Prolog: Listas

- Estructura básica
- Unificación
- Member
- Append
- Ejercicio
- Árbol de seguimiento

Estructura básica

- En prolog, una lista es una representación de un conjunto de elementos.
- La notación es la siguiente:

  [manzana, pera, banana] lista vacia: [ ]
- Se pueden utilizar cómo elementos de la lista cualquier tipo de dato de prolog, incluyendo listas:

  [[a,b,c],[d, e, f]]
- También estructuras prolog:

  [camino(tandil, bsas), camino(mardel, tandil), camino(bsas,junin)]

  [vehiculo(ale, [bici, moto, auto]), vehiculo(ariel,[bici, auto, helicoptero])]

Estructura básica

- En su forma más básica, una lista se puede ver como un predicado que tiene 2 partes:

  lista(cabeza, cola)

  en prolog:

  [Cabeza | Cola]

- Por ejemplo:

  [Cabeza | Cola]=[1,2,3]

  Cabeza=1,

  Cola=[2,3]

  [1] = [1|[ ]]

- Es una definición recursiva...

  [1,2,3] = [1 | [2 , 3]] = [1 | [2 | [3]]] = [1 | [2 | [3 | [ ]]]]

- Para probar:

  ?-X=[1,2,3],X = [1 | [2 , 3]],

  X = [1 | [2 | [3]]],X = [1 | [2 | [3 | [ ]]]] .

Unificación

- Ya vimos algunos ejemplos, pero hay mas:

[a,b,c]=[X,Y,Z]
X=a, Y=b, Z=c

[a,b,c] = [A, B, C|D]
A=a, B=b, C=c, D=[]

[a]=[A|B]
A=a, B=[]

[a,b,c,d,e] = [X,Y| Z]
X=a, Y=b, Z=[c,d,e]

[a,b,c] = [X,[Y|Z]]
No, espera que el segundo elemento de la lista sea una lista.

[]=[X|Y] o [] = [X]
No!, la lista vacia no se puede dividir en cabeza y cola
Es útil cuando definimos predicados de corte, para estar seguros que una lista vacia no hace match con una división cabeza-cola.
Por ej:

```
sumar([],R):-R=0.                Mejor: sumar([],0).
sumar([ X | Y ],R) :- sumar(Y, R1), R is R1+X.
```

Member

- Permite saber si un elemento pertenece a una lista:
  member(1,[5,6,7,8,1,2,3]).
  yes
  member(1,[a,b,c])
  no
- Si el primer argumento es una variable, podemos listar los elementos de una lista uno a uno, haciendo redo:
  member(X,[a,b,c]).
  yes, X=a;
  yes, X=b;
  yes, X=c;
  No

Append

- Permite unificar 2 listas en una:
  append([1,2],[3,4],X)
  X=[1,2,3,4]

  o verificar que una lista es la unión de otras 2:

  append([a,b],[c],[a,b,c])
  yes
- Es más útil con los argumentos sin instanciar:
  append(X,[3,4],[1,2,3,4]).
  Yes, X=[1,2]
  append([1,2],Y,[1,2,3,4])
  Yes, Y=[3,4]

Append

- Permite conseguir todas las posibilidades de partir una lista en 2:
  append(X,Y,[1,2,3,4])
  X=[], Y=[1,2,3,4] ;
  X=[1], Y=[2,3,4];
  …
  X=[1,2,3,4], Y=[]
- En algunos casos nos sirve para agregar nuevos elementos a una lista que vamos acarreando:
  agregar(X, L, Lnueva):- append([X], L, Lnueva).
  Que en realida también se podría hacer con unificación:
  agregar(X, L, Lnueva):- Lnueva=[X|L].
  pero como vimos, es lo mismo:
  agregar(X, L, [X|L]).

