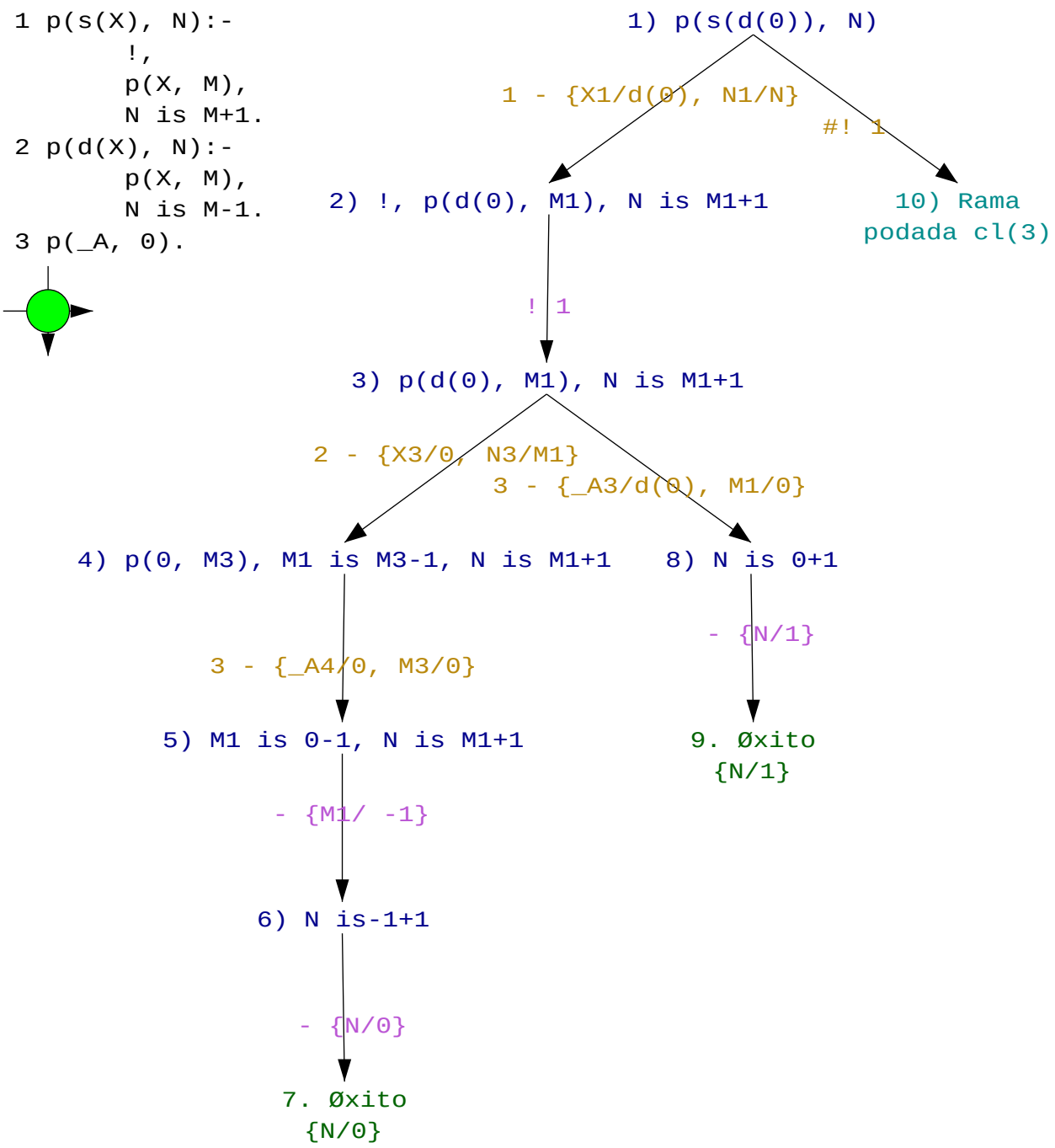
Ejercicio 1 [2 puntos] La relación `p/2` está definida por

```
p(s(X),N) :- !,p(X,M),N is M + 1.
p(d(X),N) :- p(X,M),N is M - 1.
p(X,0).
```

1. Escribe el árbol de resolución correspondiente al objetivo `p(s(d(0)),N).`



2. Indica *razonadamente* qué responde Prolog a la pregunta `?- p(X,0).`

Prolog responde `ERROR: Out of local stack`. La ejecución del programa entra en una rama infinita puesto que un objetivo de la forma `p(V1,Term)` unifica siempre con la cabeza de la primera regla y produce como próximo objetivo a resolver (una vez ejecutado el correspondiente corte) un nuevo objetivo de la forma `p(V3,V4)`.

