idéntico y con el mismo número de argumentos, como el siguiente hecho: pariente(lola, josé). En este caso Prolog vincula o "matchea" la variable X con el valor josé, así el goal coincide con el predicado hallado, anuncia en pantalla la variable y el valor para proceder a liberar la variable X del valor José con el fin de tomar la cláusula de la derecha o la inmediatamente inferior buscando nueva solución. Dos o más variables libres pueden vincularse.

ARGUMENTOS DE LOS PREDICADOS

En la sección predicates hemos declarado los predicados que se utilizarán para definir los hechos y las reglas con los tipos de argumentos que utilizan, como por ejemplo: carro(symbol, symbol, integer, real). Es posible definir los argumentos de los predicados de manera más clara diciendo por ejemplo: carro(marca, color, kilometraje, precio), solo que se debe declarar estos tipos de argumentos dentro de los tipos o dominios standar de Prolog así:

```
domains
        marca = symbol
        color = symbol
```

```
    kilometraje = integer
    precio = real

predicates
    carro( marca, color, kilometraje, precio )
```

Es posible declarar en la sección domains de la siguiente manera:

```
    marca, color = symbol
    edad, peso, estatura_cms = integer
```

Detalle el siguiente ejemplo:

```
domains
    numero,suma, producto = integer

predicates
    sum(numero,numero,suma)
    pro(numero,numero,producto)

clauses
    sum(X,Y,Suma):- Suma=X+Y.
    pro(X,Y,Producto):- Producto=X*Y.
```

Utilice en goals, números enteros grandes y observe las respuestas. Note que con enteros grandes se produce un error y en algunos casos produce resultados negativos como en el siguiente caso:

```
sum(16384, 16384, S)
S = -32768
1 solution
```

Esto se debe a que Prolog maneja enteros desde -32768 al 32767 de igual manera, se tienen rangos para reales.

A continuación se describen los tipos de argumentos admitidos por Prolog:

DOMINIO	DESCRIPCION
char	Son caracteres que se escriben entre dos comillas simples. Por ejemplo: 'a' '$'
integer	Son números enteros comprendidos entre -32768 al 32767
real	Son números que pueden deben llevar los signos + o - adelante, luego una serie de dígitos posiblemente con parte decimal que se

	separa mediante un punto y opcionalmente una parte exponencial iniciada con una letra e seguida de un signo (opcional) y tres dígitos máximo de exponente. Por ejemplo: +634653 - 6454.848455 534e+235 98.242323e-54 45e567 El significado de 34.8e-567 es : 34.8 * 10 ^ -567 El rango permitido en los reales es : 1e-307 al 1e308. Los enteros se convierten en reales cuando sea necesario.
string	Son secuencias de caracteres escritos entre comillas dobles como por ejemplo: "San Juan de Pasto" que pueden tener hasta 255 caracteres.
symbol	Puede presentarse en dos formas: una secuencia de caracteres que comienzan por una letra minúscula, separando según convenga mediante el underscore o la "la raya subrayadora _ " como en: edad_en_años, número_telefónico, luis_perez. La otra forma acepta una cadena de caracteres escritos entre comillas dobles igual que las string; a demás es posible intercambiar strings y symbols pero su almacenamiento se hace de manera diferente, mientras los symbols se almacenan en tablas y al haber repetidos, Prolog busca la manera de compactar esta información y la confrontación en un match se hace mas rápida; en cambio un match sobre una string se hace caracter por caracter.

ARIDAD MULTIPLE

La Aridad es el número de argumentos que un predicado puede tomar, luego se pueden tener dos predicados con el mismo nombre pero diferente aridad. Vease el siguiente ejemplo:

```
domains
  persona = symbol

predicates
  padre(persona)          /* Esta persona es un padre */
  padre(persona, persona)  /* La primera persona es el padre de la otra persona
*/

clauses
  padre(Hombre) :-
    padre(Hombre,_).
  padre(adam, seth).
  padre(abraham, isaac).
```

SINTAXIS DE LAS REGLAS

Las reglas se componen de tres partes: la cabecera, el cuerpo y el símbolo :- (if) que separa los dos. Esquematizando: CABECERA :- <Subgoal>, <Subgoal>, <Subgoal>, ... , <Subgoal>.

Cada subgoal es un llamado a otro predicado de prolog. Se procesa un subgoal hasta probar su veracidad, para luego continuar con el siguiente. Si un subgoal falla, Prolog regresa para buscar otra alternativa para él en subgoals cercanos, toma nuevos valores en las variables y continua el análisis. Este procedimiento se llama **Backtracking** que se ilustrará mejor más adelante.

Hay diferencia entre el **IF** de Prolog y el If de otros lenguajes como Pascal. Si consideramos la forma de escritura CABEZA - CUERPO, Para Pascal la condición CABEZA contenida en el IF debe ser verdadera antes para poder ser verdadero el CUERPO, es decir:

"SI CABEZA es Verdadero, ENTONCES CUERPO es verdadero (o ENTONCES hacer CUERPO)" es decir es un *condicional if/then.*

Para Prolog el IF se maneja de manera diferente en las reglas. "Se concluye que la CABEZA es verdadera después SI el CUERPO de la regla sucede (o SI CUERPO puede ser hecho)". De esta manera una regla en Prolog es un *condicional then/if.*

CONVERSION AUTOMATICA DE TIPOS.

Cuando se hace un match, Prolog trata de convertir los tipos de datos de las dos variables y es posible en las siguientes circunstancias:

- Entre String y Symbol
- Entre Integer, Char y Real, convirtiendo el Char a valor numérico, su valor ASCII