

Ejercicio 1 ★

Considerar la siguiente base de conocimiento.

```
padre(juan, carlos).
padre(juan, luis).
padre(carlos, daniel).
padre(carlos, diego).
padre(luis, pablo).
padre(luis, manuel).
padre(luis, ramiro).
abuelo(X, Y) :- padre(X, Z), padre(Z, Y).
```

- I. ¿Cuál el resultado de la consulta `abuelo(X, manuel)`?
 - II. A partir del predicado binario `padre`, definir en Prolog los predicados binarios: `hijo`, `hermano` y `descendiente`.
 - III. Dibujar el árbol de búsqueda de Prolog para la consulta `descendiente(Alguien, juan)`.
 - IV. ¿Qué consulta habría que hacer para encontrar a los nietos de juan?
 - V. ¿Cómo se puede definir una consulta para conocer a todos los hermanos de `pablo`?
 - VI. Considerar el agregado del siguiente hecho y regla:
- ```
ancestro(X, X).
ancestro(X, Y) :- ancestro(Z, Y), padre(X, Z).
```
- y la base de conocimiento del ítem anterior.
- VII. Explicar la respuesta a la consulta `ancestro(juan, X)`. ¿Qué sucede si se pide más de un resultado?
- VIII. Sugerir una solución al problema hallado en los puntos anteriores reescribiendo el programa de `ancestro`.

Preguntas Prolog:

- ejercicio 1, árbol.
- tema de que cuando pasa que matchea con dos ecuaciones? ¿Tiene sentido eso? Digo por ejemplo que haya encontrado todas las unificaciones para la primera ecuación y ahora vaya a la segunda.
- desde(X, X) ¿qué pasa si ambas están instanciados, devuelve false o no matchea?
- ¿Cuando algo puede dar false porque no encuentra más soluciones? ¿Cuando algo se cuelga indefinidamente?
- ¿Cuando un resultado da true, y sigo pidiendo opciones evalúa si otra ecuación vale para el mismo input? ¿En qué caso "si no vale para la primera" (false) sigue buscando en la segunda?

16:38 ✓

A) PROLOG Busca en CADA PREDICADO de la base de conocimiento.

`ABUELO(X, MANUEL)` buscará dentro de él algún `hecho` que se le llame así. No lo hay.  
Pero sí hay una regla.

`ABUELO(X, Y) :- PADRE(X, Z), PADRE(Z, Y).`  $\equiv \forall X \cdot \forall Y \cdot \forall Z \cdot \text{PADRE}(X, Z) \wedge \text{PADRE}(Z, Y) \Rightarrow \text{ABUELO}(X, Y)$

Como me interesa cumplir el `ABUELO(X, MANUEL)` busco en el PREDICADO `ABUELO`, y tomo el valor de `MANUEL` y busco en `X` que UNIFIQUE y cumpla `PADRE(X, Z), PADRE(Z, MANUEL)`.

¿Qué nos dice  $\forall X \cdot \forall Z \cdot (\text{PADRE}(X, Z) \wedge \text{PADRE}(Z, MANUEL) \Rightarrow \text{ABUELO}(X, MANUEL))$ ?

Que si encontramos un `Z` hijo de un `X`, y a su vez `Z` es `PADRE` de `MANUEL` con `X` en el abuelo de `MANUEL`.

Entonces el úniclo `ABUELO(X, MANUEL) :- PADRE(X, Z), PADRE(Z, Y)`

1      2

Con esto haces DFS, elegirás las opciones para 1 y luego para 2.

En 1 no hay restricciones, por lo tanto puedes tomar los hijos de `PADRE` donde el primer argumento verifica con `X` y el segundo con `Z`.

- |                                                                                                                          |                                        |
|--------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| - $X = \text{JUAN}, Z = \text{CARLOS} \rightsquigarrow \text{PADRE}(\text{JUAN}, \text{CARLOS})$                         | - $X = \text{Luis}, Z = \text{PABLO}$  |
| - $X = \text{JUAN}, Z = \text{Luis} \rightsquigarrow \text{PADRE}(\text{JUAN}, \text{Luis})$<br>MAS O FALLO ANT.         | - $X = \text{Luis}, Z = \text{MANUEL}$ |
| - $X = \text{CARLOS}, Z = \text{DANIEL} \rightsquigarrow \text{PADRE}(\text{CARLOS}, \text{DANIEL})$<br>MAS O FALLO ANT. | - $X = \text{Luis}, Z = \text{MIRIAM}$ |
| - $X = \text{CARLOS}, Z = \text{DIEGO}$                                                                                  |                                        |

En 2 SÍ tienen restricciones, por cosa de unificación. En la primera condición tomas los mismos valores para las reglas.

- `PADRE(X, Z) :- PADRE(Y, Z), PADRE(X, Y)`  $\rightsquigarrow \text{PADRE}(\text{JUAN}, \text{CARLOS}), \text{PADRE}(\text{JUAN}, \text{Luis})$

PADRE(JUAN, LUIS), PADRE(CARLOS, MANUEL)  $\rightsquigarrow$  AISA. BACKTRACK y ENCUENTRA  $x = \text{JUAN}$ ,  $y = \text{LUIS}$ .

- PADRE(JUAN, LUIS), PADRE(LUIS, MANUEL) ✓

:

Oír con todos.

RTA: JUAN es el ABUELO de MANUEL.

b) HIJO(X,Y) :- PADRE(Y,X).

" $X$  es HIJO de  $Y$  si  $Y$  es MI PADRE de  $X$ "

HERMANO(X,Y) :- HIJO(X,Z), HIJO(Y,Z),  $X \neq Y$

hermano de  $X$  son los que tienen al mismo.

g

" $X$  es HERMANO de  $Y$  si  $Y$  es MI hermano de  $Z$  tal que  $\text{HIJO}(X,Z)$ ,  $\text{HIJO}(Y,Z)$ ,  $X \neq Y$ "

DESCENDIENTE(X,Y) :-

DIRECTO

" $X$  es DESCENDIENTE de  $Y$  si PADRE(X,Y)",

$X$  es DESCENDIENTE de  $Y$  si existe algún  $Z$  que sea PADRE de  $X$  y a su vez sea  $Z$  es HIJO de  $Y$ ".

PADRE(MARIO, TOMAS)

$\text{LEO} \rightarrow \text{NICO} \rightarrow \text{MARIO} \rightarrow \text{TOMAS}$

PADRE(NICO, MARIO)

PADRE(LEO, NICO)

DESCENDIENTE(TOMAS, MARIO) ✓ Por problema.

DESCENDIENTE(TOMAS, NICO) :- PADRE(Z, NICO)

$\downarrow$   
Dels descendentes de  $Z$ .

DESCENDIENTE(X,Y) :- PADRE(Y,X).

DESCENDIENTE(X,Y) :- DESCENDIENTE(X,Z), PADRE(Z,Y)

F  
A  
L  
L  
A

? DESCENDIENTE(TOMAS, NICO)

DESCENDIENTE(TOMAS, NICO) :- PADRE(NICO, TOMAS).  $X \rightarrow$  lo es her.

