

Tarea Independiente 23/10/2025

David Núñez Franco

October 27, 2025

Inventario de Conceptos Claves

- Computación como búsqueda de pruebas usando DFS
- Resolución
- Árbol de prueba
- Problemas con búsqueda DFS
- Cortando la búsqueda con Cut (!)
- Backtracking como ciclos

Ejercicio 0

¿Qué hace el siguiente predicado `foo(+N, -R)` para `N` un entero no negativo:

```
foo(N, M) :- foo(N, 1, M).
foo(0, M, M) :- !.
foo(N, M, R) :- N1 is N - 1, M1 is M * N, foo(N1, M1, R).
```

Solución

```
1    % foo(+N, -R): calcula el factorial de N, para N >= 0.
2    % Usa un acumulador para calculo eficiente.
3
4    % Caso inicial: se llama con N y se crea un acumulador que
5        % empieza en 1.
6    foo(N, R) :-
7        foo(N, 1, R).
8
9    % Caso base: cuando N llega a 0, el resultado es el
10       % acumulador actual (M).
```

```

9      % El corte (!) indica que no hay mas caminos que probar.
10     foo(0, M, M) :- !.
11
12
13     % Caso recursivo: mientras N sea mayor que 0.
14     % Se disminuye N en 1 y se multiplica el acumulador por el
15     % valor actual de N.
16     % Luego se vuelve a llamar recursivamente con esos nuevos
17     % valores.
18     foo(N, M, R) :- !.
19     N1 is N - 1,           % resta 1 a N
20     M1 is M * N,          % multiplica el acumulador por N
21     foo(N1, M1, R).       % llamada recursiva
22
23
24     /*
25      Ejemplos de uso:
26
27      Paso a paso con N=5:
28
29      1. foo(5,R) -> foo(5,1,R)
30
31      2. foo(5,1,R) -> N1=4, M1=5 -> foo(4,5,R)
32
33      3. foo(4,5,R) -> N1=3, M1=20 -> foo(3,20,R)
34
35      4. foo(3,20,R) -> N1=2, M1=60 -> foo(2,60,R)
36
37      5. foo(2,60,R) -> N1=1, M1=120 -> foo(1,120,R)
38
39      6. foo(1,120,R) -> N1=0, M1=120 -> foo(0,120,R)
40
41      foo(0,120,R) hace match con la base y, por el !, unifica R
42      =120 y termina.
43
44 */

```

Ejercicio 1

Escriba begins(L, S, N) para que funcione este ejemplo

%

% begins(L, S, N): L es una lista que a la izquierda tiene la sublista S y esta es de

% begins(L, S, N) :- /* su respuesta

*/.

```
test_begins :-
    Min = 3,
```

```

L = [a,b,c,d,e,f],
forall(begins(L, S, Min),
      (length(S, N),
       format('Sublist=~w. Length= ~d >= Min= ~d~n', [S, N, Min]))
    )
)
.

:- test_begins.
% Salida
Sublist=[a,b,c]. Length= 3 >= Min= 3
Sublist=[a,b,c,d]. Length= 4 >= Min= 3
Sublist=[a,b,c,d,e]. Length= 5 >= Min= 3
Sublist=[a,b,c,d,e,f]. Length= 6 >= Min= 3
true.

```

Solución

```

1   % begins(+L, -S, +N)
2   % Genera todas las sublistas S que son prefijos de L, y cuya
3   % longitud es al menos N.
4
5   begins(L, S, N) :-
6       append(S, _, L), % S debe ser prefijo de L
7       length(S, Len), % mide la longitud de S
8       Len >= N. % aseguramos longitud minima
9
10
11  % Prueba del predicado
12
13  test_begins :-
14      Min = 3,
15      L = [a,b,c,d,e,f],
16      forall(begins(L, S, Min),
17          (length(S, N),
18           format('Sublist=~w. Length= ~d >= Min= ~d~n', [S, N, Min]))
19         )
20     ).
```

```

1   ?- [begins].
2   Sublist=[a,b,c]. Length= 3 >= Min= 3
3   Sublist=[a,b,c,d]. Length= 4 >= Min= 3
4   Sublist=[a,b,c,d,e]. Length= 5 >= Min= 3
5   Sublist=[a,b,c,d,e,f]. Length= 6 >= Min= 3
6   true.
```

