




# Prolog

Listas, Operadores y Aritmética



Laboratorio de Introducción a los Sistemas Informáticos Inteligentes  
Escuela Superior de Ingeniería Informática - Universidad Vigo

*Nota:*

*Basado en el capítulo 3 del libro [Prolog: Programming for Artificial Intelligence](#) de Ivan Bratko - Ed. Pearson - contenido en la bibliografía recomendada de la asignatura.*

## Contenido

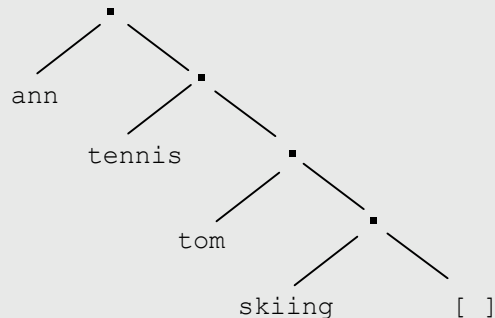
- Representación de listas
- Operaciones sobre listas
  - Pertenencia
  - Concatenación
  - Borrado e inserción de un elemento
  - Sublista
  - Permutaciones
- Operadores
- Operadores aritméticos

## Representación de listas

- Una lista es una secuencia de un número indeterminado de elementos.

```
[ann, tennis, tom, skiing]
```

- Es un árbol, como todo dato estructurado de Prolog :



- Si la lista está vacía es, sencillamente, un **átomo**: []
- Si no está vacía, consta de dos **componentes**:
  - El primer elemento: denominado cabeza (head) de la lista
  - El resto de la lista: denominado cola (tail)
- Ambas están combinadas con un **functor especial**:

```
.(Head, Tail)
```

```
.(ann, .(tennis, .( tom,.(skiing, [ ]))))
```

- Los elementos de la lista pueden ser de cualquier tipo (incluso otras listas)

## Representación de listas

- Existen varias **notaciones** posibles para trabajar con listas.

- Usando []
- Usando el functor •
- Utilizando | (para separar cabecera y cola)

```
Lista1 = [a,b,c].  
Lista2 = .(a,.(b,.(c,[])))  
  
Tail   = [b,c].  
Lista1 = .(a, Tail).  
Lista1 = [a | Tail].
```

- En resumen:

- Una lista es una estructura de datos que o bien está vacía o tiene dos componentes: cabecera y cola
- Las notaciones más usadas para listas son

```
[Item1, Item2, Item3, ...].  
  
[Head | Tail].  
[Item1, Item2, ... | Otros]
```

## Operaciones sobre listas

- Implementaremos las operaciones más habituales sobre listas.
- **Pertenencia.** Un elemento está o no en la lista.

```
member(b, [a,b,c]).           Yes
member(d, [a,b,c,[d,c,e]]).   No
```

- Un elemento X es miembro de una lista L:
  - Si X es la cabecera de la lista o,
  - Si X es miembro de la cola de la lista.

```
member(X, [X | Tail]).

member(X, [Head | Tail]): -
    member(X, Tail).
```

- member es una relación definida en la implementación SWI-Prolog.

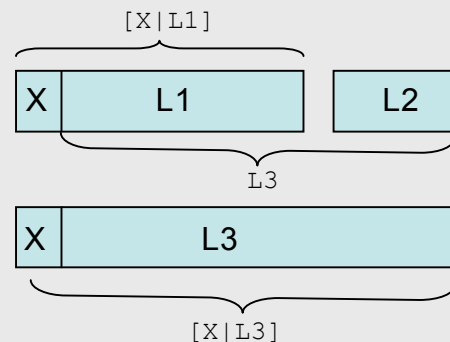
## Operaciones sobre listas

- **Concatenación.** Unión de dos listas en el orden en que aparecen

```
conc([a,b],[c,d],[a,b,c,d]).      Yes
conc([a,b],[c,d],[a,c,b,d]).      No
```

- **Definición.** Dos posibles casos:

- Si el primer argumento es la lista vacía, entonces los argumentos segundo y tercero tienen que ser la misma lista
- Si el primer argumento es una lista no vacía, debe tener una cabecera y una cola, tal que:  $[X|L1]$ , procediendo de forma recursiva podremos concatenar las dos listas.



```
conc([], L, L).

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

- Se puede usar para concatenar dos listas dadas

```
?- conc([a,b,c],[1,2,3], L).
L = [a, b, c, 1, 2, 3]
```

- Esta relación se denomina **append** en la implementación SWI-Prolog.

## Operaciones sobre listas

- Otros usos de la operación de concatenación pueden ser:
  - Usarlo en la dirección contraria para descomponer una lista en dos listas.