DESCENDIENTE(TOMAS, NICO) :- DESCENDIENTE(TOMAS, Z), PADRE(Z, NICO)

1. DESCENDIENTE(TOMAS, Z)

a) DESCENDIENTE(TOMAS, Y) :- PADRE(Y, TOMAS)  $\rightarrow Y = \text{MARIO} X = \text{TOMAS}$

b) DESCENDIENTE(TOMAS, Y) :- DESCENDIENTE(TOMAS, Z), PADRE(Z, Y)  $\rightsquigarrow$  F. Vuelve a a para analizar b.

b) DESCENDIENTE(TOMAS, Y) :- DESCENDIENTE(TOMAS, Z), PADRE(Z, Y)

$\hookrightarrow$  ¿Por qué no? Ni le dice mal: 1. Es el her.

$\text{DESCENDIENTE}(x, y) :- \text{PADRE}(y, x).$

$\text{DESCENDIENTE}(x, y) :- \text{PADRE}(z, x), \text{DESCENDIENTE}(z, y).$

$\text{PADRE}(\text{mario}, \text{romas}), (\text{PADRE}(\text{mario}, \text{nico})) /$

?  $\text{DESCENDIENTE}(\text{romas}, \text{nico})$

$\text{DESCENDIENTE}(\text{romas}, \text{nico}) :- \text{PADRE}(\text{nico}, \text{romas}). \quad x \rightarrow \text{no es fiel}$

$\text{DESCENDIENTE}(\text{romas}, \text{nico}) :- \text{PADRE}(z, \text{romas}), \text{DESCENDIENTE}(z, \text{nico})$

1.  $\text{PADRE}(z, \text{romas}) \rightarrow z = \text{mario}$

2.  $\text{DESCENDIENTE}(\text{mario}, \text{nico})$

$\hookrightarrow \text{PADRE}(\text{nico}, \text{mario}) \checkmark$

RJA: TNE.

$\text{DESCENDIENTE}(x, y) :- \text{PADRE}(y, x).$

$\text{DESCENDIENTE}(x, y) :- \text{PADRE}(z, x), \text{DESCENDIENTE}(z, y).$

C)  $\text{DESCENDIENTE}(x, \text{juan})$

$\hookrightarrow \text{PADRE}(\text{juan}, x) \rightarrow x = \text{carlos}, x = \text{luis}$  ✓  
 $\hookrightarrow \text{PADRE}(z, x), \text{DESCENDIENTE}(z, \text{juan})$

$\hookrightarrow z = \text{juan}, x = \text{carlos} \rightsquigarrow \text{DESCENDIENTE}(\text{juan}, \text{juan})$

$\hookrightarrow \text{PADRE}(\text{juan}, \text{juan}) \times$

$\hookrightarrow \text{PADRE}(z, \text{juan}), \text{DESCENDIENTE}(z, y) \times$

$\hookrightarrow z = \text{juan}, x = \text{luis} \rightsquigarrow \text{DESCENDIENTE}(\text{juan}, \text{juan})$

$\hookrightarrow \text{PADRE}(\text{juan}, \text{juan}) \times$

$\hookrightarrow \text{PADRE}(z, \text{juan}), \text{DESCENDIENTE}(z, y) \times$

$\hookrightarrow z = \text{carlos}, x = \text{luis} \rightsquigarrow \text{DESCENDIENTE}(\text{carlos}, \text{juan})$

$\hookrightarrow \text{PADRE}(\text{juan}, \text{carlos}) \checkmark$

$\hookrightarrow \text{PADRE}(z, \text{carlos}), \dots \times$

VUELVE:  $x = \text{luis}$  ✓

$\hookrightarrow z = \text{luis}, x = \text{pablo} \rightsquigarrow \text{DESCENDIENTE}(\text{luis}, \text{juan})$

$\hookrightarrow \text{PADRE}(\text{juan}, \text{luis}) \checkmark$

VUELVE:  $x = \text{pablo}$

CON 1: si no tiene CONDA, CONDA: si tiene en la REC.

a) ABUELO(JUAN, X)

c) HERMANO(PABLO, Y)

$\hookrightarrow$  HIJO(PABLO, Z), HIJO(Y, Z), X ≠ Y

$\hookrightarrow$  PADRE(Z, PABLO)  $\rightarrow$  Z = LOUIS

$\hookrightarrow$  HIJO(Y, LOUIS)

$\hookrightarrow$  PADRE(LOUIS, Y)

$\hookrightarrow$  PABLO

$\rightarrow$  HIJO(PABLO, LOUIS), HIJO(PABLO, LOUIS), PABLO ≠ PABLO X

$\hookrightarrow$  PADRE(LOUIS, Y)

$\hookrightarrow$  MANUEL

$\rightarrow$  HIJO(PABLO, LOUIS), HIJO(MANUEL, LOUIS), PABLO ≠ MANUEL ✓  $\rightarrow$  Y = MANUEL

$\hookrightarrow$  PADRE(LOUIS, Y)

$\hookrightarrow$  MARINA

$\rightarrow$  HIJO(PABLO, LOUIS), HIJO(MARINA, LOUIS), PABLO ≠ MARINA ✓  $\rightarrow$  Y = MARINA

RJA: Y = MANUEL, Y = MARINA

f) ✓

$\sim$  Né que me cuelga pero me tiene hacer el órd.

R) ? ANCESTRO(JUAN, X)

ANCESTRO(X, X).

ANCESTRO(X, Y) :- ANCESTRO(Z, Y), PADRE(X, Z)

ANCESTRO(JUAN, JUAN)  $\rightarrow$  X = JUAN ✓

ANCESTRO(Z, Y), PADRE(JUAN, Z)

$\hookrightarrow$  ANCESTRO(Z, Z)  $\rightarrow$  Z = Y, PADRE(JUAN, Y)

