**Prolog:** Lenguaje de programación,proveniente del francés Programation et Logique, concebido para escribir programas de Inteligencia Artificial. Creado en 1972, las tareas se expresan describiendo los objetos que se necesitan y las relaciones lógicas entre ellos.

# La programación lógica en PROLOG.

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

Una claúsula de Horn puede ser ó bien una conjunción de hechos positivos ó 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*** (CWA); solo es cierto lo que aparece en la base de conocimiento ó bien se deriva de esta.

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

- En PROLOG todas las variables están implícitamente cuantificadas universalmente.
- En PROLOG existe un símbolo explicito para la conjunción "y" (","), pero no existe uno para la disyunción "o", que se expresa como una lista de sentencias alternativas.
- En PROLOG, las implicaciones p --> q se escriben alrevés q :- p, ya que el interprete siempre trabaja hacia atrás sobre un objetivo, como se vera más adelante.

[ Indice ]

**2. El lenguaje de programación PROLOG.**

PROLOG puede encontrarse en múltiples versiones diferentes. La más popular es la definida por Clocksin y Mellish, y es la que se tratará aquí. Afortunadamente, al ser tan sencilla la sintaxis del PROLOG, las variaciones entre distintas implementaciones son mínimas. Los elementos fundamentales de un programa PROLOG son los siguientes:

**2.1 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 orden de los objetos dentro de la relación es arbitrario, pero debemos ser coherentes a lo largo de la base de hechos.

**2.2. Variables.**

Representan objetos que el mismo PROLOG determinar . Una variable puede estar ***instanciada*** ó ***no instanciada***. Estar 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.

**2.3. Reglas.**

Las reglas se utilizan en PROLOG para significar que un hecho depende de uno ó mas hechos. Son la representación de las implicaciones lógicas del tipo p ---> q (p implica q).

- Una regla consiste en una cabeza y un cuerpo, unidos por el signo ":-".
- La cabeza está formada por un único hecho.
- El cuerpo puede ser uno ó mas 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 está, 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 está, lloviendo (pueden haber regado las calles). La representación **correcta** en PROLOG, sería:

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

Adviértase que la regla está "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 está 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 instanciacion en (2). Sin embargo todas las X de **una misma regla** sí que se instanciarán con el mismo valor.

**2.4. Operadores.**

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

| | |
|---|---|
| X = Y | igual |
| X \= Y | distinto |
| X < Y | menor |
| X > Y | mayor |
| X =< Y | menor ó igual |
| X >= Y | mayor ó 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:

```
densidad(X,Y) :- poblacion(X,P), area(X,A), Y is P/A.
```

[ Indice ]

**3. Programación básica en PROLOG.**

**3.1 Un ejemplo sencillo.**

Con los datos que conocemos, ya podemos construir un programa en PROLOG. Necesitaremos un editor de textos para escribir los hechos y reglas que lo componen. Un ejemplo sencillo de programa PROLOG es el

siguiente:

```
quiere_a(maria,enrique).
quiere_a(juan,jorge).
quiere_a(maria,susana).
quiere_a(maria,ana).
quiere_a(susana,pablo).
quiere_a(ana,jorge).
varon(juan).
varon(pablo).
varon(jorge).
varon(enrique).
hembra(maria).
hembra(susana).
hembra(ana).
teme_a(susana,pablo).
teme_a(jorge,enrique).
teme_a(maria,pablo).
/* Esta linea es un comentario */
quiere_pero_teme_a(X,Y) :- quiere_a(X,Y), teme_a(X,Y).
querido_por(X,Y) :- quiere_a(Y,X).
puede_casarse_con(X,Y) :- quiere_a(X,Y), varon(X), hembra(Y).
puede_casarse_con(X,Y) :- quiere_a(X,Y), hembra(X), varon(Y).
```

Una vez creado, lo salvaremos para su posterior consulta desde el interprete PROLOG. Un programa PROLOG tiene como extensión por defecto '.PRO'. Le daremos el nombre 'relacion.pro'.