# Trabajo Práctico 2
# Programación Lógica

Paradigmas de Lenguajes de Programación — $2^o$ cuat. 2011

Fecha de entrega: 27/10/2011

Este trabajo consiste en implementar en Prolog un programa que resuelva tableros del juego llamado *trabado* [1].

La solución debe realizarse en un solo archivo Prolog. Pueden utilizarse todos los predicados y metapredicados existentes.

## Introducción

El juego del *trabado* consiste en un tablero casi completo de piezas. El objetivo es llevar una pieza distinguida hacia un lugar, también distinguido. Dado que el tablero suele estar casi completo (de allí el nombre del juego), para lograr el objetivo es necesario desplazar de forma ingeniosa las piezas que se encuentran entre la pieza objetivo y la posición distinguida.

Existen distintos tipos de piezas: horizontal, vertical, unidad y objetivo. Las horizontales y verticales sólo pueden moverse así como su nombre lo indica. El resto tiene libertad de moverse en las cuatro direcciones.

De esta forma, representaremos el juego del *trabado* mediante un tablero de un determinado tamaño, una lista de piezas y la posición distinguida. A su vez, cada pieza es de algún tipo y tiene asociada una posición dentro del tablero.

En Prolog esto se representará mediante los siguientes términos:

- `Tam ::= tam(NFils, NCols)` – tamaño del tablero o de una pieza.
- `TipoPieza ::= objetivo | unidad | horizontal | vertical` – tipo de pieza.
- `Pos ::= pos(Fil, Col)` – posición dentro del tablero.
- `Dirección ::= norte | este | sur | oeste` – dirección de movimiento.
- `Pieza ::= pieza(TipoPieza, Pos)` – para las piezas que ocupan más de una casilla, `Pos` corresponde a su coordenada superior izquierda.
- `Tablero ::= tablero(Tam, PosObjetivo, Piezas)` – representa el tablero; `PosObjetivo` es la posición donde queremos colocar a la pieza objetivo (es decir, su esquina superior izquierda), y `Piezas` es una lista de piezas.

[1] Ver: `http://www.stavanger-guide.com/klotski.htm`

Con el objetivo de tener una representación única para el tablero, se agregan las siguientes restricciones a la lista de piezas:

- Debe haber una pieza `objetivo` y sólo una.
- Las piezas deben estar ordenadas de forma lexicográfica utilizando `Tipo, Fil, Col`.[2] Es decir: `pieza(Tipo1, pos(F1, C1))` $<$ `pieza(Tipo2, pos(F2, C2))` si y sólo si
  - `Tipo1` $<$ `Tipo2`;
  - o bien `Tipo1 = Tipo2` y `F1` $<$ `F2`;
  - o bien `Tipo1 = Tipo2`, `F1 = F2` y `C1` $<$ `C2`.

  Los tipos de piezas se ordenan alfabéticamente:
  `horizontal` $<$ `objetivo` $<$ `unidad` $<$ `vertical`
- Cada pieza tiene un tamaño fijo y está dado de la siguiente manera:
  - `objetivo: tam(2, 2)`
  - `unidad: tam(1, 1)`
  - `horizontal: tam(1, 2)`
  - `vertical: tam(2, 1)`

Por ejemplo un tablero posible puede ser así:

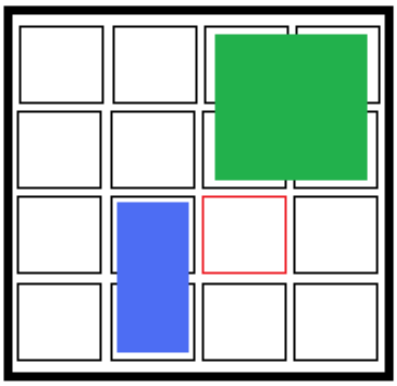

Figura 1: Tablero 1

Donde la pieza de color verde es la pieza objetivo, la azul es una pieza vertical, y la posición objetivo (3, 3) está marcada en rojo. Este tablero se representa como:
`tablero(tam(4, 4), pos(3, 3), [pieza(objetivo, pos(1, 3)), pieza(vertical, pos(3, 2))])`.

[2] Notar que este es el orden que SWI-Prolog nativamente asigna a los términos, y por lo tanto se puede usar el predicado `sort/2` para obtener la lista en el orden deseado.

## Ejercicios

Definir los siguientes predicados:

### Ejercicio 1

tamano(?TipoPieza, tam(?Alto, ?Ancho)) que enumera todos los tipos de piezas y sus tamaños.

### Ejercicio 2

movimiento_posible(+TipoPieza, −Direccion) que enumera todas las direcciones a las que se puede mover un tipo de pieza.

### Ejercicio 3

mover(+Pos, +Direccion, −Pos) que mueve la posicion dada según la dirección.
Por ejemplo

```
?– mover(pos(2, 3), norte, X).
X = pos(1, 3) ;

?– mover(pos(2, 3), sur, X).
X = pos(3, 3) ;

?– mover(pos(2, 3), este, X).
X = pos(2, 4) ;

?– mover(pos(2, 3), oeste, X).
X = pos(2, 2).
```

### Ejercicio 4

en_tablero(+Tablero, ?Pos) que indica si una pos esta dentro de las coordenadas del tablero. Por ejemplo:

```
?– en_tablero(tablero(tam(3, 3), _, _), P).
P = pos(1, 1) ;
P = pos(1, 2) ;
P = pos(1, 3) ;
P = pos(2, 1) ;
P = pos(2, 2) ;
P = pos(2, 3) ;
P = pos(3, 1) ;
P = pos(3, 2) ;
P = pos(3, 3).
```

