Trabajo Práctico 2
Programación Lógica

Paradigmas de Lenguajes de Programación — 1^o cuat. 2009

Fecha de entrega: 9 de junio

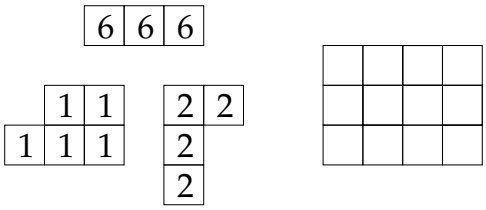
1. Introducción

Este trabajo consiste en programar en Prolog un conjunto de predicados para resolver un rompecabezas vagamente reminiscente al tetris o al *pentomino puzzle*, sujeto a algunas restricciones adicionales.

1.1. Descripción del problema

Se tiene una cuadrícula de $n \times m$ posiciones, y un conjunto de piezas, formadas por cuadraditos contiguos (como piezas de tetris). Cada cuadradito de cada pieza tiene asociado un peso. Todos los cuadraditos de una pieza dada pesan lo mismo. Dado un conjunto de piezas, el problema es encontrar alguna manera de ubicarlas todas en el tablero, sin encimarlas.

Por ejemplo, si se tienen las piezas y el tablero que se muestran en la figura,



la siguiente es una manera posible de ubicarlas:

2	2	1	1
2	1	1	1
2	6	6	6

Además de usar todas las piezas, se impone como restricción adicional que las filas y las columnas del tablero tengan un cierto peso. En el ejemplo, el peso de la primera fila es 6, y el peso de la tercera columna es 8. Estas

restricciones formarán parte de la entrada, de la manera que se explica más abajo.

La forma sugerida de resolver el ejercicio es utilizando la técnica *generate and test*: se generan todas las formas de ubicar las piezas dadas en el tablero (posibles "candidatas" a solución) y se filtran, devolviendo solamente aquellas que cumplan las restricciones sobre el peso de las filas y de las columnas.

2. Tableros y juegos

Un tablero se representa con una lista de filas. Cada fila es, a su vez, una lista de elementos. Un juego es un tablero que contiene, además, restricciones sobre las filas y sobre las columnas, y se representa con un término de la forma juego(Tablero, RF, RC), donde Tablero es un tablero, y RF y RC son listas. La lista RF contiene un elemento por fila, que determina cuál debe ser el peso de esa fila. La lista RC es similar, pero para las columnas. Se puede asumir que la entrada está bien formada, es decir que todas las filas miden lo mismo, y los tamaños de RF y RC son los correctos.

En el archivo `mostrar.pl` provisto por la cátedra, se dispone de predicados mostrarJuego(+Juego) y mostrarTablero(+Tablero) que imprimen un juego o un tablero en la pantalla. El siguiente ejemplo corresponde a un juego ya terminado:

```
?- mostrarJuego(juego([[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]],
                      [3, 6, 9, 12],
                      [10, 10, 10])).
```

```
    |  10  10  10
----+--------------
  3 |   1   1   1
  6 |   2   2   2
  9 |   3   3   3
 12 |   4   4   4
```

Para indicar que una posición del tablero está vacía, se colocan variables no instanciadas en esa posición. A medida que se va llenando el tablero, estas variables se van instanciando con los valores finales. Inicialmente, un tablero puede estar totalmente vacío, o puede contener "obstáculos", que no pueden ser ocupados con piezas. Algo similar ocurre con las restricciones: pueden estar dadas de antemano (si son números), o pueden ser variables no instanciadas si no se desea restringir el valor de la suma de la fila o de la columna.

Por ejemplo, el siguiente tablero tiene dos lugares ya ocupados, que no pueden "pisarse" con piezas. Además, sólo hay restricciones sobre la segunda fila y sobre la tercera columna.

```
?- mostrarJuego(juego([[_, _, _], [_, 2, _], [_, 2, _], [_, _, _]],
                      [_, 7, _, _],
                      [_, _, 3])).

    |  --  --   3
----+------------
--  |  --  --  --
 7  |  --   2  --
--  |  --   2  --
--  |  --  --  --
```

En Prolog, puede diferenciarse entre un valor que todavía no fue instanciado y un valor que ya se encuentra instanciado utilizando los predicados var (...) y nonvar (...) .