```
?- conc(L1,L2,[a,b,c,d]).
```

```
L1 = []  
L2 = [a, b, c, d] ;
```

```
L1 = [a]  
L2 = [b, c, d] ;
```

```
L1 = [a, b]  
L2 = [c, d] ;
```

```
L1 = [a, b, c]  
L2 = [d] ;
```

```
L1 = [a, b, c, d]  
L2 = [] ;
```

```
No
```

- Para buscar un patrón en una lista

```
?- conc(Ant,[may|Post],[ene,feb,mar,abr,may,jun,jul,ago,sep,oct,nov,dic]).
```

```
Ant = [ene, feb, mar, abr]
```

```
Post = [jun, jul, ago, sep, oct, nov, dic]
```

```
?- conc(_, [M1,may,M2|_],[ene,feb,mar,abr,may,jun,jul,ago,sep,oct,nov,dic]).
```

```
M1 = abr
```

```
M2 = jun
```

## Operaciones sobre listas

- Otros usos de la operación de concatenación pueden ser:
  - Usarlo para borrar una parte de la lista a partir de un cierto patrón

```
?- conc(L2, [z,z,z|_], [a,b,z,z,c,z,z,z,d,e]).  
L2 = [a, b, z, z, c] ;
```

- Se podría redefinir la relación member, ya definida, como:

```
miembro1(X,L):-  
    conc(L1, [X|L2], L).
```

- O bien, usando variables anónimas:

```
miembro1(X,L):-  
    conc(_,[X|_], L).
```



## Operaciones sobre listas

- **Añadir** un elemento a una lista

- Si deseamos poner el elemento X al comienzo de la lista L:

```
[X | L ]
```

- Es decir, no se necesita definir ningún procedimiento para ello. En cualquier caso, éste podría definirse como el hecho:

```
añadir(X, L, [X|L]).
```

- **Borrar** un elemento de una lista

- Se elimina un elemento X de una lista L, devolviendo la lista L1.

```
del(X, L, L1)
```

- **Definición.** Es necesario contemplar dos casos:

- Si X es la cabecera de la lista, el resultado debe ser la cola de la lista
- Si X está en la cola, tiene que ser eliminada recursivamente.

### EJECUCIÓN

```
?- del(a, [a,b,c,a], L1).
```

```
    L1 = [b, c, a] ;
```

```
    L1 = [a, b, c] ;
```

```
del(X, [X|Tail], Tail).
```

```
del(X, [Y|Tail], [Y|Tail1]) :-  
    del(X, Tail, Tail1).
```

## Operaciones sobre listas

- **Borrar** un elemento de una lista
  - Esta relación se denomina `select` en la implementación SWI-Prolog.
  - Se produce un fallo si la lista no contiene el elemento a borrar.
  - Se puede usar para realizar la operación inversa, o sea, añadir un elemento a una lista en cualquier posición.

```
?- del(a, L, [1,2,3]).  
L = [a, 1, 2, 3] ;  
L = [1, a, 2, 3] ;  
L = [1, 2, a, 3] ;  
L = [1, 2, 3, a] ;
```

- La operación anterior, se puede definir mediante la cláusula:

```
insert(X, Lista, ListaMayor) :-  
    del(X, ListaMayor, Lista).
```

- También podemos redefinir la operación de miembro, usando `del`

```
miembro2(X, Lista) :-  
    del(X, Lista, _)
```

## Operaciones sobre listas

- **Sublista** de una lista

- La relación de sublista indica si una lista está contenida en otra.

```
sublist([c,d,e], [a,b,c,d,e,f]).    Yes
sublist([c,e], [a,b,c,d,e,f]).    No
```

- Definición: S es una sublista de L si:
  - L puede descomponerse en dos listas L1 y L2 y
  - L2 puede descomponerse en dos listas S y L3

```
sublist(S,L):-
    conc(L1,L2,L),
    conc(S,L3,L2).
```

- Puede usarse para encontrar todas las sublistas de una dada

```
?- sublist(S,[a,b,c]).

S = [] ;
S = [a] ;
S = [a, b] ;
S = [a, b, c] ;
S = [] ;
S = [b] ;
...
```

## Operaciones sobre listas

- **Permutaciones** de una lista

- Permite obtener una nueva lista generada mediante permutaciones de los elementos de la otra.

```
?- permutation([a,b,c],L).  
  