$\hookrightarrow$  Y = CANCOS

#### Ejercicio 2

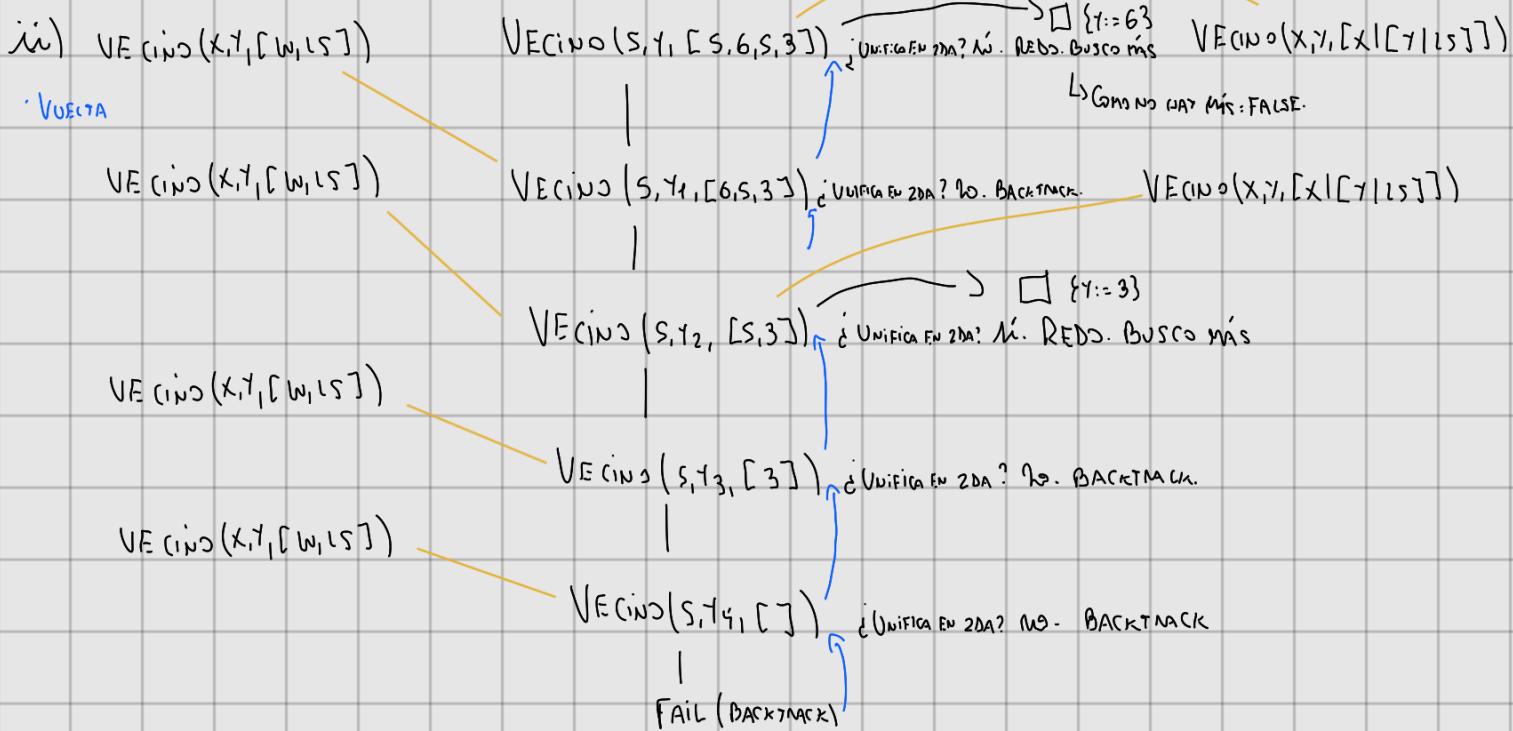
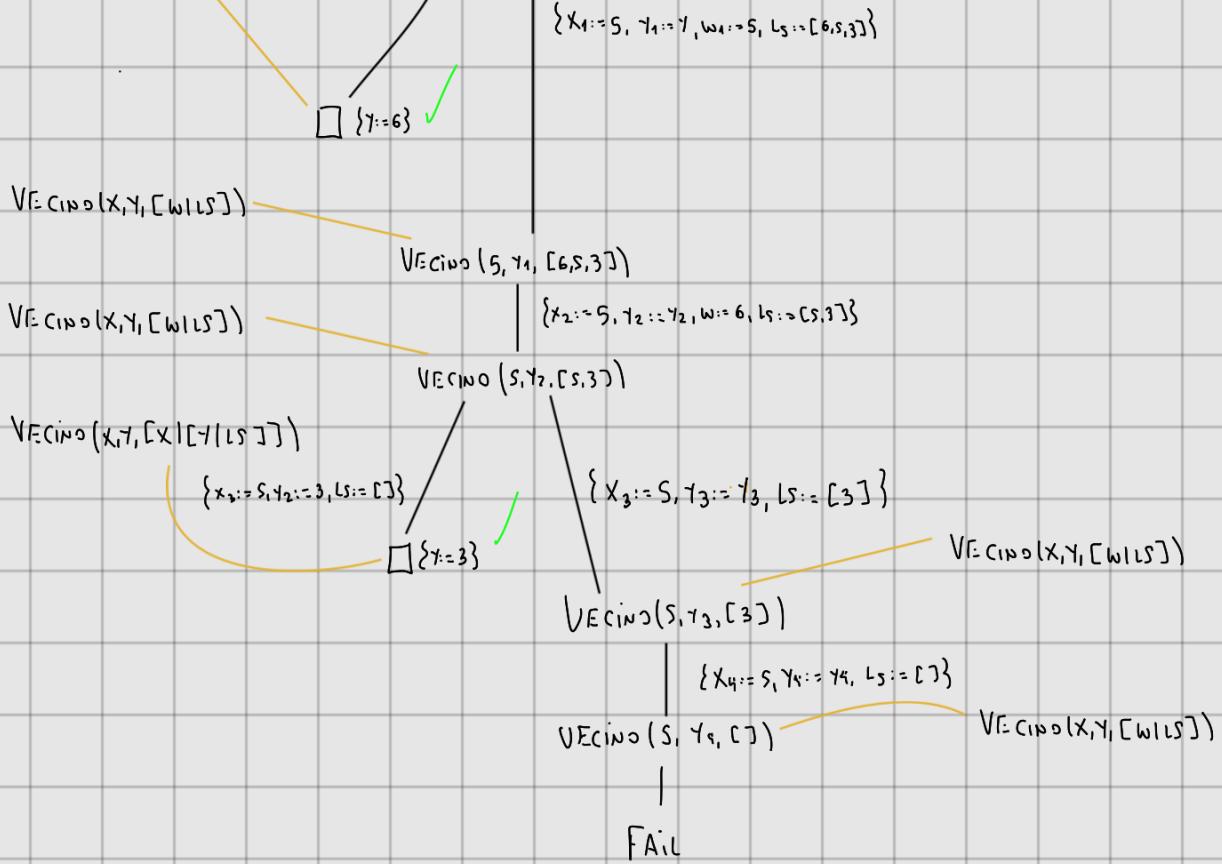
Sea el siguiente programa lógico: `vecino(X, Y, [X|Y|Ls]).`

`vecino(X, Y, [W|Ls]) :- vecino(X, Y, Ls).`

- i. Mostrar el árbol de búsqueda en Prolog para resolver `vecino(5, Y, [5, 6, 5, 3])`, devolviendo todos los valores de Y que hacen que la meta se deduzca lógicamente del programa.
- ii. Si se invierte el orden de las reglas, ¿los resultados son los mismos? ¿Y el orden de los resultados?

i) `VECINO(X, Y, [X|Y|Ls])`    `VECINO(S, Y, [S, 6, S, 3])`

$\{X_1=5, Y_1=6, Ls_1=[5, 3]\}$



Primer fue recursión hacia abajo. Faltó al final, y en base a esa decisión buscará si tiene unificación con la 2da.

Con 2da en base número visto y hace REDO (PENSA + RES).

Por recursión con los mismos pero en DIF orden.

#### Ejercicio 3 \*

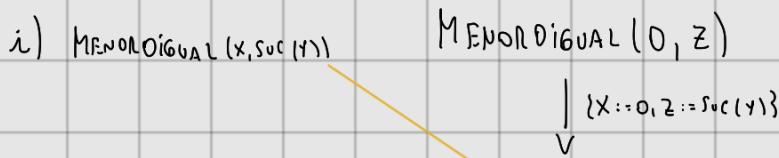
Considerar las siguientes definiciones:

```

natural(0).
natural(suc(X)) :- natural(X).
menorIGual(X, suc(Y)) :- menorIGual(X, Y).
menorIGual(X, X) :- natural(X).