### Ejercicio 5

pieza_ocupa(+Pieza, −Pos) indica si la pieza ocupa la posición `Pos`.

```
?– pieza_ocupa(pieza(vertical, pos(1, 2)), P).
P = pos(1, 2) ;
P = pos(2, 2).
```

### Ejercicio 6

quitar(?X, +L, −R) saca el elemento `X` de la lista `L`, obteniendo la lista `R`. Si `X` está instanciado, se asume que ocurre exactamente una vez en `L`. Resolverlo sin utilizar recursión[3].

```
?– quitar(X, [1, 2, 3], R).
X = 1, R = [2, 3] ;
X = 2, R = [1, 3] ;
X = 3, R = [1, 2]
```

### Ejercicio 7

movimiento_valido(+Tablero, −Pieza, −Dir) que sea verdadero sii la pieza `Pieza` es un elemento del tablero que puede moverse hacia la dirección `Dir`, quedando dentro de las dimensiones del tablero y sin "pisar" a las demás piezas.

### Ejercicio 8

mover_pieza(+Tablero1, +Pieza, +Dir, −Tablero2) debe ser verdadero sii `Tablero2` es el `Tablero1` después de mover la pieza `Pieza` hacia la dirección dada. Asumir que el movimiento es válido. Por ejemplo, si `T0` es el tablero de la figura 1:

```
?– mover_pieza(T0, pieza(objetivo, pos(3, 1)), sur, T).
T = tablero(tam(4, 4), pos(3, 3), [pieza(objetivo, pos(2, 3)), pieza(vertical, pos(3, 2))]) ;
false
```

Tener en cuenta que es necesario mantener el invariante del tablero. En particular, las piezas del tablero deben quedar ordenadas. Asegurarse de que el predicado no devuelva soluciones repetidas (ni distintas).

### Ejercicio 9

resolver (+Tablero, −MovimientosDePiezas, −TableroFinal) encuentra, si es posible, la traza de movimientos que conducen a un tablero final. La lista de movimientos es una lista de tuplas de la forma `(pieza(Tipo, pos(X, Y)), Dir)`. Por ejemplo:

[3] Por ejemplo, usando el predicado `append/3`.

```
?- problema(t0, Tablero), resolver(Tablero, _, Final),
   mostrar(Tablero), mostrar(Final).

    1  2  3  4
 1  -  a  a  -
 2  c  d {e}{f}
 3  b  b {-}{-}
 4  b  b  -  -

    1  2  3  4
 1  -  c  d  e
 2  -  - {b}{b}
 3  -  - {b}{b}
 4  -  a  a  f

Tablero = ...
Final = ...
```

### Ejercicio 10

armar_tablerosA(?Tablero) que dado un tablero con las posiciones de las piezas parcialmente instanciadas, enumere todas las posibles instanciaciones quedándose únicamente con aquellas que sean resolubles.

Se busca hacer una implementación simple que complete las posiciones de las piezas del tablero con alguno de los valores posibles y al final se verifique si cumple con el invariante y además es resoluble. (En este caso, la parte importante del invariante es que las piezas no se encimen unas con otras).

### Ejercicio 11

armar_tablerosB(?Tablero) – en esta solución se intentará mejorar el recorrido del espacio de búsqueda verificando en cada paso que se cumpla el invariante del tablero generado.

## 1. Predicados y metapredicados útiles

Además del archivo `solucion.pl` (a completar por el grupo), se proveen dos archivos adicionales. En `mostrar.pl` se encuentra el predicado `mostrar(+Tablero)` que imprime un tablero en la pantalla. En el archivo `tests.pl` hay algunos ejemplos de problemas para resolver (`t...`) y para armar (`p...`). Otros predicados que se pueden usar:

- var(?X) y nonvar(?X). Observar que nonvar(X) es equivalente a not(var(X)).

  ```
  ?– var(X).
  true.

  ?– var(42).
  fail.
  ```

```
?- X = 42, var(X).
fail.
```

- between(+Low, +High, −Value).

```
?- between(100, 102, X).
X = 100 ;
X = 101 ;
X = 102.
```

- member(−Elem, +List).

```
?- member(X, [cero, uno, dos, tres]).
X = cero ;
X = uno ;
X = dos ;
X = tres.
```

- append(?List1, ?List2, ?List3).

```
?- append([1, 2, 3], [4, 5], A).
A = [1, 2, 3, 4, 5]

?- append([1, 2, 3], W, [1, 2, 3, 4, 5]).
W = [4, 5]
```

- setof(+Template, +Goal, −Set).

```
?- setof(X, between(100, 102, X), L).
L = [100, 101, 102].
```

- forall (:Cond, :Action).

```
?- forall(member(X, [2, 4, 6, 8]),
          X mod 2 =:= 0).
true.
```

## Pautas de entrega

Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado debe contar con un comentario donde se explique su funcionamiento. Cada predicado asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección `plp-docentes@dc.uba.ar`. Dicho mail debe cumplir con el siguiente formato:

- El título del mensaje debe ser "[PLP;TP-PL]" seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el e-mail y lo debe hacer en forma de **archivo adjunto** (puede adjuntarse un .zip o .tar.gz).

El código debe poder ser ejecutado en `SWI-Prolog`. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado.

Los objetivos a evaluar en la implementación de los predicados son:

- Corrección.
- Declaratividad.
- Reuso de predicados previamente definidos.
- Uso de unificación, backtracking y reversibilidad de los predicados que correspondan.

- Salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

  **Importante**: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.