Ejercicio

- Tamaño de una lista:

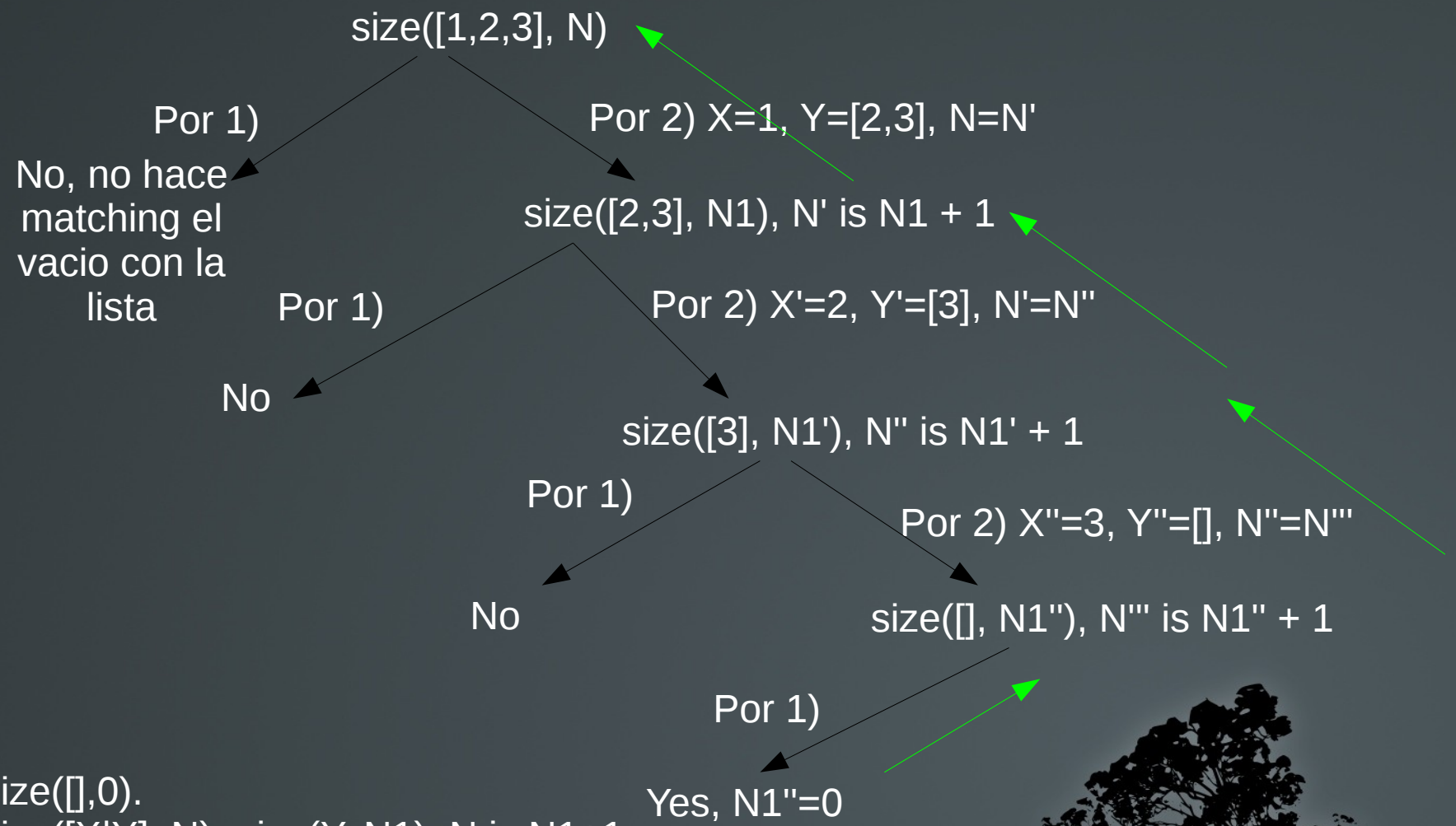
```
size([a,b,c], N).
yes, N=3.

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

- Se puede dejar sin instanciar el 1er argumento (Ejercicio 3.a).

Ejercicio

- Rotar una lista:
  Por ej:
  ?-rotar([1,2,3,4], L, 2).
  L=[3,4,1,2].
- Primer enfoque:
  rotar(X,X,0).
  rotar([X|Y], L, N):-N1 is N-1, append(Y,[X],Y1), rotar(Y1, L, N1).
- Segundo enfoque:
  rotar(L,R, N):-append(X, Y, L), size(X, N), append(Y, X, R).

Arbol de seguimiento



1) size([],0).
2) size([X|Y], N):-size(Y, N1), N is N1+1.