



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

Programación Lógica

18 Junio de 2025

Paradigmas de Lenguajes de Programación

Integrante	LU	Correo electrónico
Solana Navarro	906/22	solanan3@gmail.com
Melli, Tomás Felipe	371/22	tomas.melli1@gmail.com
Lourdes Wittmund Montero	1103/22	lourdesmonterochiara@gmail.com
Marco Romano Fina	1712/21	marcoromanofinaa@gmail.com



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (+54 +11) 4576-3300

<http://www.exactas.uba.ar>

Índice

0.1	Ejercicio 1 : sublista/4	2
0.2	Ejercicio 2 : tablero/2	2
0.3	Ejercicio 3 : tamaño/3	2
0.4	Ejercicio 4 : coordenadas/2	2
0.5	Ejercicio 5 : k-piezas/2	2
0.6	Ejercicio 6 : seccionTablero/5	2
0.7	Ejercicio 7 : ubicarPieza/2	2
0.8	Ejercicio 8 : ubicarPiezas/3	3
0.9	Ejercicio 9 : llenarTablero/3	3
0.10	Ejercicio 10 : medición	3
0.11	Ejercicio 11 : optimización	4
0.12	Ejercicio 12 : reversibilidad	4

0.1 Ejercicio 1 : sublista/4

Este predicado tiene las siguientes características de instanciación : **sublista(+Descartar, +Tomar, +L, -R)**. Debe ser cierto cuando **R** es la sublista de longitud **Tomar** luego descartar los primeros **Descartar** elementos de **L**. Para lograrlo, primero instanciamos **A** y **B**, **A** es la lista descartada de longitud **D**, para asegurarnos de eliminar los primeros **D** elementos, luego **B** es la lista restante, en la cual la volvemos a dividir en dos partes, nos quedamos con la primera ya que es la que nos importa y le restringimos a que contenga los primeros **T** elementos.

0.2 Ejercicio 2 : tablero/2

Este predicado tiene las siguientes características de instanciación : `tablero(+K,-T)`. Debe generara un tablero vacío de $K > 0$ columnas. Será una matriz de $5 \times K$ representada como lista de filas (cada casilla será una variable no instanciada diferente). Para lograrlo, primero instanciamos `tablero` con 5 filas, luego con `longitud_k_lista` genera listas con elementos no instanciados de tamaño K, y luego con `maplist` nos aseguramos que todas sublista de T van a cumplir el predicado `longitud_k_lista`.

0.3 Ejercicio 3 : tamaño/3

Este predicado tiene las siguientes características de instanciación : **tamaño(+M, -F, -C)** . Este predicado será verdadero cuando M tenga F filas y C columnas. Para lograrlo, primero conseguimos el F, tal que M tenga F filas, y luego nos aseguramos con maplist que cada fila contenga C columnas.

0.4 Ejercicio 4 : coordenadas/2

Este predicado tiene las siguientes características de instanciación : `coordenadas(+T, -IJ)`. Debe ser verdadero para todo par IJ que sea una coordenada de elementos del tablero. Tenemos restricciones para los índices. $I \in [1, 2, 3, 4, 5]$ y $J \in [1, \dots, K]$. Para lograrlo, primero tamaño nos instancia la cantidad de filas A y columnas B, luego nos aseguramos que I este entre 1 y A y luego que J este entre 1 y B, instanciando todas las posibles combinaciones entre I y J.


0.5 Ejercicio 5 : k-piezas/2

Este predicado tiene las siguientes características de instanciación : $kPiezas(+K, -PS)$. Debe ser verdadero cuando PS es una lista de longitud K de indentificadores de piezas, no debe repetir soluciones. Para lograrlo, En primer lugar instanciamos las piezas en L y utilizamos un predicado auxiliar para generar todas las permutaciones sin repetidos posibles. Este predicado cuenta con dos reglas, utilizar el primer elemento de la lista o no utilizarlo y con esto nos aseguramos de no utilizar de mas un elemento. En cada paso chequeamos que la cantidad restante de elementos en la lista sea mayor o igual a la cantidad de elementos que nos queda por agregar.

0.6 Ejercicio 6 : seccionTablero/5

Este predicado tiene las siguientes características de instanciación: `seccionTablero(+T,+ALTO, +ANCHO, +coordenada(I,J), ?ST)` . Debe ser verdadero cuando `ST` sea una sección de tamaño `ALTO`×`ANCHO` del tablero `T` a partir de la coordenada `IJ`. Para lograrlo, en primer lugar recortamos las filas con el predicado `sublista`. Esto nos devuelve un tablero con todas las filas que nos interesa seccionar. Luego nos aseguramos que la longitud de cada una de estas filas (cantidad de columnas) sea del tamaño pedido.

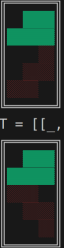

```
12 ?- tablero(3,T),pieza(e,E),tamaño(E,F,C), seccionTablero(T,F,C,(1,1),E), mostrar(T)
```



```
T = [[_A, e, e], [e, e, e], [_ , _ ], [_ , _ ], [_ , _ ]],
E = [[_A, e, e], [e, e, e]],
F = 2,
C = 3 ;
```


0.7 Ejercicio 7 : ubicarPieza/2

Este predicado tiene las siguientes características de instanciación : `ubicarPieza(+Tablero, +Identificador)`. Debe generar todas las posibles ubicaciones de la pieza en el tablero. Para lograrlo, en primer lugar instanciamos la pieza a ubicar y todas las coordenadas posibles del tablero, conseguimos el tamaño de la pieza y apartir de eso usamos el predicado `seccionTablero` para encontrarle su lugar.

```
13 ?- tablero(3,T),ubicarPieza(T,e),ubicarPieza(T,j),mostrar(T).

T = [[_, e, e], [e, e, e], [_, _, j]], [_, j, j], [j, j, _]] ;

T = [[_, e, e], [e, e, e], [j, j, _]], [_, j, j], [_, _, j]] .
```



