

Tema 2: Listas, aritmética y operadores

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

Representación de listas

- Símbolos:
 - Una constante: $[]$
 - Un símbolo de función binario: $.$
- Ejemplos de listas (notación de términos)
 - $[]$
 - $.(a, [])$
 - $.(1, .(2, .(3, .(4, []))))$
 - $.([], [])$
- Ejemplos de listas (notación reducida)
 - $[]$
 - $[a]$
 - $[1, 2, 3, 4]$
 - $[[]]$

Representación de listas

- El predicado `display`

```
?- display([]).  
[]  
Yes  
?- display([a]).  
. (a, [])  
Yes  
?- display([1,2,3,4]).  
. (1, . (2, . (3, . (4, []))))  
Yes  
?- display([[[]]).  
. ([], [])  
Yes
```

- El predicado de unificación: `=`

```
?- X = . (a, . (1, [])).  
X = [a, 1]  
Yes  
?- . (X,Y) = [1].  
X = 1  
Y = []  
Yes  
?- . (X,Y) = [a,b,c].  
X = a  
Y = [b, c]  
Yes  
?- . (X, . (2, . (Z, []))) = [1,Y,3].  
X = 1  
Z = 3  
Y = 2  
Yes
```

Unificación con listas

- ¿Son unificables las siguientes listas?

- $[X|Y]$ y $[1,2,3]$

?- $[X|Y] = [1,2,3]$.

$X = 1$

$Y = [2, 3]$

Yes

- $[X,Y|Z]$ y $[1,2,3]$

?- $[X,Y|Z] = [1,2,3]$.

$X = 1$

$Y = 2$

$Z = [3]$

Yes

- $[X,Y]$ y $[1,2,3]$

?- $[X,Y] = [1,2,3]$.

No

- $[X,Y|Z]$ y $[1,2]$

?- $[X,Y|Z] = [1,2]$.

$X = 1$

$Y = 2$

$Z = []$

Yes

Unificación con listas

- $[X,a,b|[]]$ y $[[b,L],a|L]$
?- $[X,a,b|[]] = [[b,L],a|L]$.
 $X = [b, [b]]$
 $L = [b]$
Yes
- $[X,Y,Z]$ y $[Y,Z,X]$
?- $[X,Y,Z] = [Y,Z,X]$.
 $X = _G171$
 $Y = _G171$
 $Z = _G171$
Yes
- $[X,Y,Z]$ y $[Y,Z|[]]$
?- $[X,Y,Z] = [Y,Z|[]]$.
No
- $p([X|R],h(X,[a|[R|L]]))$ y $p([a],h(a,[L]))$
?- $p([X|R],h(X,[a|[R|L]])) = p([a],h(a,[L]))$.
No
- X y $f(X)$
?- $X = f(X)$.
Action (h for help) ? a
abort
Execution Aborted

Operaciones con listas

- **Problema: Primer elemento y resto de una lista**

- `primero(L,X)` se verifica si X es el primer elemento de la lista L
- `resto(L,X)` se verifica si X es el resto de la lista L

- **Ejemplo**

```
primero([a,b,c],X) => X=a
resto([a,b,c],X) => X=[b,c]
```

- **Programa listas-1.pl**

```
primero([X|L],X).
resto([X|L],L).
```

- **Sesión**

```
?- primero([a,b,c],X).
X = a
Yes
?- primero([X,b,c],a).
X = a
Yes
?- primero([X,Y],a).
X = a
Y = _G286
Yes
?- primero(X,a).
X = [a|_G353]
?- resto([a,b,c],L).
L = [b, c]
?- resto([a|L],[b,c]).
L = [b, c]
```

Operaciones con listas

- **Añadir un elemento a una lista**

- $\text{cons}(X, L1, L2)$ se verifica si $L2$ es la lista obtenida añadiéndole X , como primer elemento, a la lista $L1$

- **Ejemplo**

$\text{cons}(a, [b, c], L2) \Rightarrow L2 = [a, b, c]$

- **Programa cons.pl**

$\text{cons}(X, L1, [X|L1]).$

- **Sesión**

?- $\text{cons}(a, [b, c], L).$

$L = [a, b, c]$

?- $\text{cons}(X, L, [a, b, c]).$

$X = a$

$L = [b, c] ;$

Operaciones con listas

- **Concatenación de listas**

- `conc(L1,L2,L3)` se verifica si `L3` es la lista obtenida escribiendo los elementos de `L2` a continuación de los elementos de `L1`.