```

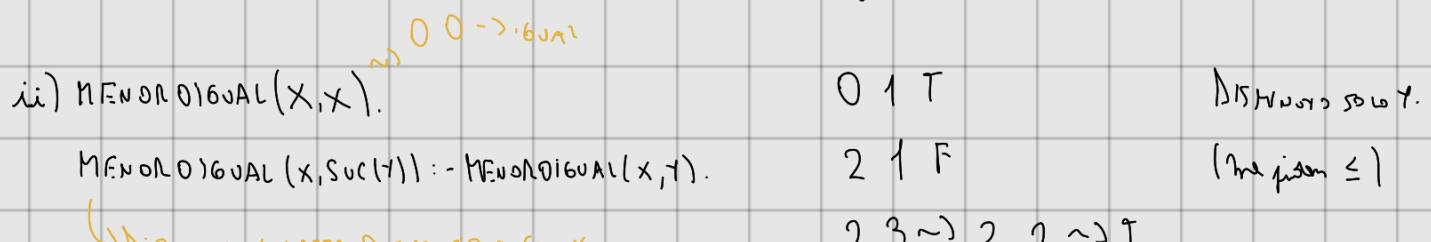
- i. Explicar qué sucede al realizar la consulta  $\text{menorIGual}(0, X)$ .
- ii. Describir las circunstancias en las que puede colgarse un programa en Prolog. Es decir, ejecutarse infinitamente sin arrojar soluciones.
- iii. Corregir la definición de  $\text{menorIGual}$  para que funcione adecuadamente.



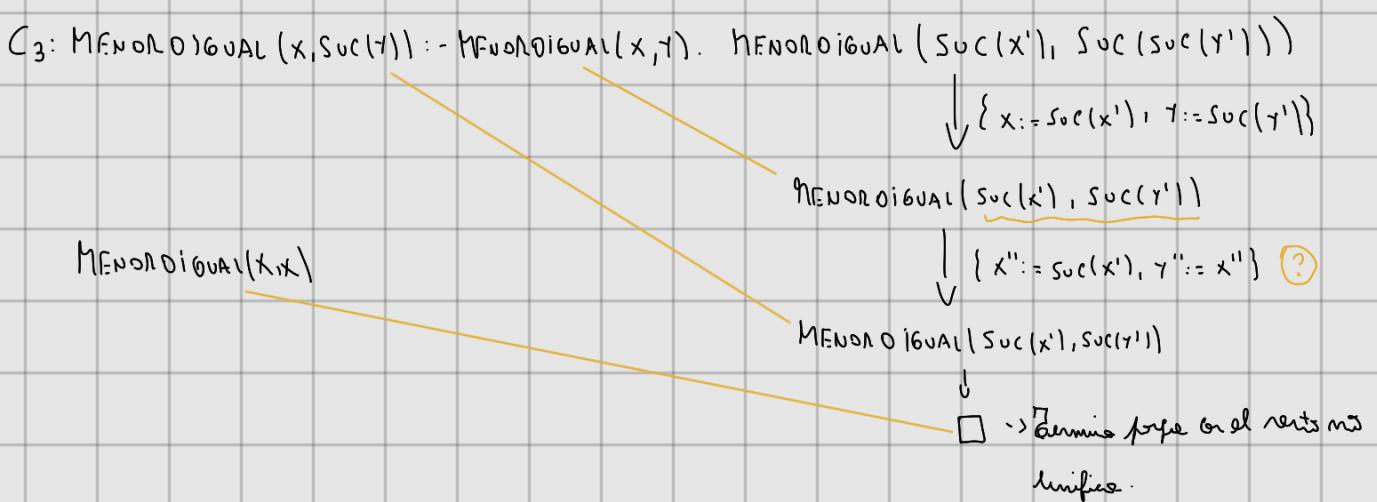
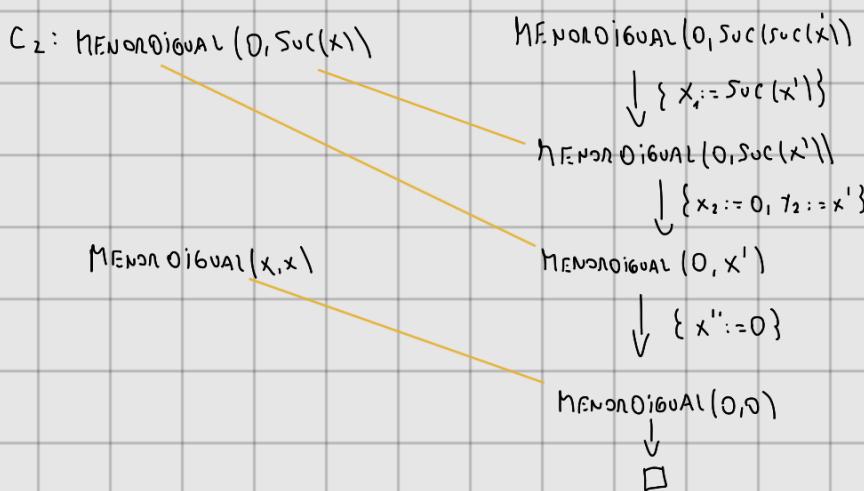
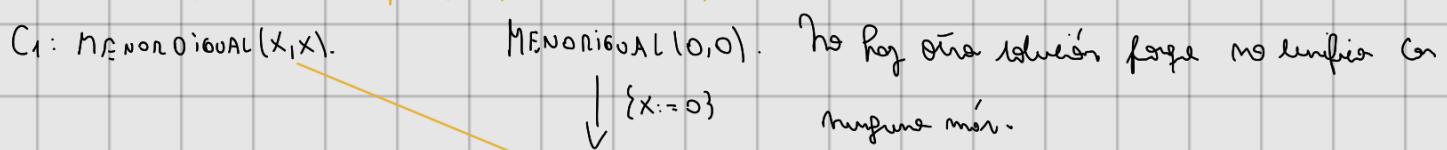


siempre unifica con la primera cláusula, nunca falla. Por eso no hace las siguientes.

iii) Que unifica siempre (en una cláusula recursiva y no le ocurre a un cerrado).



He hecho nulos el cas 2 1, pero como que  $\text{Suc}(\text{Suc}(\text{Suc}(\text{F})) \neq \text{Cero}$ .



#### Ejercicio 4 ★

Definir el predicado `juntar(?Lista1,?Lista2,?Lista3)`, que tiene éxito si `Lista3` es la concatenación de `Lista1` y `Lista2`. Por ejemplo:

```

?- juntar([a,b,c], [d,e], [a,b,c,d,e]). → true.
?- juntar([a,b,c], [d,e], L). → L = [a,b,c,d,e].
?- juntar([a,b,c], L, [a,b,c,d,e]). → L = [d,e].

```

?- juntar(L, [d,e]), [a,b,c,d,e]).  
 -> L = [a,b,c].  
 ?- juntar(L1, L2, [1,2,3]).  
 -> L1 = [], L2 = [1, 2, 3]; L1 = [1], L2 = [2, 3];  
 L1 = [1,2], L2 = [3]; L1 = [1,2,3], L2 = [].

Al igual que la mayoría de los predicados, puede dar false después de agotar los resultados.

Nota: este predicado ya está definido en prolog con el nombre append.

me daña relevancia.

CONCATENAR L<sub>1</sub> CON L<sub>2</sub>. RES:L<sub>3</sub>

JUNTAR([ ], L<sub>2</sub>, L<sub>2</sub>). *(Como hace recurrir el L<sub>1</sub>, si pone nada la res en L<sub>2</sub>)*

JUNTAR([H|T], L<sub>2</sub>, [H|L<sub>3</sub>]) :- JUNTAR(T, L<sub>2</sub>, L<sub>3</sub>).