Ejercicio 1

Escribir los siguientes predicados:

- dimension(+Tablero, −N, −M), que es verdadero si y sólo si el tablero tiene N filas y M columnas. (Puede asumirse que el tablero está bien formado).
- posicion(+Tablero, −I, −J), que es verdadero si y sólo si la fila I y la columna J determinan una posición válida del tablero.

◇

Por ejemplo:

```
?- posicion([[1, 2, 3], [4, 5, 6]], I, J).
I = 0, J = 0 ;
I = 0, J = 1 ;
I = 0, J = 2 ;
I = 1, J = 0 ;
I = 1, J = 1 ;
I = 1, J = 2.
```

Ejercicio 2

Escribir el predicado subtablero(+Tablero, +I, +J, −Subtablero), que es verdadero si y sólo si Subtablero contiene los mismos elementos que Tablero, empezando desde la fila I y la columna J. Se puede asumir que I y J están en el rango correcto. ◇

Por ejemplo:

```
?– Tablero = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]],
   subtablero(Tablero, 2, 1, Subtablero),
   mostrarTablero(Subtablero).
10  11  12
14  15  16
Subtablero = [[10, 11, 12], [14, 15, 16]] .
```

Ejercicio 3

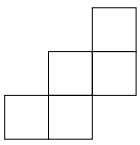
Escribir el predicado transponer(+Tablero, −Transpuesto), que es verdadero si y sólo si Transpuesto es el tablero Tablero transpuesto. ◇

Por ejemplo:

```
?– Tablero = [[1, 2, 3], [4, 5, 6]],
   transponer(Tablero, Transpuesto),
   mostrarTablero(Transpuesto).
 1   4
 2   5
 3   6
Transpuesto = [[1, 4], [2, 5], [3, 6]] ;
```

3. Siluetas y piezas

Hasta esta sección no se mencionó la posibilidad de que las piezas estén orientadas (mediante rotaciones). Para simplificar la programación, si una pieza tiene varias orientaciones, deben estar especificadas explícitamente como parte de la entrada. Llamamos *silueta* a un "molde" que dice qué posiciones están ocupadas. En particular, una pieza se representa de igual manera que un tablero, como una lista de filas. La posiciones vacías se representan con variables no instanciadas. Las posiciones ocupadas se representan con algún átomo (por ejemplo, x). Eventualmente el átomo puede ser un número que indica el peso del cuadradito. Por ejemplo:



Se representa con la siguiente silueta:

```
?– Silueta = [[_, _, x], [_, x, x], [x, x, _]], mostrarTablero(Silueta).
__  __   x
__   x   x
 x   x  __
```

Ejercicio 4

Escribir el predicado asignarPeso(+Silueta, +Peso, −SiluetaConPeso), que sea verdadero si y sólo si SiluetaConPeso es una silueta de la misma forma que Silueta, donde cada posición ocupada se reemplaza por el número Peso. ◇

Por ejemplo:

```
?- Silueta = [[_, _, x], [_, x, x], [x, x, _]],
   asignarPeso(Silueta, 10, SiluetaConPeso),
   mostrarTablero(SiluetaConPeso).
__  __  10
__  10  10
10  10  __
```

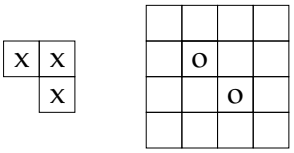
Ejercicio 5

Escribir el predicado ubicarSilueta (+Silueta, −I, −J, −Tablero), que sea verdadero si y sólo si la silueta Silueta puede ubicarse en la fila I y en la columna J del tablero Tablero, sin pasarse de los bordes y sin "pisar" posiciones ya ocupadas. Si hay más de una posición (I, J) en la que pueda ubicarse la silueta, el predicado debe devolverlas todas. El predicado recibe un tablero parcialmente instanciado: debe ser una lista de las dimensiones adecuadas, pero puede contener algunas posiciones vacías (i.e. variables no instanciadas). Además, cada vez que el predicado tenga éxito, deben instanciarse las variables del tablero que pasaron a encontrarse ocupadas por la silueta. ◇