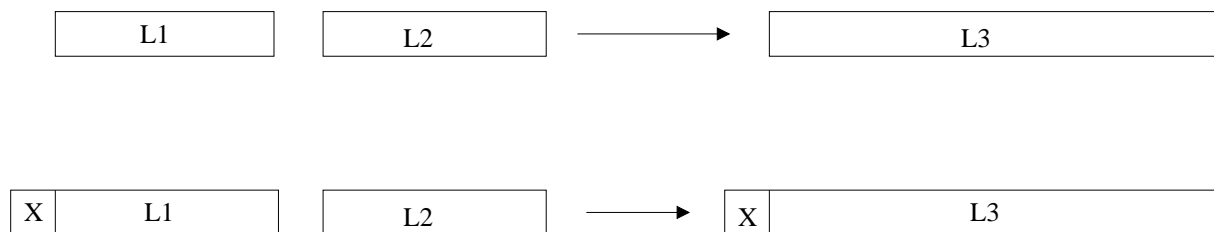
- **Ejemplo**

`conc([a,b],[c,d],L3) => L3 = [a,b,c,d]`

- **Programa `conc.pl`**

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

- **Esquema**



Operaciones con listas

- ¿Cuál es el resultado de concatenar las listas $[a,b]$ y $[c,d,e]$?

?- conc($[a,b]$, $[c,d,e]$, L).

L = $[a, b, c, d, e]$;

No

- ¿Qué lista hay que añadirle al lista $[a,b]$ para obtener $[a,b,c,d]$?

?- conc($[a,b]$, L, $[a,b,c,d]$).

L = $[c, d]$;

No

- ¿Qué dos listas hay que concatenar para obtener $[a,b]$?

?- conc(L1, L2, $[a,b]$).

L1 = $[]$

L2 = $[a, b]$;

L1 = $[a]$

L2 = $[b]$;

L1 = $[a, b]$

L2 = $[]$;

No

Operaciones con listas

- ¿Pertenece b a la lista [a,b,c]?

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

L1 = [a]

L2 = [c] ;

No

?- conc(_, [b|_], [a,b,c]).

Yes

- ¿Es [b,c] una sublista de [a,b,c,d]?

?- conc(_, [b,c|_], [a,b,c,d]).

Yes

- ¿Es [b,d] una sublista de [a,b,c,d]?

?- conc(_, [b,d|_], [a,b,c,d]).

No

- ¿Cuál es el último elemento de [b,a,c,d]?

?- conc(_, [X], [b,a,c,d]).

X = d ;

No

- Predicado predefinido: append(L1,L2,L3)

Aritmética

- Operadores prefijos e infijos

- $(a + b) * (5 / c)$

?- display((a + b) * (5 / c)).

*(+(a, b), /(5, c))

Yes

- $a + b * 5 / c$

?- display(a + b * 5 / c).

+(a, /(*(b, 5), c))

Yes

- Precedencia y tipo de operadores predefinidos:

| Precedencia | Tipo | Operadores |
|-------------|------|------------|
| 500 | yfx | +, - |
| 500 | fx | - |
| 400 | yfx | *, / |
| 200 | xfy | ^ |

- fx, fy: Prefijo
- xfx: Infijo no asociativo
- yfx: Infijo asocia por la izquierda
- xfy: Infijo asocia por la derecha
- xf, yf: Postfijo

Aritmética

- **Análisis de expresiones con operadores**

```
?- display(2+3+4).  
+(+(2, 3), 4)  
Yes
```

```
?- display(2+3*4).  
+(2, *(3, 4))  
Yes
```

```
?- display((2+3)*4).  
*(+(2, 3), 4)  
Yes
```

```
?- display(2^3^4).  
^(2, ^(3, 4))  
Yes
```

Predicados aritméticos

- Evaluador: `is`
- Predicados aritméticos:

| Precedencia | Tipo | Operadores |
|-------------|------|-----------------------|
| 700 | xfx | <, =<, >, >=, :=, =\= |

- Ejemplos

?- X is 2+3^3.

X = 29

Yes

?- 29 is X+3^3.

[WARNING: Arguments are not sufficiently instantiated]

?- X = 2+3^3.

X = 2+3^3

Yes

?- 2+3^Y = 2+3^3.

Y = 3

Yes

?- 3 =< 5.

Yes

?- 3 > X.

[WARNING: Arguments are not sufficiently instantiated]

?- 2+5 = 10-3.

No

?- 2+5 := 10-3.

Yes

?- 2+5 =\= 10-3.

No

?- 2+5 =\= 10^3.

Yes

Máximo

- Máximo de dos números

- `maximo(X,Y,Z)` se verifica si Z es el máximo de los números X e Y.

