Evaluación (1ª instancia)

Duración: 3 horas

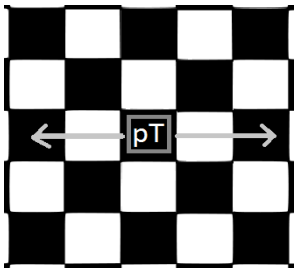
Ejercicio 1 [evaluación individual del laboratorio]

Ejercicio 2 [40 puntos]

Considere un tablero de mxn celdas y un caballo de ajedrez que busca recorrer la mayor cantidad posible de celdas, sin pasar más de una vez por una misma celda, y con la restricción de que existen otras piezas en el tablero. A las otras piezas se les llama *pseudotorres*, las cuales únicamente pueden atacar celdas de su misma fila.

Los movimientos que puede hacer un caballo son iguales que en el ajedrez:

Las pseudotorres solo tienen movimiento horizontal:



Implemente en Prolog los predicados que se describen a continuación:

a) **`celdas_atacadas(+Filas, +Columnas, +ListaTorres, ?Tablero) <-`**

Las celdas de Tablero tienen el valor *atacada* si la celda es atacada por alguna torre de ListaTorres, o el valor *libre* en caso contrario. ListaTorres es una lista de naturales entre 1 y Filas inclusive, que indica en qué filas hay pseudotorres (puede suponer que hay como máximo 1 torre por fila).

Solución:

```
celdas_atacadas(Filas, Columnas, ListaTorres, Tablero) :-
    matrix(Filas, Columnas, libre, MatrizVacia),
    marcar_horizontales(ListaTorres, Filas, Columnas, MatrizVacia, Tablero).

marcar_horizontales([X|Ts], Filas, Columnas, M, M1) :-
    marcar_h(X, Filas, Columnas, M, MInt),
    marcar_horizontales(Ts, Filas, Columnas, MInt, M1).
marcar_horizontales([], _, _, M, M).

marcar_h(X, Filas, Columnas, M, M1) :-
    Columnas >= 1,
    set_cell(M, X, Columnas, atacada, MInt),
    C1 is Columnas - 1,
    marcar_h(X, Filas, C1, MInt, M1).
marcar_h(_, _, 0, M, M).
```

b) **recorrida_caballo(+Filas, +Columnas, +X, +Y, +Tablero, -LargoRecorrida)** <-
LargoRecorrida es la cantidad de celdas que recorre un caballo comenzando en la celda (X, Y), sin ocupar en ningún momento una celda que contenga el valor *atacada* en Tablero y sin pasar más de una vez por ninguna celda, para un tablero de tamaño Filas X Columnas.

Suponga que ya cuenta con el predicado:
movimiento_caballo(+Filas, +Columnas, +X, +Y, -X1, -Y1) <-
(X1, Y1) es un movimiento válido de un caballo que se encuentra en la posición (X, Y) para un tablero de tamaño Filas X Columnas.

Solución:

```
recorrida_caballo(Filas, Columnas, X, Y, Tablero, LargoRecorrida) :-
    get_cell(Tablero, X, Y, libre),
    set_cell(Tablero, X, Y, recorrida, Tablero1),
    recorrer(1, LargoRecorrida, Tablero1, X, Y, Filas, Columnas).

recorrida_caballo(_, _, X, Y, Tablero, 0) :-
    \+ get_cell(Tablero, X, Y, libre).

recorrer(RecorridaActual, LargoRecorrida, T,X,Y,Filas,Columnas):-
    hay_movimiento_caballo(X, Y, Filas, Columnas, T, X1, Y1),
    set_cell(T, X1, Y1, recorrida, T2),
    RecorridaActual1 is RecorridaActual + 1,
    recorrer(RecorridaActual1, LargoRecorrida, T2,X1,Y1,Filas,Columnas).

recorrer(LargoRecorrida, LargoRecorrida, T, X, Y, Filas, Columnas):-
    \+ hay_movimiento_caballo(X, Y, Filas, Columnas, T, _, _).

hay_movimiento_caballo(X, Y, Filas, Columnas, T, X1, Y1) :-
    movimiento_caballo(Filas, Columnas, X, Y, X1, Y1),
    get_cell(T, X1, Y1, libre).
```

c) **todas_recorridas(+Filas, +Columnas, +X, +Y, +Tablero, -ListaLargos)** <-
ListaLargos es una lista con los largos correspondientes a todas las recorridas posibles de un caballo a partir de la posición inicial (X,Y), manteniendo las mismas restricciones que en la parte b.

