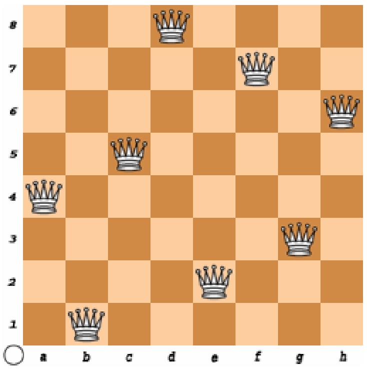

Ⓒ https://commons.wikimedia.org/wiki/File:Eight_Queens12_positions.gif

Fig. 6.1 A Solution of the 8 Queens Puzzle

6.1 Chess Puzzles

We begin with two puzzles that are laid out on a chessboard with $N \times N$ squares; typically we have $N = 8$, but any other size is also possible.

N Queens

The problem posed by the "N queens puzzle" is to place N queens on the board such that no two queens can beat each other (recall that the chess piece "queen" may beat any other piece in the same row, same column, same main diagonal, or same antidiagonal). Figure 6.1 depicts a solution to the 8 queens problem, but there are many other ones.

To represent such a solution, it may seem at first hand necessary to use N pairs of numbers, where each pair denotes the position of a queen by a row index and a column index. However, since each queen must occupy a separate row and we have as many queens as rows, it actually suffices to represent a solution by an array q of N numbers, where $q[i]$ is the column of the queen placed in row i. Thus the solution displayed in Figure 6.1 is represented as $q = [1,4,6,0,2,7,5,3]$ (assuming that we use the numbers $0,\ldots,7$ to denote the column indices $a,\ldots,h$ as well as the row indices $1,\ldots,8$ displayed in the figure). This idea gives rise to the following declarations:

```
val N:ℕ;
axiom notNull ⇔ N > 0;
type Num = ℕ[N];
type Queens = Array[N,Num];
```

We use here the type *Num* to denote the domain of numbers $0, \ldots, N$; while the value N itself is actually not a valid row/column index, it will become handy later. A value of type *Queens* represents a solution of the N queens puzzle as an an array of length N that holds column indices.

The core idea is to represent the puzzle as a system whose state is a pair $\langle q, n\rangle$ of a partial solution where the first n queens have already been placed in array q. A move of the system consists of finding a position p for the next queen such it does not beat any of the queens that have already been placed, updating $q[n]$ by p, and incrementing n by 1. This gives rise to the following definition of a nondeterministic system:

```
shared system Queens
{
  var q:Queens = Array[N,Num](N);
  var n:Num = 0;
  action place(p:Num) with admissible(q,n,p);
  {
    q[n] := p;
    n := n+1;
  }
}
```

The state of this system *Queens* is represented by the declarations of the two variables q and n. The unique initial state is represented by the initialization values of these variables (where q holds at all positions the value N that may be interpreted as "no index yet"). The possible moves of the system are represented by the action *place* that selects an arbitrary position p that satisfies the atomic formula *admissible*(q, n, p) and then updates the state as was explained above.

The core of this definition is the predicate *admissible* defined as follows:

```
pred admissible(q:Queens,n:Num,p:Num) ⇔
  n < N ∧ p < N ∧
  ∀i:Num with i < n.
    q[i] ≠ p ∧
    q[i]-i ≠ p-n ∧
    q[i]+i ≠ p+n;
```

This predicate states that, given a partial placement q of n queens, the choice of column p for the next queen is admissible if there is still a queen to be placed ($n < N$) and p is a valid column index ($p < N$) such that no queen placed previously in row i and column $q[i]$ is in the same column ($q[i] \neq p$), the same main diagonal ($q[i] - i \neq p - n$), or the same antidiagonal $q[i] + i \neq p + n$).

If we now set in the RISCAL GUI the value $N = 8$ and execute the operation *Queens*, the following output is shown:

```
Executing system Queens.
2057 system states found with search depth 9.
Execution completed (147 ms).
```

This output demonstrates that the repeated execution of all possible admissible choices for q lead to 2057 system states where the generated state sequences describing the iterative choices of positions for queens had at most length 9 (including the initial state); this implicitly shows that at least once all 8 queens could be successfully placed. But how do we actually get access to some (or, if desired, all) computed solutions?

The easiest way is to annotate the system with an invariant. Similar to the previously introduced concept of a loop invariant (which must hold before the the loop is executed and after every successive iteration of the loop), such a *system invariant* must hold in the initial state of the system and be preserved by the execution of every action. If the system invariant is violated, RISCAL aborts the execution of the system with an error message that demonstrates the sequence of states leading to this error together with the corresponding actions that have been performed.

