Introducción a Prolog

Fco. Javier Bueno

2021

Prolog = PROgramación LOGica

El usuario proporciona una descripción del problema escrita según cierta convención y el intérprete de Prolog aplica razonamiento lógico para encontrar respuesta al problema.

Otros paradigmas de programación	Prolog
El usuario indica **cómo** se debe resolver el problema	El usuario describe **cuál** es el problema
El sistema ejecuta las instrucciones en un orden dado	El sistema aplica deducción lógica

¿Cómo son los programas en Prolog?

Constan de cláusulas, que se pueden dividir en **hechos** y **reglas**.

- **Hecho**: unidad de información que se asume como cierta.
- **Regla**: afirmación condicional.

Los programas en Prolog se cargan en el intérprete (consult) y se ejecutan mediante preguntas. En el siguiente ejemplo constan varios hechos y un par de reglas:

```
%HECHOS
mujer(maria).
mujer(sara).
mujer(pepa).

hombre(carlos).
hombre(pedro).
hombre(juan).

padre(juan,pedro).        %juan es el padre de pedro
padre(juan,sara).
padre(carlos,pepa).

madre(maria,pedro).
madre(maria,sara).
madre(maria,luis).

%REGLAS
hermanos(X,Y):- padre(Z,X),padre(Z,Y). %Añadir que, además, X\=Y para
hermanos(X,Y):- madre(Z,X),madre(Z,Y). %que no se falseen los resultados
```

A todos ellos (reglas y hechos) se les puede denominar de forma genérica como **predicados**, de modo que en el programa hay 5 tipos de predicados distintos (mujer, hombre, padre, madre y hermanos).

¿Cómo se interpretan los predicados?

Los hechos se suelen interpretar de forma directa de manera que `mujer(maria)` significa que "María es una mujer". Sin embargo, en ocasiones es necesario aclarar el significado del hecho como en el caso de `padre(juan,pedro)`, donde es necesario especificar que "Juan es el padre de Pedro" porque hemos decidido que se interprete de ese modo. Es necesario aclarar que esta decisión tendrá consecuencias a la hora de implementar el resto del programa y condicionará la forma en la que definamos e interpretemos las reglas.

Así, si por ejemplo tomamos la regla:

```
hermanos(X,Y):- padre(Z,X),padre(Z,Y).
```

Significará que "X e Y son hermanos si existe un Z tal que Z es padre de X y Z es padre de Y", lo que viene condicionado por la interpretación del predicado `padre` tal y como hemos decidido en el párrafo anterior.

En las reglas se distinguen dos partes separadas por el operador :- y se pueden interpretar como un `if – then` típico de otros lenguajes de programación:

`cabecera de la regla` :- `cuerpo de la regla`.

De modo la regla se puede interpretar así "**Si** el cuerpo es cierto, **entonces** la cabecera es cierta" o, tal y como hemos visto antes, "La cabecera es cierta si el cuerpo es cierto".

Por otro lado, si nos fijamos en el cuerpo de la regla, se observa que el símbolo ',' expresa la operación lógica AND. En el caso de la operación OR, se puede expresar mediante los símbolos '.' y ';'.

¿Cómo se formulan preguntas?

De varias formas. Algunas pueden ser como las siguientes:

```
?- mujer(maria).
true.

?- mujer(X).
X = maria ;
X = sara ;
X = pepa.

?- padre(juan,pedro).
true.

?- padre(juan,pepe).
false.
```

```
?- padre(juan,X).
X = pedro ;
X = sara.

?- hermanos(X,pedro).
X = sara ;
X = sara ;
X = luis.
```

Elementos básicos de Prolog

Se distinguen entre constantes, variables y estructuras, también llamados términos complejos.

Constantes	Átomos	pepe, the_king, felipe_2, … 'Pepe', 'The King', 'Felipe II',… , . ; :-
	Números	Enteros o reales
Variables		X, Y, Pepe, X_25, _head, _tail, _entrada, Salida, _
Estructuras		mujer(maria), padre(juan,pedro), …

Donde 'mujer' o 'padre' reciben en nombre de *functores* y 'maria','juan' o 'pedro' reciben el nombre de argumentos. Al número de argumentos de una estructura se le denomina *aridad*.