JUNTAR([H|T], L<sub>2</sub>, [H|L<sub>3</sub>]) JUNTAR([1,2] L<sub>2</sub>, [1,2,3])

{T := [2], L<sub>2</sub> := L<sub>1</sub>, L<sub>3</sub> := [2,3]}



JUNTAR([H|T], L<sub>2</sub>, [H|L<sub>3</sub>]) JUNTAR([2], L<sub>2</sub>, [2,3])

{T := [], L<sub>2</sub> := L<sub>2</sub>, L<sub>3</sub> := [3]}



JUNTAR([ ], L<sub>2</sub>, [3])

Exit

□ {L<sub>2</sub> := [3]}

JUNTAR([1,2], [3], [1,2,3])

Exit

JUNTAR([2], [3], [2,3])

Exit

#### Ejercicio 5 ★

Definir los siguientes predicados sobre listas usando **append**:

- I. last(?L, ?U), donde U es el último elemento de la lista L.
- II. reverse(+L, +R), donde R contiene los mismos elementos que L, pero en orden inverso.  
Ejemplo: reverse([a,b,c], [c,b,a]).  
Mostrar el árbol de búsqueda para el ejemplo dado.
- III. prefijo(?P, +L), donde P es prefijo de la lista L.
- IV. sufijo(?S, +L), donde S es sufijo de la lista L.
- V. sublista(?S, +L), donde S es sublista de L.
- VI. pertenece(?X, +L), que es verdadero si el elemento X se encuentra en la lista L. (Este predicado ya viene definido en Prolog y se llama **member**).

i) LAST([1,2], 2) *→ Busco si Ofrecer primera 3 → ofrecer U me da L.*

Mis: last([1|2], 2) :- last([2], 2)

↳ last([2], 2) T.M.E.

LAST([U], U).

} Este funciona

LAST([\_1|T], U) :- LAST(T, U).

*LISTA CORTA      ULTIMO ELEMENTO  
                        ↓  
                        RES (INPUT)*

LAST(L1, U) :- APPEND( [ ], [U], L1).

iii) REVERSE(+L, ?R)

[1,2,3] llega a (B. BACKTRACK NO GUARDIA UN ORDER).

"CABEZA DE L ES LAST(R)" ARCHICOS AMBAS.

[1,2,3] ≡ [3,2,1]

[2,3] ≡ [3,2]

[3] ≡ [3]

¿Qué hace con APPEND? No! que maneja LAS ÚLTIMAS LISTAS, más, el APPEND no se ejecuta en el resultado final de la consulta.

↓ "El PRIMEROS DEL SUFIXO", despues SEGUNDO...

REVERSE([ ], [ ]).

REVERSE(L, R) :- APPEND(L, [U], L), REVERSE(U, P), APPEND([U], P, R). ↴  
NES ACONDICIONES.

REVERSE([1, 2], R)

{L := [1, 2]}

APPEND(L, [U], [1, 2]), REVERSE(L, P), APPEND([U], P, R) ↴ R = [2, 1]

{L = [1], U = [2]} ↗ REVERSE([1], P) ↗ APPEND([2], [1], R)

1 ↘ {L := [1]}

APPEND(L, [U], [1]), REVERSE(L, P), APPEND([U], P, R)  
{L := [1], U := [2]}

3 ↗ REVERSE([1], P) → □

4 ↗ APPEND([1], [ ], R) → R = [1]

• VUELTA

iii) PREFijo (?P, +L)

PREFijo(P, L) :- APPEND(P, \_, L). ✓

PREFijo(X, [1, 2, 3])

↳ X : [1, 2, 3] - : [ ] = L

X : [1, 2] - : [3] = L

X : [1] - : [2, 3] = L

X : [ ] - : [1, 2, 3] = L

iv) Sufijo (?S, L)

Sufijo(S, L) :- APPEND( \_, S, L).

X : [1, 2, 3] - : [ ], X : [1, 2] - : [3], X : [1] - : [2, 3]

v) ~) PREFijo DE ALGUN Sufijo

SUBLISTA (?S, +L)

L : [1, 2, 3] [2] de TNF.

Sufijos: [ ], [3], [2, 3], [1, 2, 3]

PREFijos Sufijos: [ ], [3], [2], [2, 3], [1], [1, 2], [1, 2, 3]

SUBLISTA(S, L) :- APPEND(\_, S<sub>1</sub>, L), APPEND(S<sub>1</sub>, S<sub>2</sub>).

SUBLISTA(S, L) :- S<sub>1</sub>Fijo(S<sub>1</sub>, L), PNFijo(S, S<sub>1</sub>).

Vi) PERTENECIE(E, [E|\_]).

PERTENECIE(E, [H|T]) :- E \= H, PERTENECIE(E, T).



HEAD DE SUFISO:

PERTENECIE(X, L) :- SUFISO(S, L), S \= [], S = [X|\_].

≡ PERTENECIE(X, L) :- APPEND(\_, [X|\_], L).

#### Ejercicio 6 ★

Definir el predicado aplanar(+Xs, -Ys), que es verdadero si Ys contiene los elementos de todos los niveles de Xs, en el mismo orden de aparición. Los elementos de Xs son enteros, átomos o nuevamente listas, de modo que Xs puede tener una profundidad arbitraria. Por el contrario, Ys es una lista de un solo nivel de profundidad.

Ejemplos:

?- aplanar([a, [3, b, []], [2]], L). → L=[a, 3, b, 2]  
?- aplanar([[1, [2, 3], [a]], [[[[]]]], L). → L=[1, 2, 3, a]

Nota: este predicado ya está definido en prolog con el nombre flatten.

[a, [3, b, []], [2]]

L, [a | [3, b, []], [2]]

\* L, [3 | [b | []], [2]]

↑ AT

L, [3 | [b | []]]

AT  
L, [b | []]

APLANAR([], []).

APLANAR([X|XS], [X|YS]) :- NOT(is\_lis(X)), APLANAR(XS, YS).

APLANAR([X|XS], -YS) :- is\_lis(X), APLANAR(X, ZS), APLANAR(XS, XS2), APPEND(ZS, XS2, YS).

#### Ejercicio 7 ★

Definir los siguientes predicados, usando **member** y/o **append** según sea conveniente:

- I. intersección(+L1, +L2, -L3), tal que L3 es la intersección sin repeticiones de las listas L1 y L2, respetando en L3 el orden en que aparecen los elementos en L1.
- II. partir(N, L, L1, L2), donde L1 tiene los N primeros elementos de L, y L2 el resto. Si L tiene menos de N elementos el predicado debe fallar. ¿Cuán reversible es este predicado? Es decir, ¿qué parámetros pueden estar indefinidos al momento de la invocación?
- III. borrar(+ListaOriginal, +X, -ListaSinXs), que elimina todas las ocurrencias de X de la lista ListaOriginal.
- IV. permutación(+L1, ?L2), que tiene éxito cuando L2 es permutación de L1. ¿Hay una manera más eficiente de definir este predicado para cuando L2 está instanciada?
- V. reparto(+L, +N, -Listas) que tenga éxito si Listas es una lista de N listas ( $N \geq 1$ ) de cualquier longitud - incluso vacías - tales que al concatenarlas se obtiene la lista L.