Por ejemplo, considerando las siguientes definiciones,

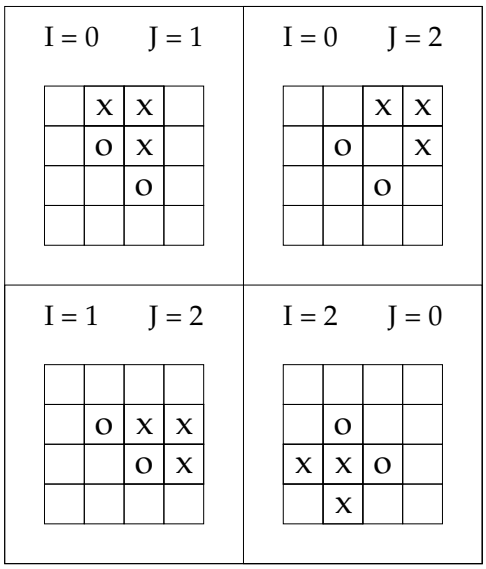
```
tablero1([
  [_, _, _, _],
  [_, o, _, _],
  [_, _, o, _],
  [_, _, _, _]
]).

silueta1([
  [x, x],
  [_, x]
]).
```

que representan la silueta y el tablero en la figura:



deben devolverse las siguientes soluciones.

?– tablero1(T), silueta1(S), ubicarSilueta(S, I, J, T), mostrarTablero(T).

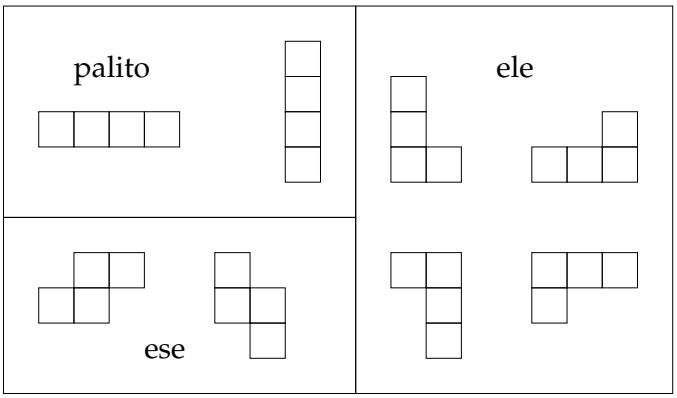


Observar que **no** puede ocuparse una posición del tablero que ya se encuentre ocupada, **aunque** el contenido de esa posición coincida con el contenido del cuadradito de la silueta que se desea ubicar.

Por otra parte, una *pieza* es un conjunto de siluetas. La intención es que las siluetas asociadas a una pieza representen una posible orientación de la pieza (por ejemplo, la pieza rotada 90 grados). Las piezas se representan usando una lista de orientaciones. Cada elemento de la lista es de la forma orientacion(NombrePieza, Silueta). Por ejemplo el siguiente predicado, algunasPiezas(−Piezas):

```
algunasPiezas([
  orientacion(palito, [[x, x, x, x]]),
  orientacion(palito, [[x], [x], [x], [x]]),
  orientacion(ele, [[x, _], [x, _], [x, x]]),
  orientacion(ele, [[_, _, x], [x, x, x]]),
  orientacion(ele, [[x, x], [_, x], [_, x]]),
  orientacion(ele, [[x, x, x], [x, _, _]]),
  orientacion(ese, [[_, x, x], [x, x, _]]),
  orientacion(ese, [[x, _], [x, x], [_, x]])
]).
```

es verdadero para un diccionario de piezas que representa lo que se ilustra a continuación:



Ejercicio 6

Escribir el predicado ubicarPieza(+NombrePieza, +Peso, +DiccPiezas, −I, −J, −Tablero), que sea verdadero si y sólo si la pieza identificada por el nombre NombrePieza se puede orientar de alguna manera y ubicarse en la fila I y en la columna J del tablero Tablero, haciendo que cada cuadradito de la silueta pese Peso. Si hay más de una orientación y una posición en la que se pueda ubicar, se deben devolver todas las soluciones posibles. ◇

