Applied Logic Weekly Assignment Report

Bohdan Tymofieienko B.S.
b.tymofieienko@student.fontys.nl
4132645

Mikolaj Hilgert M.H.
m.hilgert@student.fontys.nl
4158385

December 13, 2021

Contents

1 Week ii. 8 Queen puzzle Medicine problem.

1.1 8 Queen puzzle

As the 8 queen problem requires for 8 queens to be placed on a 8x8 chessboard (56 possible squares), we must define the constraints of the game to *Z3*. A queen in Chess is the most powerful piece. The queen can move freely up and down on any column (referred to as a **File** in Chess) and left and right on any row (referred to as a **Rank** in Chess). The queen can also move freely in a diagonal.

Based on the Sudoku example, our approach to this problem, was to first construct the chessboard and then we may begin adding the queens.

```
(declare-fun B (Int Int) Bool)
```

Knowing that the Queen may move as a Rook (and Bishop), we instantly can make the observation that we may have at most, one queen per file and rank. As then the queens will not be in one an-others line of attack. Which as the problems name implies, means a placement of **at-most** 8 queens in an 8x8 board.

```
(define-fun MinOneQueenPerFile ((file Int)) Bool
    (or (B file 1) (B file 2) (B file 3) (B file 4) (B file 5) (B file 6)
    (B file 7) (B file 8)))
(define-fun MinOneQueenPerRank ((rank Int)) Bool
    (or (B 1 rank) (B 2 rank) (B 3 rank) (B 4 rank) (B 5 rank) (B 6 rank)
    (B 7 rank) (B 8 rank)))
```

Using the aforementioned functions that use the or operator to say that just one may be true (meaning only one queen in the file/rank). We may create another two functions that combine this and set the constraints for **all** ranks and files.

```
(define-fun MinOneQueenPerAllFiles () Bool
    (and
        (MinOneQueenPerFile 1)
        (MinOneQueenPerFile 2)
        ...
        (MinOneQueenPerFile 8)))

(define-fun MinOneQueenPerAllRanks () Bool
    (and
        (MinOneQueenPerRank 1)
        (MinOneQueenPerRank 2)
        ...
        (MinOneQueenPerRank 8)))
```

Next, it is important to define to *Z3* some additional rules of Chess, namely that you may only place one piece on a square. We do this by saying that if one Queen is placed on a square, that implies that all other squares in the corresponding file and rank do **not** have another Queen on it. This fact is defined for all files and ranks in the board.
Let C represent a cell, then,

$$C_1 \implies (\neg C_2) \wedge (\neg C_3) \wedge (\neg C_4) \wedge (\neg C_5) \wedge (\neg C_6) \wedge (\neg C_7) \wedge (\neg C_8)$$
$$\ldots$$
$$C_8 \implies (\neg C_1) \wedge (\neg C_2) \wedge (\neg C_3) \wedge (\neg C_4) \wedge (\neg C_5) \wedge (\neg C_6) \wedge (\neg C_7)$$

Applying De Morgan's law

$$C_1 \implies \neg(C_2 \vee C_3 \vee C_4 \vee C_5 \vee C_6 \vee C_7 \vee C_8)$$

In $Z3$ syntax,

```
(define-fun UniquePerFile ((file Int)) Bool
    (and
    (=> (B file 1) (not (or (B file 2) (B file 3) (B file 4) (B file 5)
    (B file 6) (B file 7) (B file 8))))
    (=> (B file 2) (not (or (B file 1) (B file 3) (B file 4) (B file 5)
    (B file 6) (B file 7) (B file 8))))
    ...
    (=> (B file 8) (not (or (B file 1) (B file 2) (B file 3) (B file 4)
    (B file 5) (B file 6) (B file 7))))))

(define-fun UniquePerRank ((rank Int)) Bool
    (and
    (=> (B 1 rank) (not (or (B 2 rank) (B 3 rank) (B 4 rank) (B 5 rank)
```

```
    (B 6 rank) (B 7 rank) (B 8 rank))))
    (=> (B 2 rank) (not (or (B 1 rank) (B 3 rank) (B 4 rank) (B 5 rank)
    (B 6 rank) (B 7 rank) (B 8 rank))))
    ...
    (=> (B 8 rank) (not (or (B 1 rank) (B 2 rank) (B 3 rank) (B 4 rank)
    (B 5 rank) (B 6 rank) (B 7 rank))))))

(define-fun UniquePerAllFiles () Bool
    (and
        (UniquePerFile 1)
        (UniquePerFile 2)
        ...
        (UniquePerFile 8)))

(define-fun UniquePerAllRanks () Bool
    (and
        (UniquePerRank 1)
        (UniquePerRank 2)
        ...
        (UniquePerRank 8)))
```