- Ejemplo

`maximo(3,5,Z) => Z=5`

- Programa: `maximo.pl`

```
maximo(X,Y,X) :-  
    X >= Y.  
maximo(X,Y,Y) :-  
    X < Y.
```

- Sesión

```
?- maximo(2,3,X).  
X = 3 ;  
No  
?- maximo(3,2,X).  
X = 3 ;  
No
```

Factorial

- Factorial de un número

- `factorial(X,Y)` se verifica si Y es el factorial de X

- Programa: `factorial.pl`

```
factorial(1,1).  
factorial(X,Y) :-  
    X > 1,  
    X1 is X - 1,  
    factorial(X1,Y1),  
    Y is X * Y1.
```

- Sesión

```
?- factorial(4,Y).  
Y = 24  
Yes
```

- Cálculo de `factorial(4,Y)`

```
X = 4  
X1 is X-1 => X1 = 3  
factorial(X1,Y1) => Y1 = 6  
Y is X*Y1 => Y = 24
```

Fibonacci

- **Sucesión de Fibonacci**

- La sucesión de Fibonacci es

0, 1, 1, 2, 3, 5, 8, ...

y está definida por

$$f(0) = 0$$

$$f(1) = 1$$

$$f(n) = f(n-1)+f(n-2), \quad \text{si } n > 1$$

- `fibonacci(N,X)` se verifica si `X` es el `N`-ésimo término de la sucesión de Fibonacci.

- **Programa:** `fibonacci.pl`

```
fibonacci(0,0).
fibonacci(1,1).
fibonacci(N,X) :-
    N > 1,
    N1 is N-1,
    fibonacci(N1,X1),
    N2 is N-2,
    fibonacci(N2,X2),
    X is X1+X2.
```

- **Sesión**

```
?- fibonacci(6,X).
X = 8
Yes
```


Longitud

- Longitud de una lista

- `longitud(L,N)` se verifica si `N` es la longitud de la lista `L`

- Ejemplos

```
longitud([],N)           => N = 0
longitud([a,b,c],N)     => N = 3
longitud([a,[b,c]],N)   => N = 2
```

- Programa: `longitud.pl`

```
longitud([],0).
longitud([_X|L],N) :-
    N is M + 1.
```

- Sesión

```
?- longitud([a,b,c],N).
N = 3
Yes
```

- Predicado predefinido: `length(L,N)`

Máximo de una lista

- Máximo de una lista

- `max_list(L,N)` se verifica si `N` es el máximo de los elementos de la lista de números `L`

- Sesión

```
?- max_list([1,3,9,5],X).
```

```
X = 9
```

```
Yes
```

- Programa: `max_list.pl`

```
max_list([X],X).  
max_list([X,Y|L],Z) :-  
    max_list([Y|L],U),  
    maximo(X,U,Z).  
maximo(X,Y,X) :-  
    X >= Y.  
maximo(X,Y,Y) :-  
    X < Y.
```

Entre

- **Intervalo entero**

- `entre(N1,N2,X)` que se verifica si X es mayor o igual que $N1$ y menor o igual que $N2$.

- **Sesión**

?- `entre(2,5,X)` .

$X = 2$;

$X = 3$;

$X = 4$;

$X = 5$;

No

?- `entre(2,1,X)` .

No

- **Programa: `entre.pl`**

```
entre(N1,N2,N1) :-
```

```
    N1 =< N2.
```

```
entre(N1,N2,X) :-
```

```
    N1 < N2,
```

```
    N3 is N1 + 1,
```

```
    entre(N3,N2,X).
```

- **Predicado predefinido: `between(N1,N2,X)`**

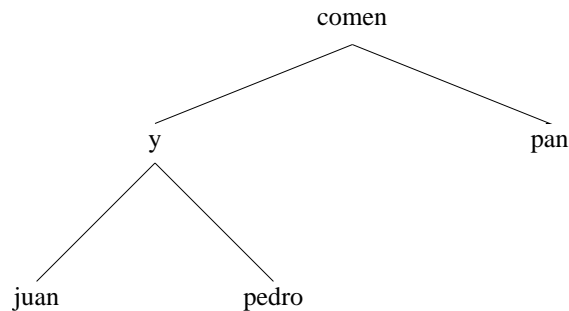
Operadores

- Operadores definidos por el usuario
 - `pedro come pan ==> come(pedro,pan)`
 - `juan tiene discos ==> tiene(juan,discos)`
- Definición de operadores (directiva `op`)
 - `:- op(600,xfx,come).`
- Precedencia: Entre 1 y 1200