Solución:

```
todas_recorridas(Filas, Columnas, X, Y, Tablero, ListaLargos) :-
    findall(Largo,
          recorrida_caballo(Filas, Columnas, X, Y, Tablero, Largo),
    ListaLargos).
```

NOTA: Asuma que cuenta con los siguientes predicados ya implementados para el manejo de matrices. La implementación interna de la matriz no es relevante, debido a que solamente utilizará los predicados para manipularla.

matrix(+M,+N,+Cell,?Matrix) ← **Matrix** es una matriz de **M** filas y **N** columnas donde cada celda tiene el valor **Cell**.

get_cell(+Matrix,+X,+Y,?Cell) ← **Cell** es el valor de la celda en la fila **X**, columna **Y** de la matriz **M**.

set_cell(+Matrix,+X,+Y,+Cell,?Matrix2) ← **Matrix2** es una matriz igual a **Matrix**, sustituyendo la celda (X,Y) por el valor **Cell**.

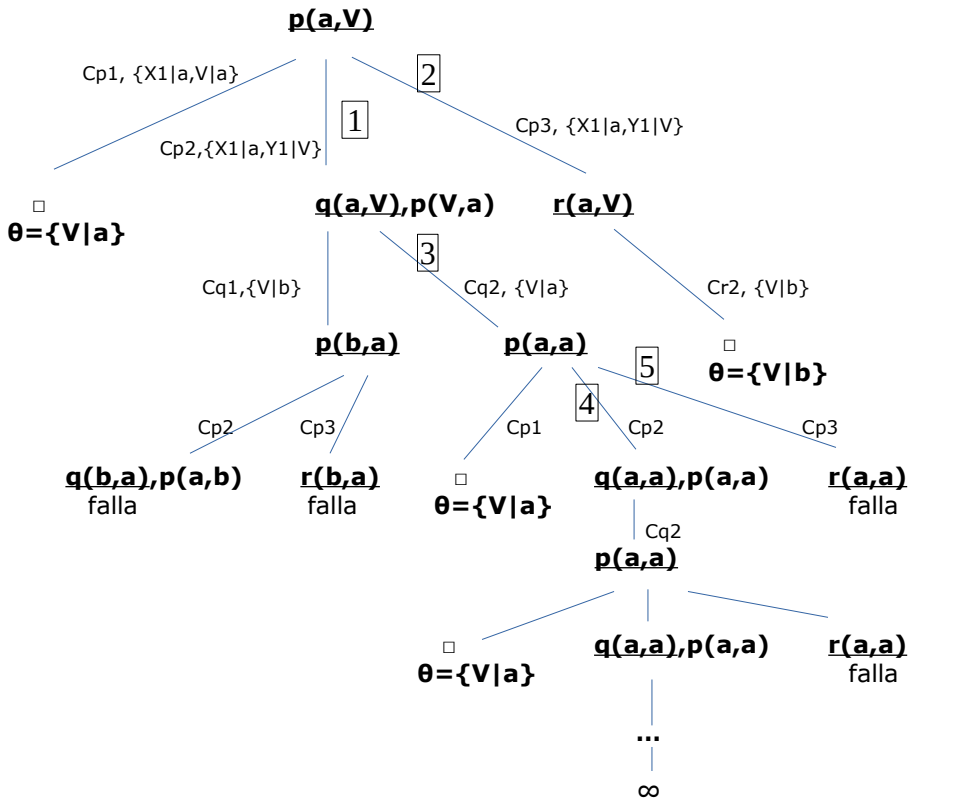
Ejercicio 3 [30 puntos]

Dado el siguiente conjunto de cláusulas para los predicados **p**, **q**, **r**:

```
Cp1:  p(X,X).
Cp2:  p(X,Y) :- q(X,Y), p(Y,X).
Cp3:  p(X,Y) :- r(X,Y).
Cq1:  q(a,b).
Cq2:  q(a,a).
Cq3:  q(b,c).
Cr1:  r(e,e).
Cr2:  r(a,b).
```

a) Dibuje el árbol SLD correspondiente a la consulta **?- p(a,V)**, suponiendo que la regla de computación toma el átomo de más a la izquierda para la resolución. Indique para cada rama de éxito cuál es la respuesta computada correspondiente.