A la variable _ se la denomina *variable anónima*.

Unificación de términos

Dos términos se unifican si ambos son iguales o si contiene variables que pueden ser instanciadas de modo que los términos resultantes sean iguales.

Por ejemplo, si realizamos la pregunta `hombre(X)` el primer resultado devuelto será `carlos` ya que Prolog unifica `hombre(X)` con `hombre(carlos)` al poder instanciar X con el valor `carlos`. A continuación, Prolog intentará unificar el término solicitado con el resto de predicados hombre devolviendo tantas soluciones como unificaciones logre.

```
?- hombre(X).
X = carlos ;
X = pedro ;
X = juan.
```

Recursividad

Prolog basa su potencia de resolución de problemas lógicos tanto en la unificación con en la recursividad.

En general una definición recursiva tiene el siguiente aspecto:

> "Se dice que una entidad es cierta si,
>
> o bien es algo trivial
>
> o bien es algo que de forma general se puede caracterizar por una relación recursiva."

En nuestro caso definiremos un **caso base** (solución trivial) y un **caso general**, que incluirá la llamada recursiva al predicado que estamos implementando.

Sea el siguiente ejemplo donde el predicado descendiente se define de forma recursiva:

```
%HECHOS
hijo(pedro,pablo).          %pedro es hijo de pablo.
hijo(pablo,josé).
hijo(josé,javier).
hijo(javier,héctor).

%REGLAS
descendiente(X,Y):- hijo(X,Y).                      %caso base
descendiente(X,Y):- hijo(X,Z),descendiente(Z,Y).    %caso general
```

Su interpretación es, como se ha visto anteriormente:

Caso base: "Si X es hijo de Y, entonces X es descendiente de Y"

Caso general: "Si X es hijo de Z y Z es descendiente de Y, entonces X es descendiente de Y".

Listas

Colección de objetos entre corchetes. Estos objetos pueden ser átomos, números, variables, estructuras u otras listas.

Ejemplos:

[a,b,c,d,e], [pepe, carlos, juan, javier], [ ], etc.

Todas las listas se pueden dividir en dos partes: cabeza y cola [Head | Tail]

En la lista [pepe, carlos, juan, javier] el elemento pepe constituye la cabeza y la sublista [carlos, juan, javier] constituye la cola.

El operador | se utiliza para descomponer las listas en dos partes. Algunos ejemplos:

```
?- [Head|Tail]=[pepe,carlos,juan,javier].
Head = pepe,
Tail = [carlos, juan, javier].
```

```
?- [X,Y|Cola]=[pepe,carlos,juan,javier].
X = pepe,
Y = carlos,
Cola = [juan, javier].
```

Observar el uso de la variable anónima:

```
?- [X,_,Y|_]=[pepe,carlos,juan,javier].
X = pepe,
Y = juan.
```

También se puede usar el operador | para construir listas:

```
?- X=[0|[1,2]].
X = [0, 1, 2].
```

Recorrido de listas

Para recorrer listas es necesario construir reglas recursivas. Veamos por ejemplo un predicado que permite calcular la longitud de una lista dada:

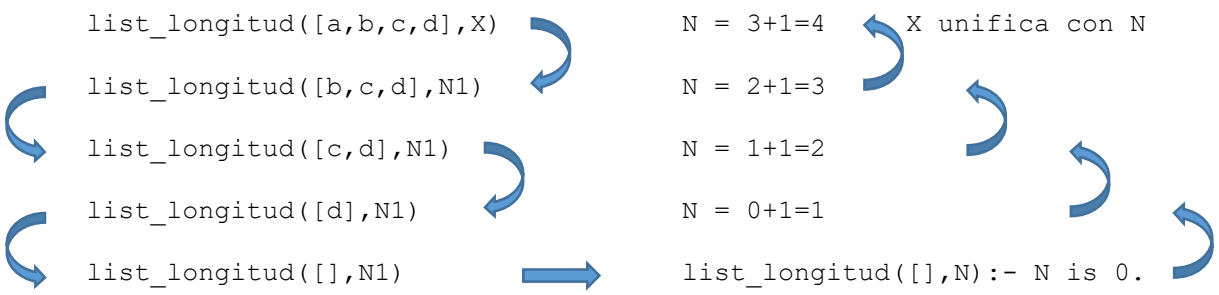
```
list_longitud([_|Y],N):- list_longitud(Y,N1), N is N1 + 1.
list_longitud([],N):- N is 0.
```