Thus if we annotate above system with the invariant

```
invariant n < N;
```

we get the following error:

```
Executing system Queens.
ERROR in execution of system Queens: evaluation of
  invariant n < N;
at line 30 in file chess.txt:
  invariant is violated
The system run leading to this error:
  0:[q:[8,8,8,8,8,8,8,8],n:0]->place(0)->
  1:[q:[0,8,8,8,8,8,8,8],n:1]->place(4)->
  2:[q:[0,4,8,8,8,8,8,8],n:2]->place(7)->
  3:[q:[0,4,7,8,8,8,8,8],n:3]->place(5)->
  4:[q:[0,4,7,5,8,8,8,8],n:4]->place(2)->
  5:[q:[0,4,7,5,2,8,8,8],n:5]->place(6)->
  6:[q:[0,4,7,5,2,6,8,8],n:6]->place(1)->
  7:[q:[0,4,7,5,2,6,1,8],n:7]->place(3)->
  8:[q:[0,4,7,5,2,6,1,3],n:8]
ERROR encountered in execution.
```

This error describes the system execution that, starting with the initial state $q = [8,8,8,8,8,8,8,8]$ and $n = 0$ (describing the situation where no queen has been placed yet), is derived by iteratively placing 8 queens at positions 0, 4, 7, 5, 2, 6, 1, 3 leading to the final system state $q = [0,4,7,5,2,6,1,3]$ and $n = 8$ which denotes a solution to the puzzle; since this state violates the invariant, the execution of the system is aborted, immediately after the first solution has been found.

However, it is also possible to compute all solutions to the problem by annotating the system with the following invariant:

```
invariant n = N ⇒ print q in ⊤;
```

The evaluation of the formula `print` q `in` $\top$ yields the same truth value as the formula $\top$ (i.e., “true”) but additionally prints the value of q as a side effect. Thus this invariant does in no case abort the execution of the system such that the exectuion elaborates all legal placements of queens and prints those that represent complete solutions to the puzzle. Indeed with this annotation, the system execution yields the following output:

```
Executing system Queens.
[0,4,7,5,2,6,1,3]
[0,5,7,2,6,3,1,4]
[0,6,3,5,7,1,4,2]
...
[7,2,0,5,1,4,6,3]
[7,3,0,2,5,1,6,4]
2057 system states found with search depth 9.
Execution completed (410 ms).
```

Indeed by using an invariant

```
invariant n = N ⇒ print q in printall;
```

with a global Boolean constant *printall*, we can configure the system to have all solutons or just one printed. With the declaration

```
val printall = ⊥;
```

we get the output

```
Executing system Queens.
[0,4,7,5,2,6,1,3]
ERROR in execution of system Queens: evaluation of
  invariant (n = N) ⇒ (print q in printall);
  ...
```

with the same system trace as shown above. To get the system traces of all solutions, we may use the following invariant:

```
invariant n = N ⇒ printtrace in printall;
```

The evaluation of the formula `printtrace in` $\top$ yields the truth value of *printall* but additionally prints as a side effect the trace leading to the current state. So for $printall = \top$, the execution of *System* shows the following output:

```
Executing system Queens.
  0:[q:[8,8,8,8,8,8,8,8],n:0]->place(0)->
  ...
  8:[q:[0,4,7,5,2,6,1,3],n:8]
  ...
  0:[q:[8,8,8,8,8,8,8,8],n:0]->place(7)->
  ...
  8:[q:[7,3,0,2,5,1,6,4],n:8]
2057 system states found with search depth 9.
Execution completed (566 ms).
```

Thus, by using invariants with `print` and/or `printtrace` annotations, states respectively system execution of interests can be displayed.

Above system definition initializes states by explicit values and updates them by imperative commands. In a pure "mathematical" formulation, a nondeterministic system with a state space S is described by its initial state set $I \subseteq S$ and its transition relation $R \subseteq S \times S$; these can be conveniently defined by logical formulas $I(s)$ and $R(s, s')$. An execution of such a system is a sequence $\langle s_0, s_1, \ldots \rangle$ of states with $I(s_0)$. If the sequence is infinite, then $R(s_i, s_{i+1})$ for every $i \in \mathbb{N}$. If the sequence is finite with final state s_n, then $R(s_i, s_{i+1})$ for every $i < n$ and there is no state s' with $R(s_n, s')$.

Actually, also RISCAL allows such a "mathematical" formulation based on logical formulas: