Practicas de Prolog

Juan Meza

- Con esta presentacion muestro las diferentes características del núcleo del lenguaje *Prolog mediante sencillos* ejemplos. Se ha realizado un tratamiento especial a las listas y a los diversos esquemas recursivos para trabajar con listas. El objetivo es ayudar al lector a identificar esquemas de programación que puedan servirle para resolver problemas similares. No se ha pretendido realizar una guía detallada sino una presentación práctica informal del lenguaje. El lector que esté interesado en profundizar en el lenguaje, deberá consultar los libros de la bibliografía.

- **Hechos**
- **Reglas**
- ***Reglas Recursivas***
- ***Utilización de funciones***
- ***Obtener un valor a partir de una lista***
- ***Acumulación de Resultados***

Hechos

```
/* Relacion Progenitor */
progenitor(pilar,belen).
progenitor(tomas,belen).
progenitor(tomas,lucia).
progenitor(belen,ana).
progenitor(belen,pedro).
progenitor(pedro,jose).
progenitor(pedro,maria).
```

- Se describen una serie de hechos conocidos sobre una familia.
- **Sintaxis de Prolog: Constantes y predicados:** empiezan por minúscula. Los hechos acaban en punto. Variables comienzan por mayúscula.

Preguntas

progenitor(pilar,belen).

progenitor(pilar,lucia).

progenitor(belen,X).

Reglas

```
progenitor(pilar,belen).
progenitor(tomas,belen).
progenitor(tomas,lucia).
progenitor(belen,ana).
progenitor(belen,pedro).
progenitor(pedro,jose).
progenitor(pedro,maria).

madre(X,Y):-mujer(X),
progenitor(X,Y).
mujer(pilar). mujer(belen).
mujer(lucia). mujer(ana).
mujer(maria).
hombre(tomas). hombre(pedro).
hombre(jose).
```

Preguntas

madre(belen,pedro).

madre(X,belen).

madre(belen,X).

madre(X,Y).

Reglas Recursivas

```
progenitor(pilar,belen).
progenitor(tomas,belen).
progenitor(tomas,lucia).
progenitor(belen,ana).
progenitor(belen,pedro).
progenitor(pedro,jose).
progenitor(pedro,maria).

madre(X,Y):-mujer(X),
progenitor(X,Y).
mujer(pilar). mujer(belen).
mujer(lucia). mujer(ana).
mujer(maria).
hombre(tomas). hombre(pedro).
hombre(jose).

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

Preguntas

antepasado(belen,X).

antepasado(X,belen).

Utilización de funciones

```
/* Relacion Progenitor */
progenitor(pilar,belen).
progenitor(tomas,belen).
progenitor(tomas,lucia).
progenitor(belen,ana).
progenitor(belen,pedro).
progenitor(pedro,jose).
progenitor(pedro,maria).

madre(X,Y):-mujer(X),
progenitor(X,Y).
mujer(pilar). mujer(belen).
mujer(lucia). mujer(ana).
mujer(maria).
hombre(tomas). hombre(pedro).
hombre(jose).

antepasado(X,Y):-progenitor(X,Y).
antepasado(X,Y):-progenitor(X,Z),
antepasado(Z,Y).

grande(pepe).
grande(cabeza(juan)).
grande(X):-mayor(X,Y).
mayor(cabeza(X),cabeza(Y)):- progenitor(X,Y).
```

Preguntas

grande(X).

Se utiliza la función: *cabeza(x)="cabeza de x"*
El programa indica:
"Pepe es grande, la cabeza de juan es grande, si X es mayor que Y, entonces X es grande, además: La cabeza de X es mayor que la de Y si X es el progenitor de Y"

Obtener un valor a partir de una lista

Suma de un vector

```
sum([],0).
sum([X|Xs],S):-sum(Xs,Sc), S is Sc + X.
```

Las preguntas:

```
?- sum([1,2,3,4],V).
V = 10
```

```
long([],0).
long([X|Xs],L):-long(Xs,Lc), L is Lc + 1.
```

Preguntas:

?- long([1,2,3,4],V).

V = 4

```
prod([],1).
prod([X|Xs],P):-prod(Xs,Pc), P is Pc * X.
```

Preguntas:

?- prod([1,2,3,4],V).

V = 24

- Obsérvese la similitud entre las tres definiciones. En algunos lenguajes se utilizan construcciones de orden superior que permiten utilizar una única definición parametrizada por las operaciones y constantes

- Las definiciones ofrecidas no aprovechan la *optimización de la recursividad de cola. Consiste* en que el último objetivo de una definición recursiva sea el predicado que se está definiendo. Los sistemas con dicha optimización permiten que las definiciones recursivas se comporten de forma similar a un bucle en un lenguaje imperativo.

