

Programación IV

Programación lógica.

Programación lógica

La programación funcional se basa en el concepto de función (que no es más que una evolución de los predicados), de corte más matemático. La programación lógica gira en torno al concepto de **predicado, o relación entre elementos**.

La lógica matemática es la manera más sencilla, para el intelecto humano, de expresar formalmente problemas complejos y de resolverlos mediante la aplicación de reglas, hipótesis y teoremas. De ahí que el concepto de "programación lógica" resulte atractivo en diversos campos donde la programación tradicional es un fracaso.

PROLOG

- **Lógica de primer orden**
- **Declarativo**
- **Tipado Dinámico**

La sintaxis del lenguaje consiste en lo siguiente:

- Declarar hechos sobre objetos y sus relaciones
- Hacer preguntas sobre objetos y sus relaciones
- Definir reglas sobre objetos y sus relaciones

Los hechos PROLOG

En consola:

```
assert(progenitor(laura, damian)).
```

En un archivo:

```
progenitor(laura, damian).
```

“progenitor” es el nombre de la relación o nombre de predicado y “laura” y “damian” son los argumentos. Los hechos acaban siempre con punto.

Las preguntas PROLOG

Sobre un conjunto de hechos se pueden realizar una serie de preguntas en la consola. Por ejemplo:

?- progenitor(laura, damian).

true.

?- progenitor(juan, damian).

false.

PROLOG busca automáticamente en la base de datos si existe un hecho que se puede unificar.

Las preguntas PROLOG

También se puede consultar por medio de variables. Por ejemplo:

?- progenitor(jose,X).

X = ana ;

X = patricia.

Debemos consultar más de un resultado por medio de la tecla “Tab”

Las reglas en PROLOG

Existe en PROLOG la posibilidad de definir la relación “abuelo(X,Y)” o la relación “tio(X,Y)” como reglas, además de poderlo hacer como hechos o como conjunción de objetivos, en consola :

```
[user].  
abuelo(X,Y):-  
    progenitor(X,Z), progenitor(Z,Y).  
tio(X,Y):-  
    progenitor(Z,Y), progenitor(V,Z), progenitor(V,X).  
<EOF>
```

Las reglas en PROLOG

En archivos :

```
predecesor(X,Y):-progenitor(X,Y).
```

```
predecesor(X,Y):-progenitor(X,Z), predecesor(Z,Y).
```

La definición de varias reglas con el mismo nombre de relación equivale en PROLOG a la “O” lógica o disyunción.

Términos. Constantes

- **Números.** Este tipo de constantes se utilizan para representar tanto números enteros como números reales y poder realizar con ellos operaciones aritméticas.
- **Átomos.** Los átomos se utilizan para dar nombre a objetos específicos, es decir, representan individuos concretos. Existen tres clases principales de átomos:
 - ~ Cadenas formadas por letras, dígitos y el símbolo de subrayado, que deben empezar necesariamente por una letra minúscula.
 - ~ Cualquier cadena de caracteres encerrada entre comillas simples.

Términos. Variables

Las variables en Prolog se representan mediante cadenas formadas por letras, dígitos y el símbolos de subrayado, pero deben necesariamente empezar por una letra mayúscula o por un símbolo de subrayado.

Ejemplos: X, Resultado_1, Entrada, _total3, _3bis

Las variables que empiezan con un símbolo de subrayado, se denominan variables anónimas, y se usan cuando se necesita trabajar con variables cuyos posibles valores no interesan.

Términos compuestos. Listas

Uno de los términos compuestos mas importantes y útiles que ofrece Prolog son las listas, secuencias ordenadas de cero o mas elementos, donde los elementos pueden ser cualquier tipo de termino.

■ la lista vacía se representa mediante el átomo []

■ toda lista no vacía tiene una cabeza y un resto (que será una lista), y se representa mediante un termino compuesto de paridad 2, cuyos argumentos son, respectivamente, la cabeza y el resto de la lista.

La notación consiste en enumerar entre corchetes todos los elementos de la lista, separados por comas.

Ejemplos, [a, []] o [a] y [a, b, c, []] o [a,b,c]

Prolog también dispone de otra notación para las listas, que consiste en representar la lista con cabeza X y resto Y mediante el termino [X|Y]. Esta ultima notación es fundamental para poder separar la cabeza del resto de una lista.

