2. Tratamiento de listas en PROLOG

CONTENIDO

2.1. Listas en PROLOG

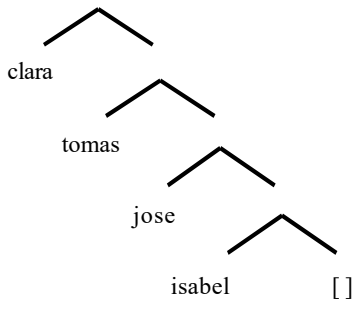
La lista es una estructura de datos simple, muy usada en la programación no numérica. Una lista es una secuencia de elementos tales como:

clara, tomas, jose, isabel

La sintaxis de PROLOG para las listas consiste en englobar las secuencias de elementos entre corchetes.

[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. De la rama que contiene el resto de la lista también se distinguen dos ramas: la cabeza y la cola. Y así sucesivamente hasta que la rama de la derecha contenga la lista vacía (representado por [])



De cualquier forma consideraremos dos casos de listas: Las listas vacías y las no vacías. En el primer caso escribiremos la lista vacía como un átomo de PROLOG, []. Y en el segundo caso, la lista la consideraremos formada por dos partes; el

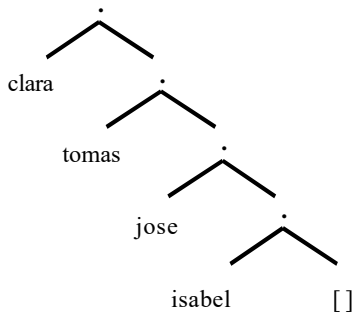
primer elemento, al que llamaremos la cabeza de la lista; y el resto de la lista al que llamaremos la cola de la lista.
En el ejemplo anterior, la cabeza será `clara` y la cola `[tomas,jose,isabel]`.

Aunque como hemos visto anteriormente no es necesario, en general la estructura lista se representa en PROLOG con el nombre de predicado o funtor ".".

Ejemplo 2.1

```
.(clara,.(tomas,.(jose,.(isabel,[]))))
```

Que PROLOG representará internamente como un árbol:



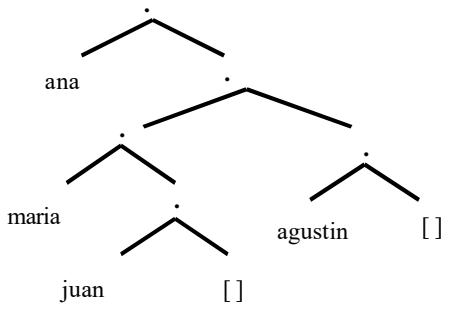
El último elemento siempre es la lista vacía ([]).

Como en PROLOG el manejo de listas es muy usual, se utiliza la notación simplificada con corchetes, que es equivalente a la anterior ([clara, tomas, jose, isabel]). Interiormente PROLOG lo interpretará con la notación árbol.
Los elementos de una lista pueden ser de cualquier tipo, incluso otra lista.

Ejemplo 2.2

```
[ana, [maria, juan], agustin]
```

PROLOG representará internamente como un árbol:



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

Ejemplo 2.3
Las siguientes son expresiones válidas sobre listas y se refieren a la misma lista:

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

El orden de los elementos en la lista importa y un elemento se puede repetir en una lista, de forma similar a como ocurre en las tuplas y pares ordenados.

2.2. Ejemplos de solución de problemas de listas en PROLOG

La práctica en el manejo de listas con PROLOG es básica para el aprendizaje del lenguaje. Ese es el objetivo de este capítulo.
Para resolver un problema de listas con PROLOG, primero pondremos varios ejemplos de su funcionamiento, para determinar los argumentos que son necesarios en la definición de la relación y la respuesta de PROLOG en distintos casos, que nos indicará el comportamiento deseado de la cláusula.