Next, came the most laborious part of the problem. As the Queen not only moves as a Rook, but also may move in diagonals as a Bishop. This means that we must define that a Queen cannot be placed in the direct line of attack of another Queen.

This was done using some of the previous logic. The difference being that there are more diagonals on a chessboard, which also are of **non** equal lengths. The largest spanning over 8 squares, and the smallest just two. For the sake of clarity we decided to create one function that has 8 parameters, so that we may reuse it for all of the diagonals.

```
(define-fun UniquePer8Diagonal((val1 Int) (val2 Int) (val3 Int) (val4 Int)
(val5 Int) (val6 Int) (val7 Int) (val8 Int)) Bool

    (and
        (=> (B 1 val1 ) (not (or (B 2 val2 ) (B 3 val3 ) (B 4 val4 )
        (B 5 val5 ) (B 6 val6 ) (B 7 val7 ) (B 8 val8 ))))

        (=> (B 2 val2 ) (not (or (B 1 val1 ) (B 3 val3 ) (B 4 val4 )
        (B 5 val5 ) (B 6 val6 ) (B 7 val7 ) (B 8 val8 ))))

        (=> (B 3 val3 ) (not (or (B 2 val2 ) (B 1 val1 ) (B 4 val4 )
        (B 5 val5 ) (B 6 val6 ) (B 7 val7 ) (B 8 val8 ))))

        (=> (B 4 val4 ) (not (or (B 2 val2 ) (B 3 val3 ) (B 1 val1 )
        (B 5 val5 ) (B 6 val6 ) (B 7 val7 ) (B 8 val8 ))))

        ...

        (=> (B 8 val8 ) (not (or (B 2 val2 ) (B 3 val3 ) (B 4 val4 )
        (B 5 val5 ) (B 6 val6 ) (B 7 val7 ) (B 1 val1 ))))))
```

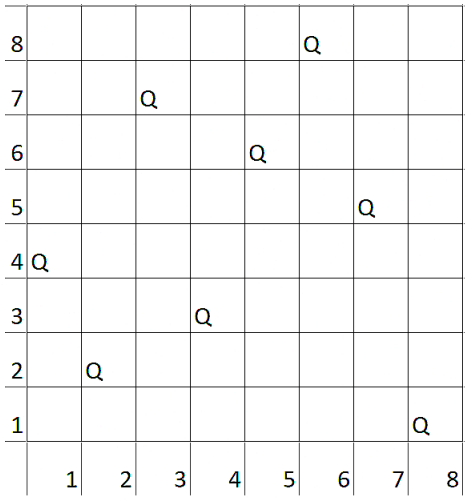
With this we could now construct all of the possible diagonals that a Queen may attack. We split the diagonals into Main diagonals, Upper-left diagonals, Bottom-right diagonals, Upper-right reverse diagonals and finally Upper-right reverse diagonals.

```
(define-fun UniquePerAllMainDiagonal() Bool
    (and
        ;Main diagonals
        (UniquePer8Diagonal 1 2 3 4 5 6 7 8)
        (UniquePer8Diagonal 8 7 6 5 4 3 2 1)

        ;Upper-left diagonal /
        (UniquePerMainDiagonal 2 3 4 5 6 7 8 0)
        (UniquePerMainDiagonal 3 4 5 6 7 8 0 0)
        (UniquePerMainDiagonal 4 5 6 7 8 0 0 0)
        (UniquePerMainDiagonal 5 6 7 8 0 0 0 0)
        (UniquePerMainDiagonal 6 7 8 0 0 0 0 0)
        (UniquePerMainDiagonal 7 8 0 0 0 0 0 0)

        ;Bottom-right diagonal /
        (UniquePerMainDiagonal 0 1 2 3 4 5 6 7)
        (UniquePerMainDiagonal 0 0 1 2 3 4 5 6)
        (UniquePerMainDiagonal 0 0 0 1 2 3 4 5)

        ect...
```

