ISP: Configuration project

Authors: Jan Vium Enghoff and Søren Harrison

Introduction

The intent for this project was to build an interactive configurator for specifying a valid solution to the n-queen problem. At no point should it be possible for the user to end up with an invalid queen configuration. This meant disabling the cells, which did not have a valid solution with a queen placed there. The configuration should work on any size chessboard, but for simplicity, we will only look at the standard 8x8 board in this report.

Implementation

The implementation used the BDD framework javabdd[1] to generate a decision tree that could be queried to get the valid configurations for the chessboard.

BDD structure

The binary decision diagram consisted of 64 variables, one for each cell (8x8). The value of the variable indicates if a queen is placed in the cell. True means there is a queen, while false means there is not. The board gets restricted as the user places the queen on the board. This is done by continually assigning the variables to true for cells as the user goes through a configuration problem and the logic engine updates the surrounding cells to false should they break any of the n-queen rules. These rules are assigned before the beginning of every configuration to get the initial BDD. This graph will be vast early on, but as the configuration, progresses will quickly be reduced due to the unknown variables being replaced by constants.

The rules we applied are the following:

1. Horizontal rule
2. Vertical rule
3. Diagonally descending rule
4. Diagonally ascending rule

These can summed up, as no queen must be immediate threat to another queen on the board. From these other restrictions can be applied as we will see in the conclusion. Rules are implemented as logical conjunctions of each other to create a group of binary propositions that must hold for there to exist a solution. These conjunction groups consists of smaller conjunctions, which describes either a row, column or diagonal, depending on the rule. These smaller conjunctions consists of disjunctions, which describes the rules a single cell must obey to. Formally this can be expressed as:

$$R_c = C_c \wedge (-C_1 \wedge -C_2 \wedge \ldots \wedge -C_n)$$
$$R_{rcd} = R_{c_1} \vee R_{c_2} \vee \ldots \vee R_{c_n}$$
$$R_d = R_{rcd_1} \wedge R_{rcd_2} \wedge \ldots \wedge R_{rcd_n}$$
$$R_{nqueen} = R_{d_1} \wedge R_{d_2} \wedge \ldots \wedge R_{d_n}$$

[1] http://javabdd.sourceforge.net/

Where $C_c$is the cell variable, R_c is the rules of the cell, R_{rcd} is the rule of the row/column/diagonal, R_d is the rules of a direction and finally R_{nqueen} is the rules of the n-queen configuration. A snapshot of the rules for vertical direction are included on the figure below. We have left most of the variables for clarity.

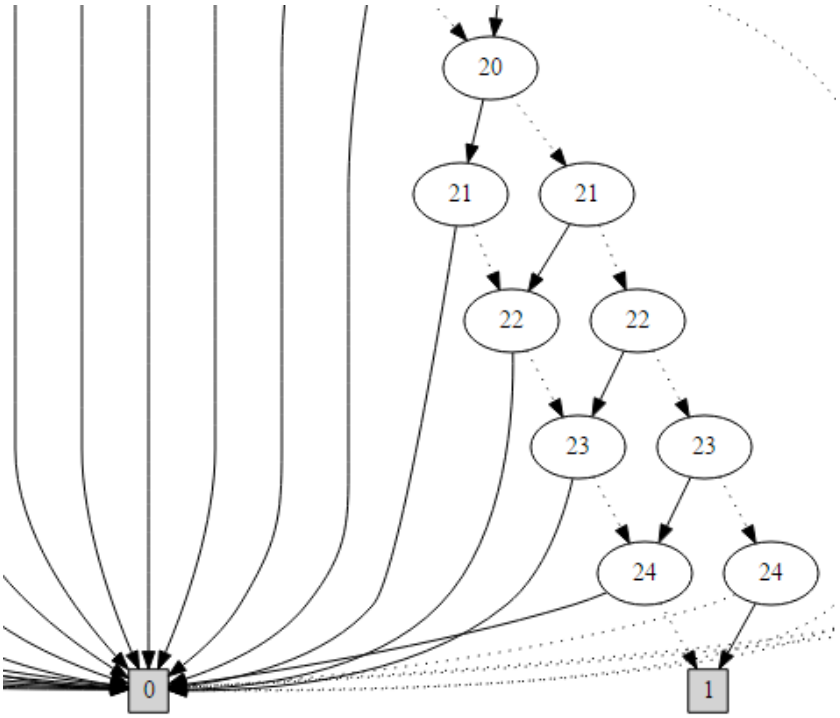

Figure 1: A snapshot of the initial BDD for the vertical rules on a 5x5 board.

To map from a cell in the matrix to a single variable index in the BDD we use the formula:

$$C_{xy} = x * x_{max} + y$$

Completing the configuration

To compare the BDD to the actual board we go through every cell coordinate combination and check if there exists a solution in the AllSat, which has assigned the current cell to true. If this is the case we leave the cell as a valid option, otherwise mark it as invalid. In the case that the SatCount is equal to one, that is there only exists a single valid assignment for the remaining possible cells, we auto complete the rest of the board assignments.

Conclusion

For this solution, we used AllSat to identify all the possible solution and then iterate through them for each cell to see if any of them contained a queen. Another possible solution is to use to try to assign the variable in the BDD for a cell and then check if AnySat is true.

One thing we have not touched on in this report is the ordering of the variables. We picked a straightforward approach, which was to match the cell variable using the formula presented in the implementation section. This will yield a small tree when generating the vertical rules, but it will obviously not hold for every rule.

As seen on the figure below will the n-queen rules we introduced bring with them additional restrictions. These can be hard for a human to identify, as these are not trivial and therefore increases the quality of the configurator even further.

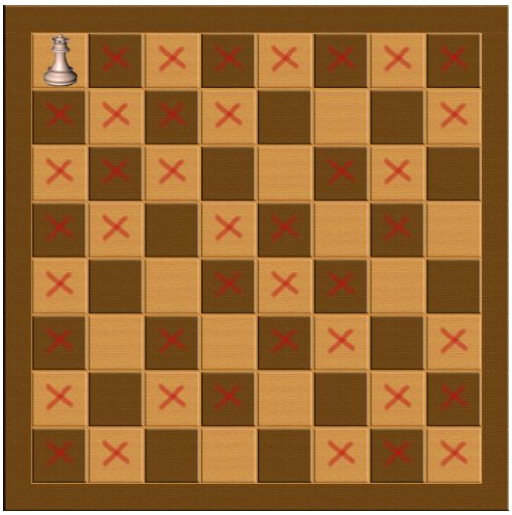

Figure 2: Valid board configurations after assigning the first queen in the top left corner on an 8x8 board.