Imperial College London – Department of Computing

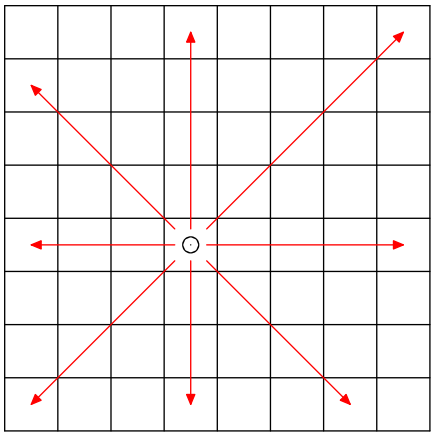
276: Introduction to Prolog
Exercise 5: N Queens

(Based on an Exercise and Solution by Robert Craven)

Solutions, with commentary, will be provided in a separate set of notes.

November 2013

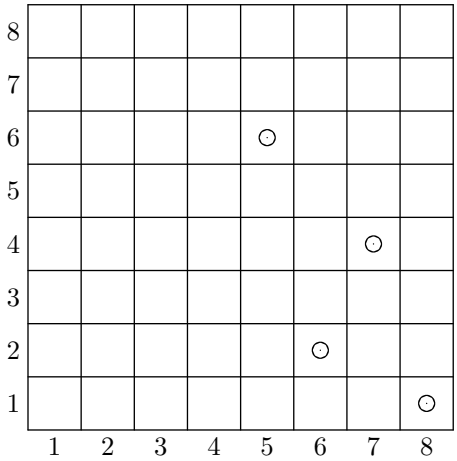
A chessboard is an 8×8 grid; the chess piece known as a 'queen' can attack along any horizontal, vertical and diagonal, as shown below:



The problem: place 8 queens on a chessboard, in such a way that no queen is attacking any other queen. Suppose that a queen in the m^{th} column and the n^{th} row is represented by the `Prolog` term `q(m,n)` (like Cartesian co-ordinates); a configuration of the board is to be represented by a `Prolog` list. For example, the list

```
[q(5,6),q(6,2),q(7,4),q(8,1)]
```

would represent the partially-completed non-solution:



Notice that if all solutions are forced to have the 'canonical' form

```
[q(1,Y1),q(2,Y2),q(3,Y3),q(4,Y4),q(5,Y5),q(6,Y6),q(7,Y7),q(8,Y8)]
```

then we do *not* need to check that there are no vertical attacks. (Why?)

1. Write a program `no_attack/2`, which succeeds on a call

```
?- no_attack(PartialSoln, q(M,N))
```

when a queen can be placed in the m^{th} column and n^{th} row without falling under attack from any of the queens in `PartialSoln`. (`PartialSoln` will be ground.) For example (with reference to the grid above):

```
?- no_attack([q(7,4),q(8,1)], q(6,1)).
no
?- no_attack([q(7,4),q(8,1)], q(6,2)).
yes
```

You can assume that solutions are in the canonical form described above, so that checking for vertical attacks is unnecessary. You should assume that none of the queens in `PartialSoln` attack each other.

2. Using your answer from (**1**), write a program `queens8/1`, which takes as argument a list representing an empty grid, and returns a solution to the 8 Queens problem. You should have a `Prolog` fact `template/1` in the program, to give the 'canonical' form of the solution, so that

```
?- template(Sol), queens8(Sol).
```

will give solutions, thus:

```
template([q(1,_),q(2,_),q(3,_),q(4,_),q(5,_),q(6,_),q(7,_),q(8,_)]).
```

3. Construct a one-line query which gives you the number of solutions to the 8 Queens problem.

4. Write a program `print_board/1` which will output a picture of a *completed* solution, given the canonical list representation of the solution as input. Something like the following should result:

```
_ _ _ _ _ _ _ Q
_ Q _ _ _ _ _ _
_ _ _ Q _ _ _ _
Q _ _ _ _ _ _ _
_ _ _ _ _ _ Q _
_ _ _ _ Q _ _ _
_ _ Q _ _ _ _ _
_ _ _ _ _ Q _ _
```

Prolog I/O primitives are *not* an examinable part of the course. The model solution will show examples of typical bits of procedural Prolog code.

5. There is a redundancy in the data structures we are using to represent our chessboards. As the m^{th} element of our list represents the m^{th} column, we could alter our terms so that, for example, instead of

```
[q(1,3),q(2,8),q(3,6),q(4,2),q(5,1),q(6,4),q(7,7),q(8,5)]
```

we have

```
[3,8,6,2,1,4,7,5]
```

This gives a much more compact representation. In a new file, rewrite your program to use this new representation. (New file so we can use the same predicates without name clashes.)

Every solution will thus be a *permutation* of

```
[1,2,3,4,5,6,7,8]
```

6. The 8 Queens problem can be generalized to the 'n Queens' problem, where instead of an 8×8 chessboard, there is an $n \times n$ grid and n queens. Modify your answer for (**5**) to solve the n Queens problem.