L = [a, b, c] ;  
L = [b, a, c] ;  
L = [b, c, a] ;  
...
```

- Definición:

- Si la primera lista está vacía, la segunda también debe estarlo.
- Si la primera lista es no vacía, será de la forma  $[X|L]$ , y la permutación se debe obtener permutando primero  $L$  para obtener  $L1$  y luego insertar  $X$  en cualquier posición de  $L1$ .

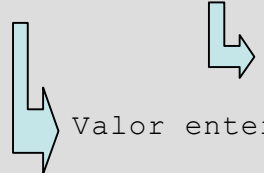
```
permutation([], []).  
  
permutation([X|L],P):-  
    permutation(L,L1),  
    insert(X,L1,P).
```

- Esta relación se denomina `permutation` en la implementación SWI-Prolog.

## Operadores

- Las expresiones aritméticas se tratan como objetos estructurados, en el que los operadores son los funtores del término.
- Los operadores pueden definirse por el programador, usando la relación `op`

```
op(precedencia, tipo, nombre)
```



Pre, post, infijo

Valor entero 1..1200

- En cualquier caso, las diversas implementaciones de Prolog proporcionan gran cantidad de operadores predefinidos.

## Operadores aritméticos

- Los operadores más habitualmente definidos en Prolog son:
  - Aritméticos : +, \*, -, /, mod, //, max, min, \*\*, sqrt, log, log10
  - Comparación: ==, /=, >, <, <=, >=
  - Lógicas: not, V, ^, xor
  - Trigonómicas: sin, cos, tan, ...
- Operadores aritméticos:

mod	<i>Módulo</i>
//	<i>División entera</i>
max	<i>Máximo</i>
min	<i>Mínimo</i>
**	<i>Exponenciación</i>
sqrt	<i>Raíz cuadrada</i>
log	<i>Logaritmo neperiano</i>

- Una expresión se evalúa si usamos el operador `is`:

<code>?- X=1-2.</code>	<i>Resultado X = 1-2</i>
<code>?- X is 1-2.</code>	<i>Resultado X = -1</i>

## Operadores aritméticos

- Proporciona funciones trigonométricas: sin, cos, tan, asin, acos, atan.
- Operadores aritméticos de comparación de valores:

<code>X &gt; Y</code>	<i>Mayor que</i>
<code>X &lt; Y</code>	<i>Menor que</i>
<code>X &gt;= Y</code>	<i>Mayor o igual</i>
<code>X &lt;= Y</code>	<i>Menor o igual</i>
<code>X == Y</code>	<i>Igual</i>
<code>X /= Y</code>	<i>Distinto</i>

- Diferencia entre `=` y `==`

```
?- 1+2=2+1.  
No  
  
?- 1+2==2+1.  
Yes  
  
?- 1+X=B+2.  
X = 2  
B = 1
```

## Operadores aritméticos

- Algunos ejemplos:

- Sea D el máximo común divisor (mcd) de X e Y. Para su cálculo pueden darse tres casos:

- Si X e Y son iguales entonces D es igual a X
- Si  $X < Y$  entonces D es igual al mcd de X y la diferencia  $X - Y$
- Si  $Y > X$ , idem. que en el caso anterior, intercambiando X e Y

```
mcd(X,X,X) .  
  
mcd(X,Y,D) :-  
    X < Y,  
    Y1 is Y-X,  
    mcd(X,Y1,D) .  
  
mcd(X,Y,D) :-  
    X > Y,  
    mcd(Y,X,D) .
```

- Cálculo de la longitud de una lista (en SWI-Prolog está predefinida la función length).

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



## Ejercicios propuestos

1. Escribe un objetivo de Prolog, utilizando `conc` (`append`) , para borrar los últimos tres elementos de la lista `L`, produciendo una nueva lista `L1`.
2. Escribe un objetivo para borrar los tres primeros elementos y los tres últimos de una lista `L`, produciendo la lista `L2`.
3. Define la relación `last(Item, Lista)` , de tal modo que `Item` sea el último elemento de la lista `List`. Se piden dos versiones, una usando `conc` y la otra sin usarlo.
4. Define dos predicados `listapar(Lista)` y `listaimpar(Lista)` , que sean verdaderos cuando una lista tenga un número par o impar de elementos, respectivamente.
5. Define la relación `revese(Lista, ListaInvertida)` que invierte una lista.
6. Define el predicado `palindromo(Lista)` . Una lista es un palíndromo si se lee igual en ambas direcciones, por ejemplo: `[m,a,d,a,m]` .
7. Define la relación `maximo(X,Y,Max)`, en la que máximo es el valor más alto de `X` e `Y`.
8. Define el predicado `sumarlista(Lista, Suma)`, que obtiene la suma de los elementos numéricos de la lista.