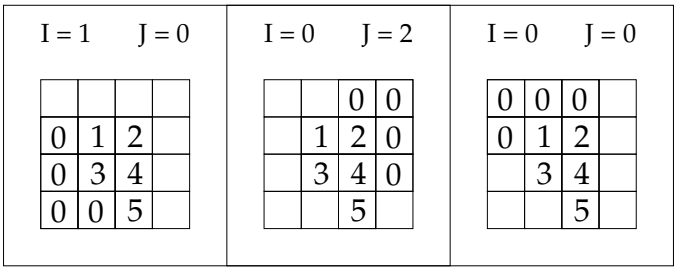
Por ejemplo, con esta definición de tablero2,

```
tablero2([
  [_, _, _, _],
  [_, 1, 2, _],
  [_, 3, 4, _],
  [_, _, 5, _]
]).
```

la consulta:

```
?- algunasPiezas(DiccPiezas),
   tablero2(Tablero),
   ubicarPieza(ele, 0, DiccPiezas, I, J, Tablero),
   mostrarTablero(Tablero).
```

debe devolver las siguientes soluciones.



4. Solución del rompecabezas

Ejercicio 7

Escribir el predicado solucionValida(−Juego), que sea verdadero si y sólo si los números ubicados en el tablero cumplen las restricciones de filas y columnas. El juego estará parcialmente instanciado: todos los elementos del tablero deben ser números, pero las restricciones de filas y columnas pueden estar no completamente instanciadas. ◇

Por ejemplo:

```
?- Juego = juego([[1, 2, 3], [4, 5, 6]], RF, RC),
   solucionValida(Juego), mostrarJuego(Juego).
    |   5   7   9
----+-------------
 6  |   1   2   3
15  |   4   5   6
RF = [6, 15],
RC = [5, 7, 9] ;

?- Juego = juego([[1, 2, 3], [4, 5, 6]], [6, 15], [SumaColumna1, 7, _]),
   solucionValida(Juego).
SumaColumna1 = 5 ;

?- Juego = juego([[1, 2, 3], [4, 5, 6]], [10, _], _),
   solucionValida(Juego).
fail.
```

Ejercicio 8

Escribir el predicado resolver(+DiccPiezas, +PiezasDisponibles, −Juego), que sea verdadero si y sólo si hay una manera de resolver el juego usando las piezas dadas. El conjunto de piezas disponibles se recibe en PiezasDisponibles, que es una lista de elementos de la forma pieza(NombrePieza, Peso). El predicado debe instanciar las partes del tablero y las restricciones del juego que

no se encuentren instanciadas. Si hay más de una solución, deben devolverse todas las posibles. No se deben devolver soluciones repetidas, aunque puede haber soluciones simétricas. ◇

Por ejemplo, con esta definición de juego2,

```
juego2(juego([ [1, _, _, 7],
               [_, _, _, _],
               [0, _, _, _],
               [_, _, 5, _] ],
             [_, _, _, _],
             [4, 6, 11, 16])).
```

la consulta:

```
?– juego2(Juego), algunasPiezas(DiccPiezas),
   PiezasDisponibles = [pieza(ese, 2), pieza(ele, 3), pieza(ese, 1)],
   resolver(DiccPiezas, PiezasDisponibles, Juego), mostrarJuego(Juego).
```

debería devolver dos soluciones:

```
    |   4   6  11  16
----+----------------
12  |   1   2   2   7
10  |   2   2   3   3
 5  |   0   1   1   3
10  |   1   1   5   3

    |   4   6  11  16
----+----------------
10  |   1   1   1   7
 8  |   1   1   3   3
 7  |   0   2   2   3
12  |   2   2   5   3
```

5. Predicados útiles

- 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.

- length(+List, −Int).

  ?– length([cero, uno, dos, tres], N).
  N = 4.

- member(−Elem, +List).

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

- nth0(+Index, +List, −Elem).

  ?– nth0(2, [cero, uno, dos, tres], X).
  X = dos.

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 subject 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 **attachment** (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 (por ejemplo, si hay dos piezas iguales).

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