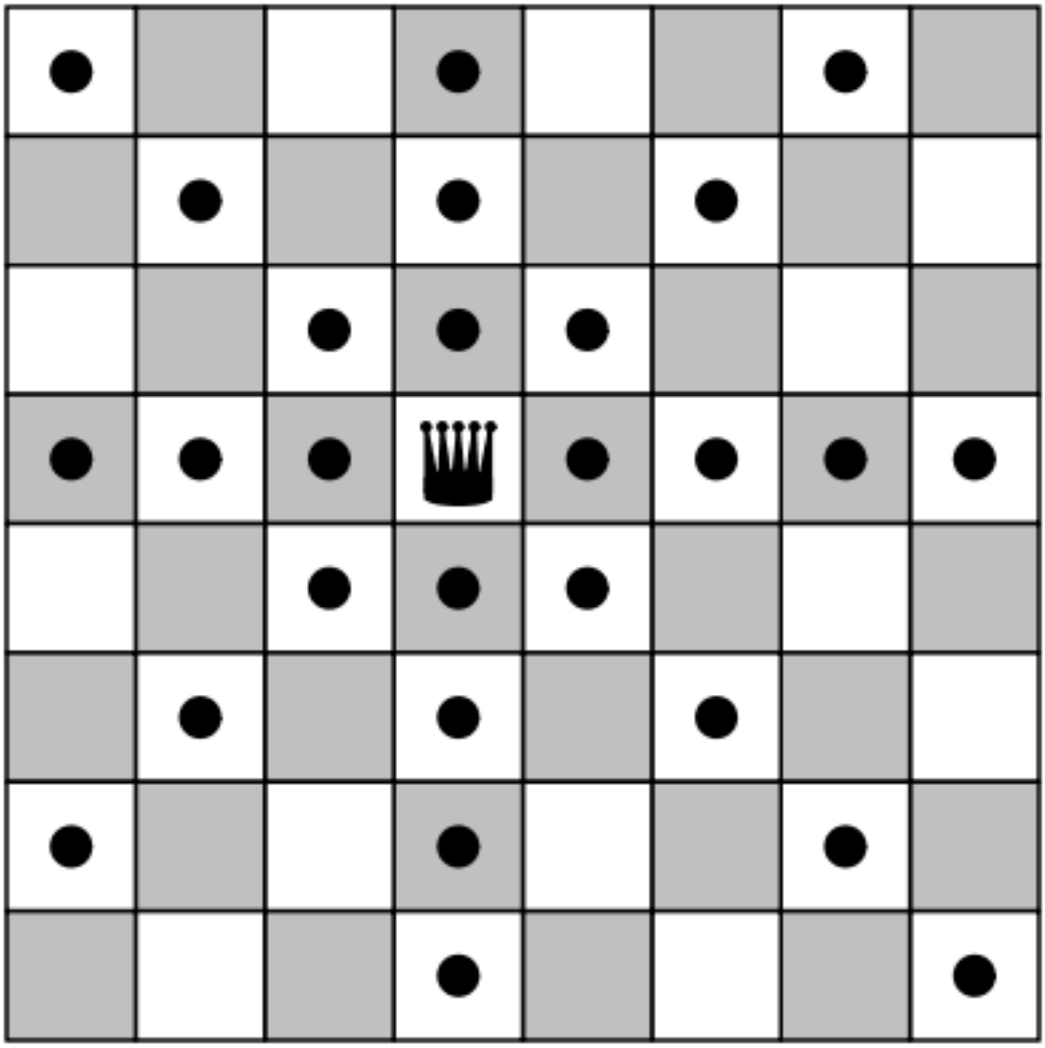
The *n*-Queens Problem



- The *n*-Queens problem is to place *n* queens on an *n* by *n* chessboard such that no two queens are mutually attacking (are in the same row, column, or diagonal). The above image shows the squares attacked by a queen on an 8 by 8 chessboard.
- The problem is solvable for all *n* larger than 3.
- We can use Maple's built-in efficient SAT solver to quickly solve this problem. A SAT solver takes as input a formula in Boolean logic and returns an assignment to the variables which makes the formula true (if one exists). See the Satisfy command for more information.