Ejemplo 2.4
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.
```

El símbolo "%" permite poner comentarios que ocupen sólo la línea donde aparece dicho símbolo. Para comentarios más largos se puede utilizar los símbolos "/*" para empezar y "*/" para terminar.
La siguiente podría ser una definición de la relación "miembro":

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

El elemento será miembro de la lista si coincide con la cabeza de la lista (independientemente de lo que contenga el resto de la lista, por eso ponemos una variable anónima en su lugar). En este caso el predicado será cierto y PROLOG no comprobará nada más. Si no es así, podrá ser miembro si lo es del resto de la lista, en la llamada recursiva de la segunda definición del predicado. La cabeza de la lista es una variable anónima, porque cuando vaya a comprobar esta segunda regla, es seguro que la primera no se cumple (es decir, el elemento no coincide con la cabeza de la lista), ya que PROLOG busca la cláusula en el orden de aparición en la base de datos.

Ejemplo 2.5
Suponemos que queremos concatenar dos listas y devolver el resultado en una tercera lista, es decir:

```
concatenar([a,b,c],[d,e],[a,b,c,d,e]).  % PROLOG respondería que sí.
concatenar([a,b],[1,2,3],X).  % PROLOG respondería X=[a,b,1,2,3].
```

Solución

```
concatenar([],L,L).
concatenar([X|L1],L2,[X|L3]):- concatenar(L1,L2,L3).
```

La lista resultante se construye (o se comprueba) en la cabeza de la regla recursiva. En la regla que corresponde a la salida de la recursividad se inicializa el argumento resultado a la lista que aparece en el segundo argumento.

Ejercicio 2.1

Utilizar la definición de concatenación para:

a) Descomponer una lista en dos, obteniendo todas las posibles descomposiciones.
b) Borrar algunos elementos de una lista.
Por ejemplo: considera la lista L1=[a,b,c,d,z,z,e,z,z,z,f,g] y borra todos los elementos que siguen a la secuencia z,z,z.
c) Borrar los tres últimos elementos de una lista.
d) Definir de nuevo la operación miembro.
e) Definir la relación para extraer el último elemento de la lista
    e.1) Utilizando la definición de concatenación.
    e.2) Sin utilizarla.

Ejercicio 2.2

Borrar un elemento de una lista.

Ejercicio 2.3

Añadir un elemento en una lista.

Ejercicio 2.4

Comprobar si dos elementos son consecutivos en una lista.

Ejemplo 2.6
Calcular la inversa de una lista.
Ejemplo de utilización:

```
inversa([a,b,c,d],[d,c,b,a]).  %PROLOG devolvería sí.
```

Solución 1:

```
inversa([],[]).
inversa([X|L1],L):-
        inversa(L1,Resto),
        concatenar(Resto,[X],L).
```

Solución 2:

```
inversa(L1,L):- inversa(L1,[],L).
inversa([],L,L).
inversa([X|L1],L2,L3):- inversa(L1,[X|L2],L3).
```

/* Para las dos últimas reglas es indistinto el orden, porque sólo una de las dos será cierta cada vez */

En esta segunda versión, se utiliza otro predicado inversa/3 (de aridad 3, y por tanto distinto al predicado inversa/2), en donde el segundo argumento se inicializa a lista vacía. En esta versión la lista se construye (o se comprueba) en el cuerpo de la regla de la segunda regla inversa/3 (al contrario de como sucedía en la definición del ejemplo 2.3). Para estos casos, la salida de la recursividad deberá realizar un "volcado" de la lista construida en el segundo argumento en la lista resultante, que aparece en este caso en el tercer argumento. De esta manera el resultado quedará almacenado en la lista resultante ya que el resultado obtenido en el segundo argumento se irá perdiendo a medida que se cierre la recursividad.
Muchas veces, para un mismo problema, se pueden plantear ambas versiones, la versión que construye o comprueba en la cabeza de la regla recursiva y la versión que construye o comprueba en el cuerpo de la regla. La primera solución es más elegante. La segunda solución, sin embargo, puede resultar más fácil para programadores de lenguajes procedurales, ya que la lista se construye "manualmente", desde la lista vacía. Es interesante conocer ambas formas de resolverlo para poder elegir la más conveniente a cada problema.

Ejercicio 2.5

Comprobar si una lista es un palíndromo.
(Un palíndromo es una palabra que se lee igual de izquierda a derecha que de derecha a izquierda)

Ejercicio 2.6

Escribir un conjunto de cláusulas que resuelvan las siguientes operaciones sobre conjuntos:

a) Subconjunto.
Ejemplo 1: ?- subconjunto([b,c],[a,c,d,b]). % El resultado sería verdadero

Ejemplo 2: ?- subconjunto([a,c,d,b],[b,c]). /* El resultado sería verdadero, pero se obtendría con un conjunto de cláusulas distintas a las de la solución para el ejemplo 1. Probad a escribir el conjunto de cláusulas necesarias para el caso del ejemplo 1 y para el caso del ejemplo 2. */

b) Disjuntos.
Ejemplo: ?- disjuntos([a,b,d],[m,n,o]). % El resultado sería verdadero

Ejercicio 2.7

Definir dos predicados:

par(Lista), impar(Lista)

que serán verdaderos si Lista tiene un número par e impar de elementos, respectivamente.

Ejercicio 2.8

Definir la relación

shift(L1,L2)

tal que L2 es la lista L1 después de una rotación de un elemento a la izquierda.
Ejemplo: ?- shift([a,b,c,d],L), shift(L,L1).
Da como resultado:

L=[b,c,d,a]
L1=[c,d,a,b]

Ejercicio 2.9

Definir la relación

trasladar(L1,L2)

que permita trasladar una lista de números (L1) a una lista de sus correspondientes nombres (L2).

Ejemplo: ?- trasladar([1,2,3,7],[uno,dos,tres,siete]). % El resultado sería veradero

Utiliza la siguiente relación auxiliar:

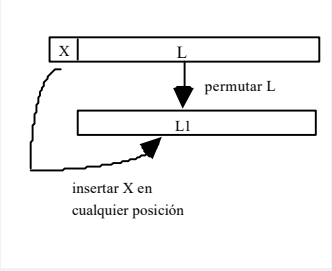
significa(0,cero).
significa(1,uno).
significa(2,dos).
etc.

Ejercicio 2.10

Calcular las permutaciones de una lista.
Ejemplo:

?- permutacion([a,b,c,],P).

P=[a,b,c];
P=[a,c,b];
P=[b,a,c];…



Pista: para insertar en cualquier posición se puede utilizar el predicado borrar.

Si queremos contar los elementos de una lista en PROLOG, necesitamos operadores para sumar. Los siguientes son nombres de predicados en PROLOG que son predefinidos (ya están definidos, sólo hay que utilizarlos) con un comportamiento distinto del explicado hasta ahora y que como efecto colateral nos permitirán sumar, etc.

2.3. Construcción de expresiones aritméticas en PROLOG

A continuación se muestran los operadores que se pueden utilizar para construir expresiones:

```
X+Y              /*suma*/
X-Y              /*resta*/
X*Y              /*multiplicación*/
X/Y  ó X div Y   /*división real y entera*/
X mod Y          /*resto de la división de X entre Y*/
X^Y              /*X elevado a Y*/
-X               /*negación*/

abs(X)           /*valor absoluto de X*/
acos(X)          /*arco coseno de X*/
asen(X)          /*arco seno de X*/
atan(X)          /*arco tangente de X*/
cos(X)           /*coseno de X*/
exp(X)           /*exponencial de X; [e^X]*/
ln(X)            /*logaritmo neperiano de X*/
log(X)           /*logaritmo en base 2 de X*/
sin(X)           /*seno de X*/
sqrt(X)          /*raíz cuadrada de X*/
tan(X)           /*tangente de X*/
round(X,N)       /*redondeo del real X con N decimales*/
```

2.4. Comparación de términos en PROLOG.

Los siguientes operadores son los que permiten comparar términos (véase el apartado 1.5 para un repaso del concepto de término):

```
X<Y              /*X es menor que Y*/
X>Y              /*X es mayor que Y*/
```

```
X=<Y        /*X es menos o igual que Y*/
X>=Y        /*X es mayor o igual que Y*/
X=Y         /*X es igual que Y*/
X\=Y        /X es distinto que Y*/
```

2.5. Comparación de expresiones en PROLOG.

Una expresión es un conjunto de téminos unidos por operadores aritméticos (los introducidos en el apartado 2.3). Los siguientes predicados predefinidos comparan expresiones sin evaluarlas, mediante una comparación sintáctica siguiendo el siguiente orden:

- variables,
- enteros y reales,
- átomos en orden alfabético,
- términos complejos: aridad, nombre y orden según la definición recursiva.