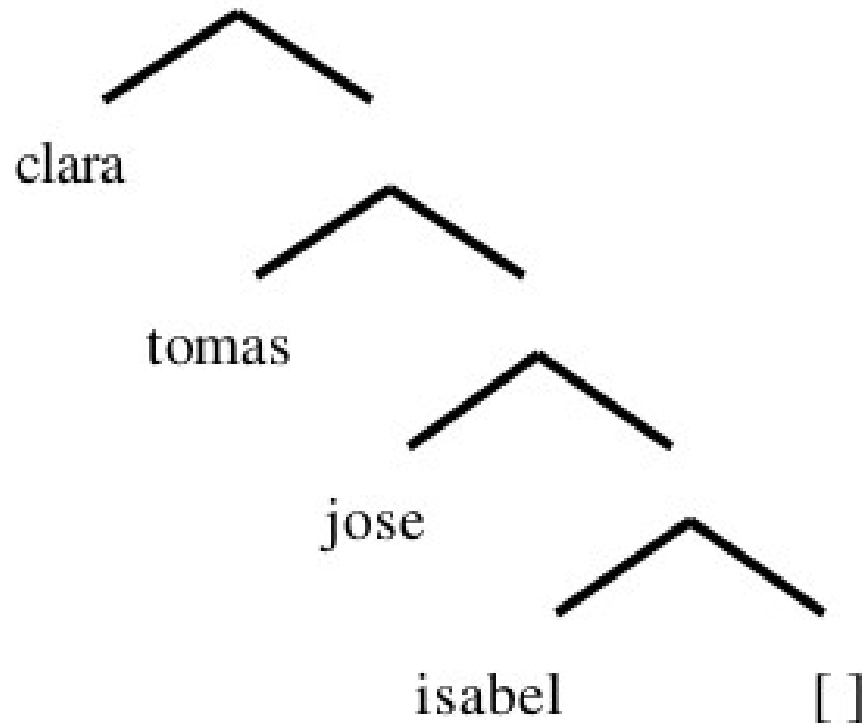
Listas

Una lista es una secuencia de elementos tales como:
[clara,tomas,jose,isabel]

La representación interna de las listas en PROLOG es con árboles binarios, donde la rama de la izquierda es el primer elemento –o cabeza– de la lista y la rama de la derecha es el resto –o cola– de la lista.

[]. Es una lista vacia. Y es el final de la lista (igual que funcional)

Listas



En el ejemplo anterior, la cabeza será clara y la cola [tomas,jose,isabel].

El último elemento siempre es la lista vacía ([]).

Listas

La cabeza y la cola de una lista se pueden separar con el símbolo “|”.

[a,b,c]

[a|[b,c]]

[a,b|[c]]

[a,b,c|[]]

[a|X],[Y|[b,c]]

?- [a,b,c] == [a,b|[c]].

true.

El orden de los elementos en la lista importa y un elemento se puede repetir en una lista.

Listas

Supongamos que queremos determinar si un elemento es miembro de una lista. Los siguientes serían ejemplos del funcionamiento de la relación “miembro”.

miembro(b,[a,b,c]). %PROLOG respondería sí.
miembro(b,[a,[b,c]]). %PROLOG respondería no.
miembro([b,c],[a,[b,c]]). %PROLOG respondería sí.

miembro(X,[X|_]).
miembro(X,[_|R]):-miembro(X,R).

Operadores aritméticos

$X+Y$	suma de X e Y
$X-Y$	X menos Y
$X*Y$	producto de X por Y
X/Y	cociente real de la división de X por Y
$X//Y$	cociente entero de la división de X por Y
$X \bmod Y$	resto de la división entera de X por Y
$\text{abs}(X)$	valor absoluto de X
$\text{sqrt}(X)$	raíz cuadrada de X
$\log(X)$	logaritmo neperiano de X

El siguiente operador permite evaluar expresiones:

$X \text{ is } Y$

“is” en Listas

longitud([],0).

longitud(_|Resto,N):-

longitud(Resto,N1), N is N1+1.

Predicados aritméticos

$X ::= Y$	cierto si los valores numéricos de X e Y son iguales
$X \neq Y$	cierto si los valores numéricos de X e Y son distintos
$X < Y$	cierto si el valor numérico de X es menor que el de Y
$X \leq Y$	cierto si el valor numérico de X es menor o igual que el de Y
$X > Y$	cierto si el valor numérico de X es mayor que el de Y
$X \geq Y$	cierto si el valor numérico de X es mayor o igual que el de Y

```
% suma(+X,+Y,?Z): cierto si Z es la suma de X e Y
suma(X,Y,Z) :- Z is X+Y.
```