Ejercicio 2 [2 puntos]

1. Se considera la siguiente relación `distancia/4`.

```
distancia(A,B,L,N) :-
 append( , , ),
 append( , , ),
 length(L2,N).
```

   Completa la definición de la relación anterior de modo que `distancia(A,B,L,N)` se verifique si `N` es la distancia entre los elementos `A` y `B` en la lista `L`. Por ejemplo:

```
?- distancia(a,d,[a,b,c,d],N).
N = 2
?- distancia(A,B,[1,2],N).
A = 1  B = 2  N = 0 ;
No
```

(●) Solución 1:

```
distancia(A,B,L,N) :-
 append(_,[A|L1],L),
 append(L2,[B|_],L1),
 length(L2,N).
```

(●) Solución 2:

```
distancia(A,B,L,N) :-
 append(L1,[B|_],L),
 append(_,[A|L2],L1),
 length(L2,N).
```

---

2. La relación `q/3` está definida por

```
q(_,[],[]).
q(F,[X|L1],L2) :- X @> F,!,q(F,L1,L2).
q(F,[X|L1],[X|L2]) :- q(F,L1,L2).
```

   Obténgase una definición *no recursiva* de la relación `q(+F,+L1,-L2)` anterior.

```
q(F,L1,L2) :- findall(X,(member(X,L1),X @=< F),L2).
```

Ejercicio 3 [3 puntos] Define la relación `existe_todos(+L1,+L2)` que se verifica si existe un elemento de la lista `L1` que satisface todos los predicados que aparecen en la lista de predicados unarios `L2`. Por ejemplo:

```
?- existe_todos([a,-5,5],[atomic,number,integer]).
Yes
?- existe_todos([a,-5,5],[atom,integer]).
No
```

(•) Solución 1:

```
existe_todos(L1,L2) :-
    member(X,L1),not((member(P,L2),not(apply(P,[X])))).
```

(•) Solución 2:

```
existe_todos([X|_],L2) :- todos(X,L2),!.
existe_todos([_|L],L2) :- existe_todos(L,L2).
```

`todos(+X,+L)` se verifica si `X` satisface todos los predicados que aparecen en la lista de predicados unarios `L`.

```
todos(_,[]).
todos(X,[P|L]) :- apply(P,[X]),todos(X,L).
```

Ejercicio 4 [3 puntos] Un árbol binario es vacío, o bien consta de tres partes: una raíz, un subárbol izquierdo (que debe ser un árbol binario) y un subárbol derecho (que debe ser un árbol binario). Consideraremos la siguiente representación:

- la constante `nil` representa el árbol vacío,
- si el árbol no es vacío, entonces tendrá la forma `t(I,R,D)` donde `R` es la raíz, `I` es el subárbol izquierdo y `D` es el subárbol derecho.

1. Define la relación `rama(+A,-L)` que se verifica si `L` es una rama del árbol binario `A`. Por ejemplo:

```
?- rama(t(t(t(nil,b,nil),d,t(nil,e,nil)),a,t(nil,c,nil)),L).
L = [a, d, b] ;
L = [a, d, e] ;
L = [a, c] ;
No
```

```
rama(t(nil,R,nil),[R]) :- !.
rama(t(I,R,nil),[R|L]) :-
   !,rama(I,L).
 rama(t(nil,R,D),[R|L]) :-
   !,rama(D,L).
rama(t(I,R,D),[R|L]) :-
   (rama(I,L) ; rama(D,L)).
```

2. Un grafo (y, por tanto, un árbol binario) también puede representarse mediante un conjunto de hechos de la forma `arista(+V1,+V2)`, donde `V1`, `V2` son dos vértices del grafo unidos por una arista.

   Define el predicado `transforma(+A)` que añade a la base de conocimiento los hechos de la forma `arista(+V1,+V2)` que representan el árbol binario `A`. Por ejemplo:

```
?- transforma(t(t(t(nil,c,nil),b,nil),a,t(nil,d,nil))).
Yes
?- listing(arista).
arista(a,b).
arista(a,d).
arista(b,c).
```

```
transforma(nil).
transforma(t(nil,_,nil)).
transforma(t(t(I,N,D),R,nil)) :-
    assert(arista(R,N)), transforma(t(I,N,D)).
transforma(t(nil,R,t(I,N,D))) :-
    assert(arista(R,N)), transforma(t(I,N,D)).
transforma(t(t(I1,N1,D1),R,t(I2,N2,D2))) :-
    assert(arista(R,N1)), assert(arista(R,N2)),
    transforma(t(I1,N1,D1)), transforma(t(I2,N2,D2)).
```