```
X==Y        /*la expresión X es igual que la expresión Y*/
X\==Y       /*la expresión X es distinta que la expresión Y*/
X@<Y        /*la expresión X es menor que la expresión Y*/
X@>Y        /*la expresión X es mayor que la expresión Y*/
X@=<Y       /*la expresión X es menor o igual que la expresión Y*/
X@>=Y       /*la expresión X es mayor o igual que la expresión Y*/
```

El siguiente operador permite evaluar expresiones:

```
X is Y
```

Ejemplo 2.7

```
X is 3*5+3/2    /*PROLOG contestaría X=9*/
```

Para obtener un resultado equivalente a lo que en los lenguajes procedurales es la asignación, en PROLOG usaremos el predicado “is” para instanciar la variable que aparece a la izquierda a un valor concreto o al resultado de la evaluación de una expresión que aparece a la derecha. Sin embargo, esa asignación sólo podrá ser satisfecha si la variable que aparece a la izquierda del “is” no está instanciada, en caso contrario la operación fallará. Es decir, en PROLOG no se puede cambiar el valor de una variable sin desistanciarla mediante backtracking.

Los siguientes predicados predefinidos comparan términos haciendo una evaluación de expresiones:

```
X=:=Y       /* El resultado de evaluar la expresión X es igual al resultado
               de evaluar la expresión Y*/
X=\=Y       /* El resultado de evaluar la expresión X es distinto al
               resultado de evaluar la expresión Y*/
```

Ejemplo 2.8

Escribir un predicado PROLOG para determinar la longitud de una lista.

Ejemplo: `?-longitud([a,b,c,d,e],5).`
Solución:

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

Para la siguiente consulta al anterior predicado "?-longitud([e,w,f,v],X).", las variables N y N1 no tendrán ningún valor hasta que sean inicializadas en la salida de la recursividad, cuando la lista esté vacía. A partir de ese momento, se contabilizará la longitud de la lista en el cierre de la recursividad. Para una mejor comprensión del funcionamiento del PROLOG es aconsejable ejecutar los programas paso a paso, entendiendo cada uno de los pasos.

Ejercicio 2.11

a) Escribir un predicado PROLOG para que dada una lista de números, determine el número de elementos positivos (el 0 se considera como positivo) y el número de elementos negativos.

Ejemplo: ?-pos_y_neg([3,-1,9,0,-3,-5,-8],3,4). % PROLOG respondería sí.

b) Escribir un predicado PROLOG para que dada una lista de números, construya dos listas, una de números positivos y otra de negativos y que determine la cardinalidad de ambas listas.
Ejemplos:
?-pos_y_neg([3,-1,9,0,-3,-5,-8],3,[3,9,0],4,[-1,-3,-5,-8]). % PROLOG respondería sí.
?-pos_y_neg([3,-1,9,0,-3,-5,-8],Npos,Pos,Nneg,Neg). /* PROLOG respondería:
Npos=3, Pos=[3,9,0], Nneg=4, Neg=[-1,-3,-5,-8] */

Ejercicio 2.12

Dada una lista, escribir los predicados PROLOG necesarios para obtener dos sublistas, una que contenga los elementos de las posiciones pares y otra que contenga los elementos de las posiciones impares.
Ejemplo: ?-par_impar([a,b,c,d,e,f],[b,d,f],[a,c,e]). % PROLOG contestaría que sí.

Ejercicio 2.13

Dada una lista de números, escribir los predicados PROLOG necesarios para hallar el elemento mínimo y el máximo.
Ejemplo: ?-minimo([4,6,2,7,9,0],Min). % PROLOG respondería Min=0
Ejemplo: ?-maximo([4,6,2,7,9,0],Max). % PROLOG respondería Max=9

Ejercicio 2.14

Dada una lista, escribir los predicados PROLOG necesarios para contabilizar las apariciones de un determinado elemento en la lista.
Ejemplo: ?-aparicion(a,[a,b,c,a,b,a],X). % PROLOG contestaría X=3.

2.6. Bibliografía

[Clocksin 93] Clocksin, W.F., Mellish, C.S., Programación en PROLOG. Segunda edición. Colección Ciéncia Informática. Editorial Gustavo Gili S.A., 1993.

[Bratko 94] Bratko,I., PROLOG programming for Artificial Intelligence. Second edition. Addison-Wesley Publishing Company, 1994.

[Konigsberger & Bruyn 90] Konigsberger, H., Bruyn, F., PROLOG from the beginning. McGraw-Hill book Company, 1990.

[Starling & Shapiro 94] Starling, L., Shapiro, E., The art of PROLOG. Second edition. The MIT Press, 1994.

[Starling 90] Starling, L. editor, The practice of PROLOG. The MIT Press, 1990.

[O'Keefe 90] O'Keefe, R.A., The craft of PROLOG. The MIT Press, 1990.

[Stobo 89] Stobo, J., Problem solving with PROLOG. Pitman publishing, 1989.