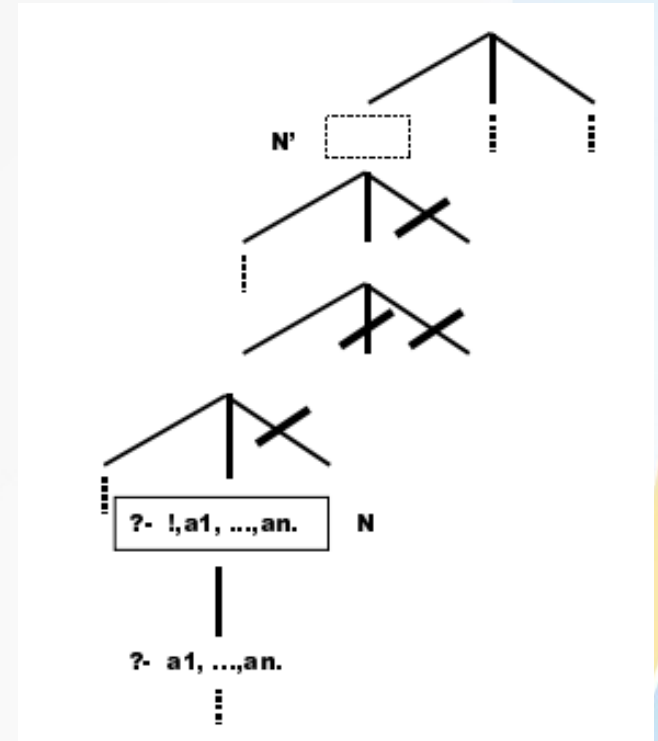
```
% factorial(+X, ?Y): cierto si Y es el factorial de X.
factorial(0, 1).
factorial(X, Y) :-
    X > 0, X1 is X-1, factorial(X1, FactX1), Y is X*FactX1.
```

Predicado de control. Corte (!)

El corte es un predicado predefinido que se denota mediante un punto de exclamación (!), no tiene argumentos, y cuya evaluación es siempre cierta.

Los cortes permiten al programador intervenir en el control del programa, puesto que su presencia hace que el sistema ignore ciertas ramas del árbol.

Su utilidad básica, dado que reduce el espacio de búsqueda de soluciones, es mejorar la eficiencia de los programas, evitando la exploración de partes del árbol de resolución de las que se sabe de antemano que no conducirán a ninguna nueva solución.



Ejemplo de corte (!)

$$fun(x) = \begin{cases} 0, & \text{si } x \leq 10 \\ 1, & \text{si } 10 < x \leq 20 \\ 2, & \text{si } x > 20 \end{cases}$$

Una primera aproximación para la resolución del problema anterior es definir un predicado $f(X,Y)$, cierto si Y es igual a $fun(X)$, mediante las tres siguientes reglas:

$f(X,0) \text{ :- } X \leq 10.$

$f(X,1) \text{ :- } X > 10, X \leq 20.$

$f(X,2) \text{ :- } X > 20.$

La representación anterior calcula correctamente los valores de la función, pero tiene el siguiente inconveniente.

Supóngase que se realiza la consulta:

?- $f(0,Z), Z > 1.$

Ejemplo de corte (!)

La respuesta de Prolog será “no”, pero para llegar a dicha conclusión el sistema tiene que recorrer las 3 posibles ramas del árbol

Lo anterior es poco eficiente, puesto que, al ser las tres reglas que describen el predicado f mutuamente excluyentes, una vez que se ha encontrado una solución con una de ellas no tiene sentido probar con el resto. En efecto, la función que se está calculando tiene la siguiente estructura condicional:

si $X \leq 10$ entonces $Y = 0$;

si no: si $X \leq 20$, entonces $Y = 1$;

si no: $Y = 2$

Ejemplo de corte (!)

```
f(X,Y) :- X =< 10, !, Y=0.  
f(X,Y) :- X =< 20, !, Y=1.  
f(_X,2).
```

Con esta nueva versión, la respuesta de Prolog a la consulta

?- f(0,Z), Z>1.

será también “no”, pero ahora, gracias a la introducción del corte en la primera regla, el sistema solo tendrá que explorar la primera rama del árbol

Ejemplos de programa

PROGRAMA LÓGICO DEFINIDO	PROGRAMA PROLOG
<pre>Factorial(0,s(0)) ← Factorial(s(x),s(x)*y) ← Factorial(x,y)</pre>	<pre>% factorial(?X,?Y) % cierto si Y es el factorial de X factorial(0,s(0)). factorial(s(X),s(X)*Y) :- factorial(X,Y).</pre>
<pre>Suma(x,0,x) ← Suma(x,s(y),s(z)) ← Suma(x,y,z).</pre>	<pre>% suma(?X,?Y,?Z) % cierto si Z es la suma de X e Y suma(X,0,X). suma(X,s(Y),s(Z)) :- suma(X,Y,Z).</pre>

Modo depuración

Los sistemas Prolog suelen incorporar un mecanismo de depuración que permite seguir paso a paso la ejecución de las consultas, establecer puntos de corte en la ejecución, etc. Su funcionamiento básico es el siguiente:

 para entrar en modo de depuración


?- trace.

 para desactivar el modo de depuración:

?- notrace.

Herramientas

 SWI-Prolog

 <https://swish.swi-prolog.org>

Preguntas