VI. repartoSinVacías(+L, -Listas) similar al anterior, pero ninguna de las listas de Listas puede ser vacía, y la longitud de Listas puede variar.

MEMBERDETERMINISTICO(L, E) :- APPEND(S, [E|\_], L), NOT(MEMBERDETERMINISTICO(S, E)).

i)  $L_3 : L_1 \cap L_2$ .

$L_3$ : NO TIENE REPETICIONES.

$L_3$ : MISMO ORDEN QUE  $L_1$ .

INTERSECCION([], \_, []).

INTERSECCION([X|XS], YS, [X|ZS]) :- INTERSECCION(XS, YS, ZS), MEMBERDETERMINISTICO(X, YS).

INTERSECCION([X|XS], YS, ZS) :- INTERSECCION(XS, YS, ZS), NOT(MEMBERDETERMINISTICO(X, YS)). ✓

PARTIR(N, L, L1, L2) :- LENGTH(L, LL), LL ≥ N, LENGTH(L1, N), LENGTH(L2, N), APPEND(L1, L2, L). ✓

N debe ser INSTANCIA de ≥ N, porque > es un operador ARITMÉTICO.

ii) BORRAR([], \_, []).

BORRAR([X|XS], X, R) :- BORRAR(XS, X, R).

BORRAR([X|XS], E, R) :- E ≠ X, BORRAR(XS, E, RR), APPEND([X], RR, R). ✓

iii) SACARDUPPLICADOS(+L1, -L2)  $(1, 2, 3, 1) \sim (1, 2, 3)$

SACARDUPPLICADOS([], []).

SACARDUPPLICADOS([X|XS], R) :- BORRAR(XS, X, LR), SACARDUPPLICADOS(LR, RR), APPEND([X], RR, R). ✓

V) REPARTO(+L, +N, -L2)

$\sigma(L, N, L_1, L_2) :- APPEND(L_1, L_2, L), S \dots$

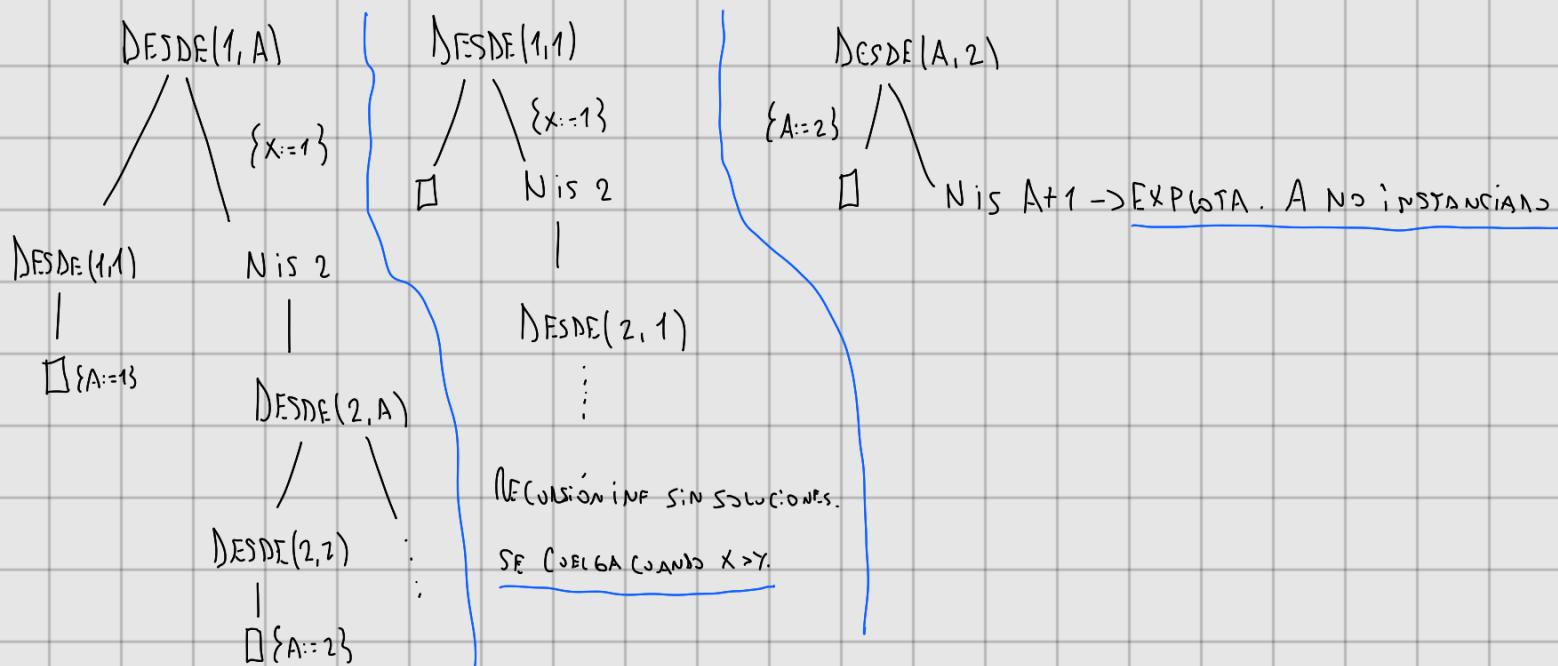
desde(X, Y) :- N is X+1, desde(N, Y).

• I. ¿Cómo deben instanciarse los parámetros para que el predicado funcione? (Es decir, para que no se cuelgue ni produzca un error). ¿Por qué?

• II. Dar una nueva versión del predicado que funcione con la instancia `desdeReversible(+X, ?Y)`, tal que si `Y` está instanciada, sea verdadero si `Y` es mayor o igual que `X`, y si no lo está genere todos los `Y` de `X` en adelante.

i)  $\Delta_{DESDE}(x, x)$ .

$\Delta_{DESDE}(X, Y) :- N \text{ is } X+1, \Delta_{DESDE}(N, Y)$ .



Entonces que Funcione:  $\Delta_{DESDE}(+X, -Y)$

ii)  $\Delta_{DESDEREVISBLE}(X, Y) :- \text{NOVAN}(Y), Y \geq X$ .

$\Delta_{DESDEREVISIBLE}(X, Y) :- \text{VAN}(Y), \Delta_{DESDE}(X, Y)$ .

#### Ejercicio 11 ★

Un árbol binario se representará en Prolog con:

- `nil`, si es vacío.
- `bin(izq, v, der)`, donde `v` es el valor del nodo, `izq` es el subárbol izquierdo y `der` es el subárbol derecho.

Definir predicados en Prolog para las siguientes operaciones: `vacío`, `raíz`, `altura` y `cantidadDeNodos`. Asumir siempre que el árbol está instanciado.

`VACIO(nil)`.

`RAIZ(bin(l, r, o), r)`.

`ALTURA(nil, 0)`.

`ALTURA(bin(l, r, d), X) :- ALTURA(l, AL), ALTURA(d, AD), ALT is MAX(AL, AD), X is 1 + ALT.`

CANTNODOS(NIL, 0).

CANTNODOS(BIN(I, R, D), X) :- CANTNODOS(I, CI), CANTNODOS(D, CD), X is CI + CD.