Solución:



b) ¿Qué respuestas dará un intérprete prolog que aplique la regla de computación de la parte a), tomando las cláusulas en su orden de aparición y haciendo recorrida en profundidad del árbol?

Solución:

El intérprete dará la respuesta V=a, luego, con punto y coma, irá dando infinitas respuestas V=a. Nunca podrá dar la repsuesta V=b porque, al recorrer el árbol en profundidad, se pierde en la rama infinita.

c) Diga cuáles serían las respuestas de la parte b) si las cláusulas de **p** estuvieran en el siguiente orden:

i.
```
Cp1
Cp3
Cp2
```

ii.
```
Cp3
Cp2
Cp1
```

Solución:

i. Dará la respuesta V=a, luego con punto y coma dará la respuesta V=b (porque sería la respuesta correspondiente a la segunda rama del árbol), y si se vuelve a ingresar punto y coma, irá dando infinitas respuestas V=a (debido a la rama infinita).

ii. Dará en primer lugar la respuesta V=b, ya que la rama correspondiente a esta respuesta será la primera en ser recorrida. Luego, con punto y coma, se recorrerá la segunda rama y, al entrar en el subárbol infinito, siempre se tomará primero la rama que falla y en segundo

lugar la rama infinita, o sea que no se vuelven a dar las respuestas V=a como en los casos anteriores. Al entrar en una rama infinita que no da respuestas, el intérprete en algún momento dará un error por falta de memoria.

d) Indique qué ramas del árbol de la parte a) son podadas por los cuts de las siguientes variantes del conjunto de cláusulas de **p**. Diga qué respuestas daría en cada caso un intérprete con las características de la parte b).

i.
```
Cp1:  p(X,X) :- !.
Cp2:  p(X,Y) :- q(X,Y), p(Y,X).
Cp3:  p(X,Y) :- r(X,Y).
```

ii.
```
Cp1:  p(X,X).
Cp2:  p(X,Y) :- q(X,Y), !, p(Y,X).
Cp3:  p(X,Y) :- r(X,Y).
```

iii.
```
Cp1:  p(X,X).
Cp2:  p(X,Y) :- q(X,Y), p(Y,X), !.
Cp3:  p(X,Y) :- r(X,Y).
```

Solución:

i. El cut corta las ramas 1 y 2, por lo que la única respuesta será V=a (una sola vez).
ii. El cut corta las ramas 3 y 2, por lo que la única respuesta será V=a (una sola vez).
iii. El cut corta las ramas 2, 4 y 5, por lo que se obtendrá la respuesta V=a correspondiente a la primera rama de más a la izquierda y otra respuesta V=a correspondiente a la rama exitosa que hay por debajo de la rama 3.

Ejercicio 4 [15 puntos]

Implemente en Prolog el metaintérprete **resNoInstanciado(+G)** que funciona de la siguiente manera:

Si durante la resolución se encuentra una invocación al predicado **is/2**, se debe verificar que el lado derecho de la invocación esté completamente instanciado. Si no lo está, en vez de finalizar la ejecución con un error, se debe imprimir un mensaje de advertencia "El lado derecho no está instanciado" y fallar, permitiendo que Prolog continúe con el backtracking. Asuma que el único predicado built-in que puede utilizar **G** es el predicado **is/2**.

Solución:

```
resNoInstanciado(true):-!.
resNoInstanciado((A,B)):-!,
resNoInstanciado(A),
resNoInstanciado(B).
resNoInstanciado(is(X,Y)):-
    !,
verificar_instanciacion(Y),
is(X,Y).
resNoInstanciado(C):-
clause(C,Body),
resNoInstanciado(Body).

verificar_instanciacion(Arg):-
ground(Arg),!.
verificar_instanciacion(_):-
writeln('El lado derecho no está instanciado'),
fail.
```