Si preguntamos por la longitud de la lista [a,b,c,d] el programa devuelve 4 como era de esperar.

```
?- list_longitud([a,b,c,d],X).

X = 4.
```

Internamente el proceso de llamadas de forma recursiva es el siguiente:



Donde el valor de la variable N se va actualizando al deshacer el proceso de llamadas recursivas al predicado `list_longitud`.

¿Cómo implementar predicados recursivos para construir listas?

Si tomamos como ejemplo el árbol genealógico de la familia Corleone y el predicado `descendiente` visto con anterioridad, podemos implementar un predicado recursivo que permita obtener la lista de descendientes de un personaje dado.

De este modo, creamos una función auxiliar donde vamos construyendo la lista de forma recursiva para, finalmente, copiarla en el argumento en el que el predicado devolverá la lista consultada:

```
lista_descendientes(X,Y):-lista_descendientes_aux(X,[],Y).

lista_descendientes_aux(X,Z,Y):-
                              descendiente(X1,X),
                              not(member(X1,Z)),
                              lista_descendientes_aux(X,[X1|Z],Y),!.
lista_descendientes_aux(_,X,X).
```

Interacción con el usuario y consulta de ficheros

La interacción con el usuario se logra mediante el uso de los predicados `write` y `read`. El primero permite sacar un mensaje por pantalla, que se evalúa siempre a `true`. Para leer del teclado se utiliza el predicado `read` tal y como se muestra en este ejemplo:

```
:-read(X),write('Has dicho...'),write(X).
```

Por otro lado, el predicado `consult` permite cargar en el intérprete de Prolog cualquier otro fichero y utilizarlo como si formase parte del archivo en el que se incluye dicha órden. En el ejemplo de la familia Corleone, podemos tener el árbol genealógico en un fichero y el programa que trate con los parentescos en otro fichero separado. Esto permite organizar nuestros programas en varios archivos de extensión moderada facilitando su manejo y legibilidad.

```
:-consult(arbol_familia_corleone).
%--------------------------RELACIONES--------------------------------
% PADRE, MADRE
padre_de(X,Y):- hombre(X), progenitor(X,Y).       /*X es el padre de Y*/
madre_de(X,Y):- mujer(X), progenitor(X,Y).        /*X es la madre de Y*/
padres_de(X,Y,Z):- padre_de(X,Z), madre_de(Y,Z).
...
```

Uso del operador de transformación =.. y del predicado `call`

Uno de los predicados solicitados en la práctica 0 es `relacion(X,Y,Relacion)` que será cierto si `Relacion(X,Y)` es cierta siendo `Relacion` es una de las relaciones definidas en la práctica (`padre, madre, hermano-a, abuelo-a`, etc.).

Una posible solución es la siguiente, que implica el uso del operador =.. y del predicado `call`:

```
% Generación de relaciones

 relaciones([esposos,progenitor,hijo,hermano,abuelo,nieto,tio,sobrino,
 primo,suegro,cunado,yerno,nuera,ancestro,descendiente,padre_de,madre_
 de]).

% relacion(X,Y,Relacion) cierta si Relacion(X,Y) es cierta. Por ejemplo,
relacion(X,Y,sobrino) cierta si X es sobrino de Y.

 relacion(X,Y,Relacion):-

        relaciones(L),

        member(Relacion,L),

        Z=..[Relacion,X,Y],

        call(Z).
```

La lista de relaciones se carga en `L` y se comprueba si `Relación` está incluido en esa lista. En caso positivo, el operador =.. transforma una estructura en una lista y viceversa, de modo que `Z` se unifica con `Relacion(X,Y)`. Por último, `call` se encarga de ejecutar el predicado `Relacion(X,Y)`.