### Ejercicio 12 ★

Definir los siguientes predicados, utilizando la representación de árbol binario definida en el ejercicio 11:

I. inorder(+AB, -Lista), que tenga éxito si AB es un árbol binario y Lista la lista de sus nodos según el recorrido inorder.

II. arbolConInorder(+Lista, -AB), versión inversa del predicado anterior.

III. aBB(+T), que será verdadero si T es un árbol binario de búsqueda.

IV. aBBInsertar(+X, +T1, -T2), donde T2 resulta de insertar X en orden en el árbol T1. Este predicado ¿es reversible en alguno de sus parámetros? Justificar.

i)

INORDER(NIL, []).

INORDER(BIN(I, R, D), L) :- INORDER(I, LI), INORDER(D, LD), APPEND(LI, [R], LD), APPEND(LD, L).

ii) Añadir un /NODOS para cada posición miz

[1, 2, 3] → R=1.



La relación de la lista con el árbol en la rig:

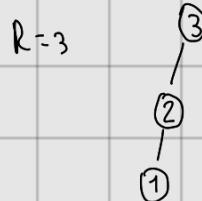
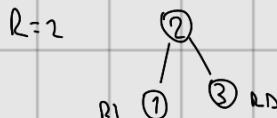
• Cada elem en Miz es algún SÓLO nodo.

• O bien el ACTUAL es Miz o el Miz es entero.

rig → bien en DFA.

• Se lanza hacia en el Árbol NIL.

• De varios soluciones.



ARBOLCONINORDER([ ], NIL).

ARBOLCONINORDER([H|T], BIN(I, R, D)) :- H = R, ARBOLCONINORDER(T, I), ARBOLCONINORDER(T, D).

ARBOLCONINORDER([H|T], BIN(I, R, D)) :- ARBOLCONINORDER(T, I), ARBOLCONINORDER(T, D).

? ACI([1], A)

ARBOLCONINORDER([H|T], BIN(I, R, D))

{ H := 1, T := [], A := BIN(I2, R2, D2) }

?  
I1 = R2, ACI([], I1), ACI([], D2)

{ [] := [], I2 := NIL }

ACI([], D2)

{ [] := [], D2 := NIL }

?

S = { D2 := NIL } ∪ { T2 := NIL } ∪ { H := 1, T := [], A := BIN(I2, R2, D2) }

L → { D2 := NIL, I2 := NIL, H := 1, T := [], A := BIN(I2, R2, D2) }

?

c|R2 Que da GRO VAN

Andrómata? CNEO QUE PSE CASO FA MAL.

?

PNEG: DOS N.C EN UNA CLÁUSULA VS 1 N.C POR CLÁUSULA.

INORDER([NIL, [ ]]).

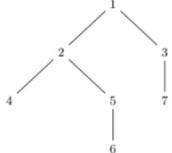


INORDER(BIN(I,R,D), L) :- INORDER(I, L<sub>1</sub>), INORDER(D, L<sub>2</sub>), APPEND(L<sub>1</sub>, [R], L<sub>3</sub>), APPEND(L<sub>3</sub>, L<sub>2</sub>, L).

Trabajaremos con árboles binarios, usando nil y bin(I, V, D) para representarlos en Prolog. Como ejemplo, usaremos el siguiente hecho:

arbol(bin(bin(nil, 4, nil), 2, bin(nil, 5, bin(nil, 6, nil))), 1, bin(bin(nil, 7, nil), 3, nil)).

que representa el árbol:



- a. Definir el predicado rama(+A,-C) que es verdadero cuando C es un camino desde la raíz del árbol A hasta alguna de sus hojas. Por ejemplo:

```
?- arbol(A), rama(A,C).
C = [1,2,4];
C = [1,2,5,6];
C = [1,3,7];
false.
```

- b. ¿El predicado anterior es reversible en C? Justificar brevemente.

- c. Definir el predicado ramaMasLarga(+A,-C) que es verdadero cuando C es la rama más larga de A. Por ejemplo:

```
?- arbol(A), ramaMasLarga(A,C).
C = [1,2,5,6];
false.
```

- d. Definir el predicado ramaUnicaDeLong(+A,+N,-C) que es verdadero cuando C es la única rama de A que tiene longitud N. Por ejemplo:

```
?- arbol(A), ramaUnicaDeLong(A,4,C).
C = [1,2,5,6];
false.
?- arbol(A), ramaUnicaDeLong(A,3,C).
false.
```

RAMA: TODO IZQ, T >= DER O CONTRADICCIONES

RAMA(BIN(NIL,R NIL), [R])

RAMA(BIN(I,R,D), [R|T]) :- RAMA(T,I).

RAMA(BIN(T,R,D), [R|T]) :- RAMA(T,D).

RAMAMASLARGA(A,C) :- RAMA(A,C), LENGTH(C,R<sub>L</sub>), NOT((RAMA(A,B<sub>2</sub>), LENGTH(B<sub>2</sub>,R<sub>C</sub>), R<sub>C</sub> > R<sub>L</sub>))

RAMAUNICADELONG(A,N,C) :- RAMA(A,C), LENGTH(C,N), NOT((RAMA(A,B<sub>2</sub>), LENGTH(B<sub>2</sub>,N), C = B<sub>2</sub>))

iii) ABB: Se hace USANDO INORDER, AGREGUE COMPUTACION LISTAS.

Necesito función max (para iZQ) y min (para DER)

#### Ejercicio 13 ★

Definir el predicado coprimos(-X,-Y), que genere uno a uno *todos* los pares de números naturales coprimos (es decir, cuyo máximo común divisor es 1), sin repetir resultados. Usar la función gcd del motor aritmético.

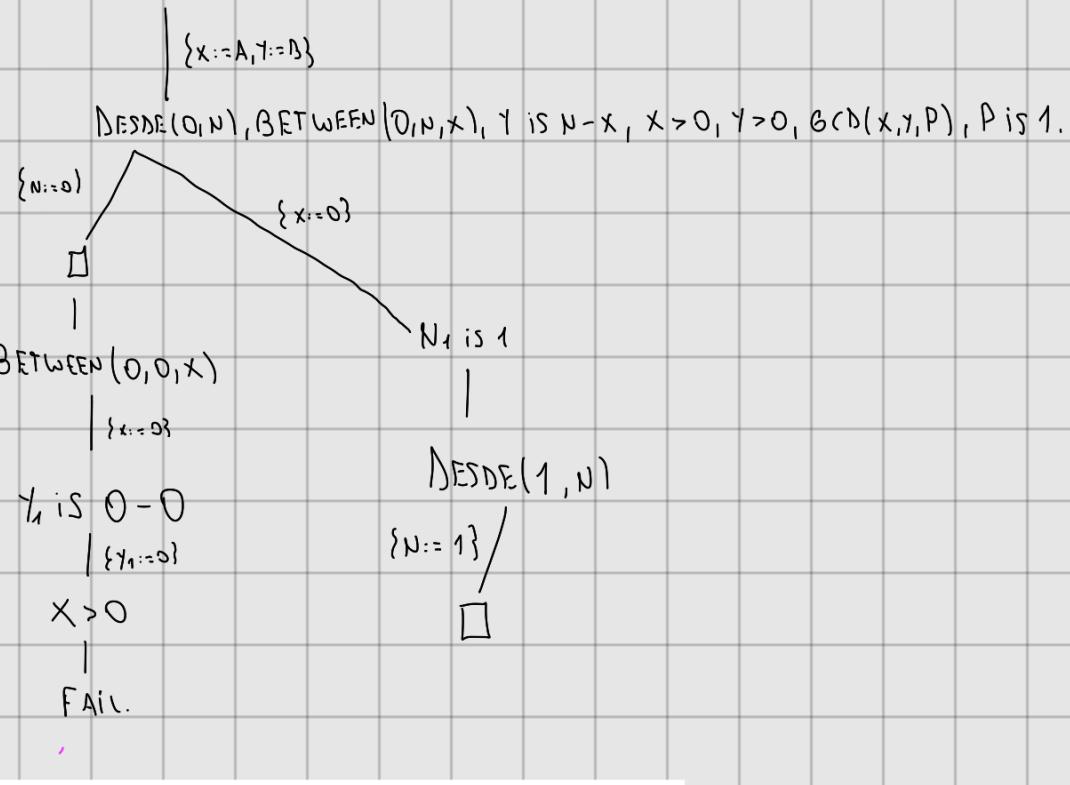
(COPRIMOS(-X,-Y))