```
sum1(Xs,S):-sumAux(Xs,0,S).
sumAux([],S,S).
sumAux([X|Xs],Sa,S):-Sn is X + Sa,
sumAux(Xs,Sn,S).
```

Preguntas:

?- sum1([1,2,3,4],V).

V = 10

Acumulación de Resultados

- A diferencia de los lenguajes imperativos, *Prolog utiliza variables lógicas. En el momento en que una variable* lógica es instanciada, dicho valor no puede modificarse. De esta forma, no es posible utilizar variables globales cuyo valor se modifique durante la resolución del objetivo. Existen ciertos algoritmos que requieren la utilización de un estado que almacena resultados intermedios. Para implementar dichos algoritmos es necesario utilizar un predicado auxiliar con un argumento extra que almacenará el estado que se modifica

```
sumAcum(Xs,Ys):-sumAc(Xs,0,Ys).
sumAc([],S,[]).
sumAc([X|Xs],Sa,[Sp|Ys]):-Sp is X + Sa, sumAc(Xs,Sp,Ys).
```

$$\texttt{sumAcum(Xs, Ys)} :- \; y_j = \sum_{i=1}^{j} x_i \;\; \text{para cada} \;\; y_j \in Ys$$

Preguntas:

?- sumAcum([1,2,3,4],V).

V = [1, 3, 6, 10] ;

Combinación miembro a miembro de los elementos de dos listas

```
prodEscalar(Xs,Ys,P):- pEsc(Xs,Ys,0,P).
pEsc([],[],P,P).
pEsc([X|Xs],[Y|Ys],Pa,Pr):-Pn is Pa + X * Y, pEsc(Xs,Ys,Pn,Pr).
```

prodEscalar(Xs,Ys,P):-P es el producto escalar de los vectores Xs e Ys ($P = \Sigma x_i y_i$)

Preguntas:

?- prodEscalar([1,2,3],[4,5,6],P).

P = 32 ;

Generación de una lista a partir de un valor

- Es posible generar una lista mediante la descomposición de un valor. En el primer ejemplo, se descompone un número natural hasta llegar a cero, en el segundo, se descompone un intervalo hasta que los extremos son iguales.

```
repite(0,X,[]).
repite(N,X,[X|Xs]):-N > 0, N1 is N - 1,
repite(N1,X,Xs).
```

repite(N,X,Xs):- Xs con N elementos de valor X

Preguntas:

?- repite(3,a,V).

V = [a, a, a] ;

```
intervalo(X,X,[X]).
intervalo(X,Y,[X|Xs]):-X < Y, Z is X + 1,
intervalo(Z,Y,Xs).
```

intervalo(X,Y,Xs):-Xs es una lista creciente cuyo primer valor es *X y su último valor Y*

Preguntas:

?- intervalo(1,5,V).

V = [1, 2, 3, 4, 5]

Generación de listas por filtrado de elementos

- Los siguientes ejemplos muestran cómo se puede generar una o varias listas filtrando los elementos de otras listas.
  - En *pares, la condición de filtrado (ser par) indica si se inserta o no el elemento*
  - En *inserta la condición de filtrado (ser menor o mayor que el valor X) indica si se selecciona el elemento Y de* la lista que se recorre o el elemento *X a insertar*
  - En *particion la condición de filtrado indica en qué lista se inserta*
  - En el último caso, se recorren dos listas y la condición indica de cuál de las listas seleccionar el valor

```
inserta(X,[],[X]).
inserta(X,[Y|Ys],[X,Y|Ys]):-X < Y.
inserta(X,[Y|Ys],[Y|Zs]) :-X >= Y,
inserta(X,Ys,Zs).
```

inserta(X,Xs,Ys):-Ys es el resultado de insertar *X* en la posición adecuada de la lista ordenada *Xs* de forma que *Ys* se mantenga ordenada.

Preguntas:

?- inserta(3,[1,2,4],V).

V = [1, 2, 3, 4] ;