0.8 Ejercicio 8 : ubicarPiezas/3

Este predicado tiene las siguientes características de instanciación : `ubicarPiezas(+Tablero, +Poda, +Identificadores)`. Este predicado debe listar todas las posibles opciones de ubicar todas las piezas mencionada en `Identificadores`. Para lograrlo, teniendo instanciado el tablero, una poda y una lista de ID de piezas lo que hacemos es acertar que para cada pieza podemos ubicarla en un tablero. Para lograrlo hacemos uso del predicado auxiliar *ubicar_con_poda* que nos garantiza que la pieza ubicada cumpla la poda.

```
14 ?- tablero(3,T), ubicarPiezas(T, sinPoda, [e,j]), mostrar(T).

T = [[_, e, e], [e, e, e], [_, _, j]], [_, j, j], [j, j, _]] .
```

0.9 Ejercicio 9 : llenarTablero/3

Este predicado tiene las siguientes características de instanciación : `llenarTablero(+Poda, +Columnas, -Tablero)`. Debe enumerar todas las distintas formas de llenar un tablero con la cantidad de columnas (y piezas) indicada. Para lograrlo, en primer lugar instanciamos el tablero y luego buscamos las permutaciones sin repetidos de C piezas (pedimos C piezas ya que no hace falta mas de C piezas para completar un tablero de C columnas debido a las dimensiones) para luego ubicarlas en un tablero utilizando el predicado `ubicarPiezas`.

```
15 ?- llenarTablero(sinPoda,3,T), mostrar(T).

T = [[a, a, b], [a, b, b], [a, h, b]], [a, h, b], [h, h, h]] ;

T = [[h, h, h], [b, h, a], [b, h, a], [b, b, a], [b, a, a]] .
```

0.10 Ejercicio 10 : medición

Nos dan un predicado definido para poder medir las distintas soluciones según la `Poda`, `Columnas`, `N` que con uso del predicado `time` podremos medir el tiempo de ejecución como se ve a continuación. Definimos el siguiente predicado para medir:

```
1 tiempos(Poda, N) :- between(3, 4, K), time(cantSoluciones(Poda, K, N)).
```

Obtuvimos los siguientes resultados

```
1 ?- tiempos(sinPoda, N).
37,958,869 inferences, 1.922 CPU in 1.928 seconds (100% CPU, 19750001 Lips)
N = 28 ;
1,485,726,353 inferences, 77.268 CPU in 77.476 seconds (100% CPU, 19228252 Lips)
N = 200.
```

0.11 Ejercicio 11 : optimización

Tenemos que definir una estrategia de poda `podaMod5`. Para lograrlo, en primer lugar utilizamos el predicado **findall** para encontrar todas las coordenadas libres del tablero. Luego agrupamos estas posiciones (con el predicado `agrupar`) de manera que podamos garantizar que estos grupos de coordenadas vecinas sean de tamaño 5 y así cumplir con la poda pedida. Los tiempos obtenidos con esta poda son :

```
1 ?- tiempos(podaMod5, N).
16,181,606 inferences, 0.901 CPU in 0.934 seconds (96% CPU, 17959089 Lips)
N = 28 ;
329,335,525 inferences, 18.441 CPU in 19.551 seconds (94% CPU, 17858846 Lips)
N = 200.
```

0.12 Ejercicio 12 : reversibilidad

Nos piden hacer un análisis de reversibilidad del predicado `sublista/4`. En particular, si es reversible en el primer y cuarto argumento. Probar reversibilidad en `sublista(+D,+T,+L,-R)` sobre D y R. Para probarlo, nos preguntamos si teniendo la instanciación opuesta, produce un resultado adecuado.

Caso `sublista(-D,+T,+L,-R)`

Como `length(?List, ?Int)` es reversible, entonces `length(A, D)` funciona adecuadamente instanciando en D en run-time.

Zero

```
?- sublista(X, 5, [a,b,c,d], R).
false.
```

One

```
?- sublista(X, 4, [a,b,c,d], R).
X = 0,
R = [a, b, c, d] ;
false.
```

Many

```
?- sublista(X, 2, [a,b,c,d], R).
X = 0,
R = [a, b] ;
X = 1,
R = [b, c] ;
X = 2,
R = [c, d] ;
false.
```

Caso `sublista(+D,+T,+L,+R)`

Como `length(?List, ?Int)` y `append(?List1, ?List2, ?List1AndList2)` son completamente reversibles, aceptan todos sus parámetros instanciados. En este caso, podemos decir que es reversible ya que en `[append(R,-,B), length(R,T).]` `append/3` cumple: `append(?List1, ?List2, ?List1AndList2)`. `List1AndList2` es la concatenación de `List1` y `List2`. Por tanto, nos garantiza reversibilidad, y `length` lo mismo como ya vimos.

Zero

```
?- sublista(0, 5, [a,b,c,d], [a,b,c,d]).
false.
```

One

```
?- sublista(0, 4, [a,b,c,d], [a,b,c,d]).
true ;
false.
```

Caso sublista(-D,+T,+L,+R)

El caso en que ambas están en su opuesta instanciación.

Zero

```
?- sublista(X, 5, [a,b,c,d], [a,b,c,d]).  
false.
```

One

```
?- sublista(X, 4, [a,b,c,d], [a,b,c,d]).  
X = 0 ;  
false.
```