Setting up the problem

- We'll use the Boolean variables $Q[x, y]$ for x and y between 1 and n to denote if there is a queen on the square at (x, y).
- We'll also use a few simple functions to compute the set of variables which are attacked by a queen at (x, y).
- To define $AttackedVars(x, y)$ we'll use the following four functions:
  * *VarsInCol*: Returns the set of variables in the same column as the square at (x, y).
  * *VarsInRow*: Returns the set of variables in the same row as the square at (x, y).
  * *VarsInDiag*: Returns the set of variables in the same diagonal as the square at (x, y).
  * *VarsInAntiDiag*: Returns the set of variables in the same anti-diagonal as the square at (x, y).

▶ **Function definitions**

Generating the constraints

- For a given n we need to generate the constraints for the n by n chessboard.
- There are two types of constraints:
  * Postive constraints (saying there are at least n queens on the board)
  * Negative constraints (saying each queen doesn't attack any other queen)

Positive constraints

- Each row must contain a queen since there are n rows and only one queen can go in each row.
- These clauses have the form $Q[1, j] \vee Q[2, j] \vee \cdots \vee Q[n, j]$ for j from 1 to n.
- Similarly, each column must contain a queen.

```
positive_constraints := proc(n)
     local x, constraints;
     constraints := Array(1..2*n);
     for x from 1 to n do
          constraints[2*x-1] := &or(seq(Q[i, x], i=1..n));
          constraints[2*x] := &or(seq(Q[x, j], j=1..n))
     end do:
     return &and(entries(constraints, nolist));
end proc:
```

Negative constraints

- If a square contains a queen then no other square in that row, column, or diagonal contains a queen.
- These clauses have the form $Q[x, y] \Rightarrow \neg A$ where A is in $AttackedVars(x, y)$.
- Equivalently, they have the form $Q[x, y] \Rightarrow \neg \left(A_1 \vee \cdots \vee A_k\right)$ where $AttackedVars(x, y) = \left\{A_1, ..., A_k\right\}$.

```
negative_constraints := proc(n)
     local x, y, constraints;
     constraints := Array(1..n, 1..n);
     for x from 1 to n do
          for y from 1 to n do
               constraints[x, y] := &implies(Q[x, y], &not(&or
(AttackedVars(x, y)[])));
          end do;
     end do;
     return &and(entries(constraints, nolist));
end proc:
```

Finding a solution

- We use the Satisfy command from the Logic package which finds a satisfying assignment of a logical formula if one exists.
- We use the Usage command from the CodeTools package to mesaure how quickly the solution is found.

```
n := 8:

with(Logic):
all_constraints := &and(positive_constraints(n),
negative_constraints(n)):
satisfying_assignment := CodeTools:-Usage(Satisfy(
all_constraints)):
```

Visualization of the solution

- The following functions use commands from the plottools package to draw a graphical representation of a chessboard and a chess queen on square (x, y).

```
with(plottools):

drawQueen := proc(x, y)
     return ellipse([x+.5, y+.35], .24, 0.05, filled = true),
     rectangle([x+.25, y+.35], [x+.75, y+.2]),
     ellipse([x+.5, y+.2], .25, 0.05, filled = true),
     disk([x+.5, y+.75], 0.03),
     disk([x+.625, y+.75], 0.03),
     disk([x+.375, y+.75], 0.03),
     disk([x+.25, y+.75], 0.03),
     disk([x+.75, y+.75], 0.03)
```

- The following code draws the chessboard and the queens on the board in the locations that correspond to the squares which are assigned to true in the satisfying solution which was found.

```
queens := Array(1..n):
i := 1:
for eq in satisfying_assignment do
     if rhs(eq) then
          queens[i] := drawQueen(op(lhs(eq)));
          i := i + 1;
     end if;
end do:

plots:-display(entries(queens, nolist), drawChessboard(),
scaling=constrained, axes=none);
```