Listing 1: output

Ejercicio 2

Escriba range(A, B, N) que genere N con A <= N < B de forma que lo siguiente funciona.

```
range(A, B, L) :- /* su respuesta */

test_range :-
A=5, B= 10,
forall((range(A, B, R), member(N, R)),
format('~d <= ~d <= ~d~n', [A, N, B])
)
.

:- test_range.
```

Solución

```
1   % range(+A, +B, -L)
2   % Genera una lista L con todos los enteros desde A hasta B-1
3   % Ejemplo: ?- range(5,10,L). -> L = [5,6,7,8,9].
4
5   range(A, B, L) :-
6       A < B, % verifica que A menor que B
7       range_aux(A, B, L). % llama al predicado auxiliar recursivo
8
9   % Caso base: Cuando A alcanza o supera B, ya no hay mas
10    numeros que agregar.
11   % La lista resultante es vacia.
12   range_aux(A, B, []) :-
13       A >= B,
14       !.
15
16   % Caso recursivo: Si A menor que B, se agrega A al inicio de
17   % la lista, y se llama
18   % nuevamente con A+1.
19   range_aux(A, B, [A|R]) :-
20       A < B,
21       A1 is A + 1, % calcula siguiente numero
22       range_aux(A1, B, R). % continua construyendo la lista
23
24   test_range :-
25       A = 5,
```

```

24      B = 10,
25      forall( (range(A, B, R), member(N, R)),
26              format('`~d <= ~d <= ~d`~n', [A, N, B])
27      ) .
28
29      :- test_range.

```

```

1      ?- [range].
2          5 <= 5 <= 10
3          5 <= 6 <= 10
4          5 <= 7 <= 10
5          5 <= 8 <= 10
6          5 <= 9 <= 10
7          true.

```

Listing 2: output

Ejercicio 3

Escriba `solve(-X, -Y, +Z)` talque encuentra enteros no negativos X y Y tales $X + Y = Z$

Solución

```

1      % solve(-X, -Y, +Z) genera todos los pares de enteros no
2          % negativos (X, Y)
3          % tales que X + Y = Z.
4
4      solve(X, Y, Z) :-
5          integer(Z), % asegura que Z sea entero
6          Z >= 0, % asegura que Z sea positivo
7          solve_aux(0, X, Y, Z). % comienza la busqueda con X=0
8
8      % Caso base: Incrementa X desde 0 hasta Z, calculando Y = Z
9          - X.
10     solve_aux(Current, X, Y, Z) :-
11         Current <= Z, % mientras no se sobreponga Z
12         X = Current, % asigna el valor actual a X
13         Y is Z - Current. % calcula Y
14
15     solve_aux(Current, X, Y, Z) :-
16         Current < Z, % si aun no alcanza Z
17         Next is Current + 1, % incrementa X
18         solve_aux(Next, X, Y, Z). % continua generando soluciones
19
20     test_solve :-

```

```
21     Z = 5,  
22     forall(solve(X, Y, Z),  
23             format('X=~d, Y=~d, X+Y=~d~n', [X, Y, Z]))  
24     ).  
25  
26 :- test_solve.
```

```
1  ?- [solve].  
2  X=0, Y=5, X+Y=5  
3  X=1, Y=4, X+Y=5  
4  X=2, Y=3, X+Y=5  
5  X=3, Y=2, X+Y=5  
6  X=4, Y=1, X+Y=5  
7  X=5, Y=0, X+Y=5  
8  true.
```

Listing 3: output