Operadores

- Estructura de arbol

Juan y Pedro comen pan



- Sin operadores:

```
comen(y(juan,pedro),pan).
```

- Con operadores:

```
:-op(800,xfx,comen).
```

```
:-op(400,xfx,y).
```

```
juan y pedro comen pan.
```

```
?- display(juan y pedro comen pan).
```

```
comen(y(juan, pedro), pan)
```

```
Yes
```

```
?- Quienes comen pan.
```

```
Quienes = juan y pedro ;
```

```
No
```

```
?- Alguien y pedro comen pan.
```

```
Alguien = juan ;
```

```
No
```

```
?- juan y pedro comen Algo.
```

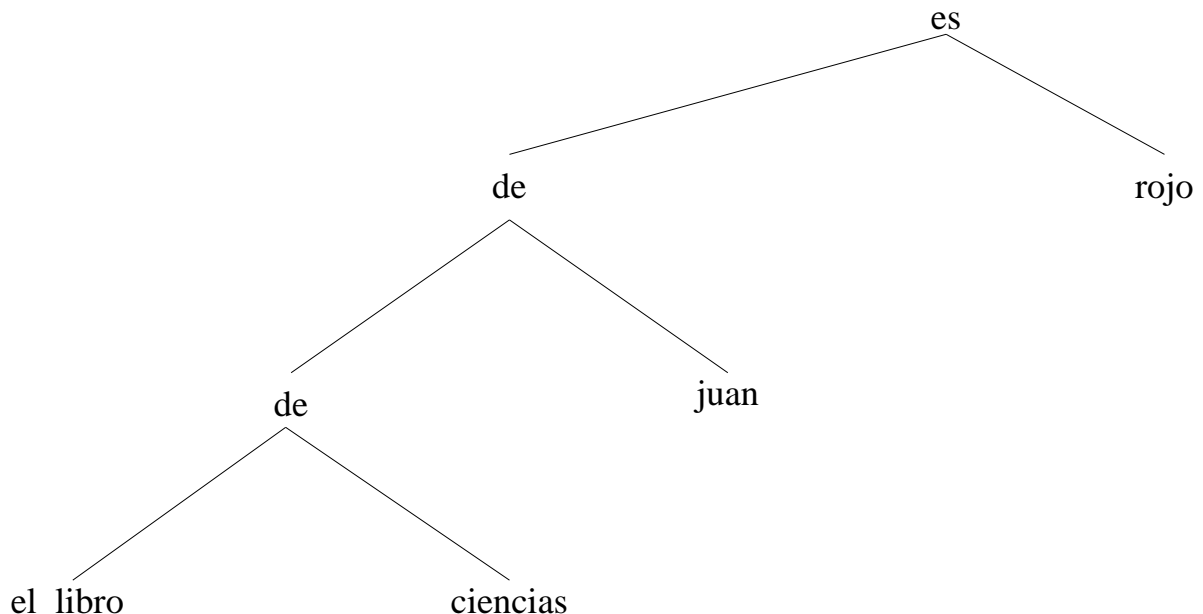
```
Algo = pan ;
```

```
No
```

Operadores

- Precedencia entre argumentos
 - La precedencia de un término simple es cero
 - La precedencia de un término compuesto es la del símbolo de función principal y cero si no está predefinido
- x e y
 - x representa un argumento cuya precedencia es estrictamente menor que la del operador
 - y representa un argumento cuya precedencia es menor o igual que la del operador
- Ejemplo:

El libro de ciencias de Juan es rojo



Operadores

- Programa

```
:-op(800,xfx,es).
:-op(400,yfx,de).
el_libro de ciencias de juan es rojo.

?- display(el_libro de ciencias de juan es rojo).
es(de(de(el_libro, ciencias), juan), rojo)
Yes
?- display(X es rojo).
es(_G177, rojo)
X = _G177
Yes
?- X es rojo.
X = el_libro de ciencias de juan
Yes
?- display(X de Y es rojo).
es(de(X, Y), rojo)
Yes
?- display(el_libro de ciencias de juan es rojo).
es(de(de(el_libro, ciencias), juan), rojo)
Yes
?- X de Y es rojo.
X = el_libro de ciencias
Y = juan
Yes
?- display(el_libro de X es rojo).
es(de(el_libro, X), rojo)
Yes
?- el_libro de X es rojo.
No
```

Bibliografía

- Bratko, I. *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
 - Cap. 3: “Lists, operators, arithmetic”
- Clocksin, W.F. y Mellish, C.S. *Programming in Prolog (Fourth Edition)* (Springer Verlag, 1994)
 - Cap. 2: “A closer look”