Tarea Independiente 27/10/2025

David Núñez Franco

October 30, 2025

Inventario de Conceptos Claves

- Patrones para repeticiones o "loops" de iteradores (generador/consumidor) y backtracking
- Metapredicados
- forall
- findall
- _+ (not)
- call, apply
- Haciendo un metapredicado como _+ usando call
- maplist, includes/excludes, foldl

Ejercicio 0

Operador ternario: Escriba if_then_else(?C, ?T, ?E): si C es true llama a T en otro ca

0.1 Solución

```
% if_then_else(+Cond, +Then, +Else)
% Ejecuta Then si Cond tiene exito una vez, de lo contrario
   ejecuta Else.
if_then_else(C, T, _E) :-
    call(C),
    !,
    call(T).
if_then_else(_, _, E) :-
    call(E).
```

```
?- [if_then_else].
true.

?- if_then_else(true, writeln('yes'), writeln('no')).
yes
true.

?- if_then_else(fail, writeln('yes'), writeln('no')).
no
true.
```

Listing 1: output

Ejercicio 1

1) Escriba una versión propia recursiva de maplist: map(?G, +L, -M) que hace lo mism

Solución

```
% map(+Goal, +ListIn, -ListOut)
% Aplica Goal a cada elemento de ListIn, produciendo ListOut
   .
map(_, [], []). % caso base, lista vacia produce lista vacia
map(G, [X|Xs], [Y|Ys]) :- % aplica Goal a la cabeza y
   recursivamente al resto
call(G, X, Y),
map(G, Xs, Ys).


% predicado para las pruebas: duplica un numero
double(X, Y) :-
Y is X * 2.

% Ejemplo de uso:
:- writeln('prueba map casero'),
map(double, [1,2,3,4], R1),
writeln(R1).
```

```
?- [map].
prueba map casero
[2,4,6,8]
true.
```

Listing 2: output

Ejercicio 2

Lo mismo que 1) pero para include y exclude.

Solución

```
/*
* include(+Goal, +ListIn, -ListOut)
* Conserva los elementos de ListIn para los cuales Goal(X) tiene exito.
*/
include(_, [], []). % caso base: lista vacia produce lista vacia
include(G, [X|Xs], [X|Ys]) :- % si el primer elemento X cumple la condicion
call(G, X), !, % se llama al predicado Goal con X, y si tiene exito,
include(G, Xs, Ys). % se sigue procesando el resto de la lista
include(G, [_|Xs], Ys) :- % si el primer elemento no cumple la condicion
include(G, Xs, Ys). % se omite y se sigue con el resto

/*
* exclude(+Goal, +ListIn, -ListOut)
* Elimina los elementos de ListIn para los cuales Goal(X) tiene exito.
*/
exclude(_, [], []). % caso base: lista vacia produce lista vacia
exclude(G, [X|Xs], Ys) :- % si el primer elemento X cumple la condicion
call(G, X), !, % se llama al predicado Goal con X, y si tiene exito,
exclude(G, Xs, Ys). % el elemento se descarta y continua.
exclude(G, [X|Xs], [X|Ys]) :- % si el primer elemento no cumple la condicion
exclude(G, Xs, Ys). % se conserva y se sigue con el resto

% ejemplo de uso
par(X) :-
0 is X mod 2.

:- writeln('--- pruebas ---'),
include(par, [1,2,3,4,5,6], R1), writeln(R1),
exclude(par, [1,2,3,4,5,6], R2), writeln(R2).
```

```
?- [include_exclude].
--- pruebas ---
[2,4,6]
[1,3,5]
true.
```

Listing 3: output

Ejercicio 3

Usando foldl calcule el máximo de una lista: maximun(+L, -M): M es el máximo de la lis

Solución

```
/*
maximum(+List, -Max)
Usa foldl para recorrer la lista y eoncontrar el maximo valor.
Si la lista esta vacia, falla.
*/

% Se usa el primer elemento como inicial del acumulador
maximum([H|T], M) :-
foldl(max_acum, T, H, M).

/*
max_acum(+Elem, +Acum, -NuevoAcum)
Compara el elemento actual con el acumulador y guarda el mayor.
*/
max_acum(X, A, A) :- % Si el elemento es menor o igual al acumulador, se mantiene el acumulador
X =< A,
!.
max_acum(X, _, X). % si el elemento es mayor, el nuevo acumulador es el elemento

% ejemplo
:- writeln('--- prueba ---'),
maximum([3,7,2,9,5], M1), writeln(M1),
maximum([-10,-3,-50,-1], M2), writeln(M2),
writeln('--- fin ---').
```

```
?- [max].
--- prueba ---
9
-1
--- fin ---
true.
```

Listing 4: output