MISWIPL LO USA ASÍ:



$\text{Coprimos}(x, y) :- \text{DESDE}(0, n), \text{BETWEEN}(0, n, x), y \text{ is } n - x, x > 0, y > 0, \text{GCD}(x, y, p), p \text{ is } 1.$

$\text{Coprimos}(A, B)$



NEGACIÓN POR FALLA Y CUT

### Ejercicio 16 ★

A Ana le gustan los helados que sean a la vez cremosos y frutales. En una heladería de su barrio, se encontró con los siguientes sabores:

|                                |                                  |                                     |
|--------------------------------|----------------------------------|-------------------------------------|
| <code>frutal(frutilla).</code> | <code>cremoso(banana).</code>    | <code>cremoso(dulceDeLeche).</code> |
| <code>frutal(banana).</code>   | <code>cremoso(americana).</code> |                                     |
| <code>frutal(manzana).</code>  | <code>cremoso(frutilla).</code>  |                                     |

Ana desea comprar un cucurcho con sabores que le gustan. El cucurcho admite hasta 2 sabores. Los siguientes predicados definen las posibles maneras de armar el cucurcho.

`leGusta(X) :- frutal(X), cremoso(X).`  
`cucurcho(X, Y) :- leGusta(X), leGusta(Y).`

- I. Escribir el árbol de búsqueda para la consulta `?- cucurcho(X, Y).`
- II. Indicar qué partes del árbol se podan al colocar un ! en cada ubicación posible en las definiciones de `cucurcho` y `leGusta`.

Le juntan lo q van juntando juntos: Combinación de Frutal y BANANA.

$\text{Cucurcho}(X, Y) :- \text{leGusta}(X), \text{leGusta}(Y)$

$\text{Cucurcho}(X, Y)$

$\{X := X_1, Y := Y_1\}$

$\text{LE-GUSTA}(X) :- \text{FRUTAL}(X), \text{CREMOSO}(X)$

$\text{LE-GUSTA}(X_1), \text{LE-GUSTA}(Y_1) \quad \text{LE-GUSTA}(X) :- \text{FRUTAL}(X), \text{CREMOSO}(X)$

$\text{FRUTAL}(X_2), \text{CREMOSO}(X_2)$

$\{X_2 := \text{FRUTILLA}\}$

$\{X_2 := \text{FRESCA}\}$

□

□

$\{Y_1 := Y_2\}$

$\text{FRUTAL}(Y_2), \text{CREMOSO}(Y_2)$

$\{Y_2 := \text{FRUTILLA}\}$

$\{Y_2 := \text{FRESCA}\}$

□

□

- PRIMERA RTA:  $\{X := \text{FRUTILLA}, Y := \text{FRUTILLA}\}$

REDO

LF.GUSTA( $x_1$ ), LF.GUSTA( $y_1$ )

↓

IGUAL AYNT.

LF.GUSTA( $x$ ) :- FNUJAL( $x$ ), CREMOS( $x$ )

{ $y_1 := y_2$ }

FNUJAL( $y_2$ ), CREMOS( $y_2$ )

{ $y_2 := BANANA$ }

□

{ $y_2 := BANANA$ }

□

- 2DA AYA: { $X := FNUJAL$ ,  $Y := BANANA$ }

(MANZANA NO ES (BANANA))

DE DO, no hay MAS OPT para. LEGUSTA( $y_1$ ). CANTINA  $x_2$ . \*

A partir de aqüo hay PERMUTACIONES (Duplicados).

F.GUSTA( $x$ ) :- FNUJAL( $x$ ), CREMOS( $x$ )

LF.GUSTA( $x_1$ ), LF.GUSTA( $y_1$ )

LF.GUSTA( $x$ ) :- FNUJAL( $x$ ), CREMOS( $x$ )

{ $x_1 := x_2$ }

{ $y_1 := y_2$ }

FNUJAL( $x_2$ ), CREMOS( $x_2$ )

{ $x_2 := BANANA$ }

□

FNUJAL( $y_2$ ), CREMOS( $y_2$ )

{ $y_2 := FNUJAL$ }

□

- 3RA AYA: { $X := BANANA$ ,  $Y := FNUJAL$ }

Pero FNUJAL, BANANA ≡ BANANA, FNUJAL. NO REPETIDAS.

PARA FIJAR el FNUJAL uno en ( $w$ ).

CUCURUCHO( $x, y$ ) :- LEGUSTA( $x$ ), !, LF.GUSTA( $y$ ).



( $\rightarrow$  F.UITA \*1. Es decir, no permite que cambie el valor de  $X$  uno no se reutilice uno.

### Ejercicio 17 ★

- i. Sean los predicados  $P(?X)$  y  $Q(?X)$ , ¿qué significa la respuesta a la siguiente consulta?  
?-  $P(Y)$ ,  $\text{not}(Q(Y))$ .
- ii. ¿Qué pasaría si se invirtiera el orden de los literales en la consulta anterior?
- iii. Sea el predicado  $P(?X)$ , ¿Cómo se puede usar el  $\text{not}$  para determinar si existe una única  $Y$  tal que  $P(?Y)$  es verdadero?

NOT es un METAPREDICADO que tiene dos casos el primero que le pasemos falle.

i) La Comisión Orgánica no tiene P y tiene Q.

ii) EXPLOTA. no es un metapredicado que exprese que algunos de los argumentos son falsos (en todo los argumentos inválidos).

↳ NOT(Q(Y)) :- Q(Y), !, FAIL

EXPLOTA.

A DENTRO DE NOT

iii) Podemos decir que existe un X que vale P. si tenemos en Y = X que vale P(X) en debe fallar.

-> P(X), NOT((P(Y), X=Y))



### Ejercicio 18 ★

Definir el predicado corteMásParejo(+L, -L1, -L2) que, dada una lista de números, realiza el corte más parejo posible con respecto a la suma de sus elementos (puede haber más de un resultado). Por ejemplo:

?- corteMásParejo([1,2,3,4,2], L1, L2). → L1 = [1, 2, 3], L2 = [4, 2] ; false.  
?- corteMásParejo([1,2,1], L1, L2). → L1 = [1], L2 = [2, 1] ; L1 = [1, 2], L2 = [1] ; false.

"Unir se la suma de los elementos en mínimo" → Mejor.