After giving all these conditions, we must assert them together. To get the output.

```
(assert (and MinOneQueenPerAllFiles MinOneQueenPerAllRanks UniquePerAllFiles
UniquePerAllRanks UniquePerAllMainDiagonal))

(check-sat)
(get-model)
```

The output from the solver gives us a valid arrangement of Queens on a board that adheres to all the puzzles requirements. We also tried setting constraints to certain squares to test if the solution worked properly and we got valid outputs regardless.



1.2 Medicine testing

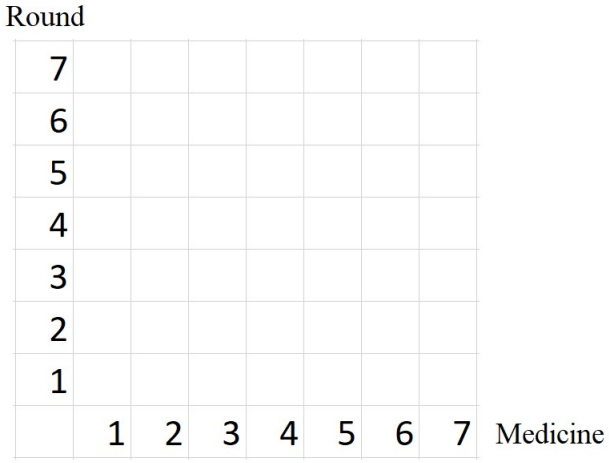
The main goal in this problem is to define if a test with a certain rules might be conducted. In particular:

1. **Every medicine is tested in exactly three rounds.**
2. **Every test round tests exactly three different medicines.**
3. **No pair of medicine is tested more than once in the same test round.**

Third constraint essentially means that if one pair of medicines were tested together, it cannot be tested together in any of the following rounds. This problem can be solved with $Z3$ if we formulate next statements:

Let T be a 2-dimensional array of boolean values where rows are rounds and columns are medicines. True is equivalent to *"tested in this round"* and **False** would be equivalent to the opposite. Then,

1. There are exactly three true value per each row. (Three different medicines per round.)
2. There are exactly three true value per each column. (Three different rounds per medicine.)
3. There exist no two rows where number of intersecting (elements > 1. No pair of medicines.)



We will express these statements in form of Z3 syntax.

1. In order to assure that there are exactly three True values per row (Round), we will count the amount and compare to 3. We declare a function that does so for one row and reuse it in later stage.

```
(define-fun ExactlyThreePerRound((Round Int)) Bool
    (and (= (N_rows Round) (+
            (ite (T Round 1) 1 0)
            (ite (T Round 2) 1 0)
            (ite (T Round 3) 1 0)
            (ite (T Round 4) 1 0)
            (ite (T Round 5) 1 0)
            (ite (T Round 6) 1 0)
            (ite (T Round 7) 1 0)))
    (= (N_rows Round) 3)))
```

Now assure next for all rounds:

```
(define-fun ExactlyThreeForAllRounds() Bool
    (and (ExactlyThreePerRound 1)
         ...
        (ExactlyThreePerRound 7)))
```

2. To assure there are exactly three True values in each column we declare a very similar function but for a column (medicine).

```
(define-fun ExactlyThreePerRound((Round Int)) Bool
    (and (= (N_rows Round) (+
            (ite (T Round 1) 1 0)
             ...
            (ite (T Round 7) 1 0)))
    (= (N_rows Round) 3)))

(define-fun ExactlyThreeForAllMeds() Bool
    (and (ExactlyThreePerMedicine 1)
         ...
        (ExactlyThreePerMedicine 7)))
```

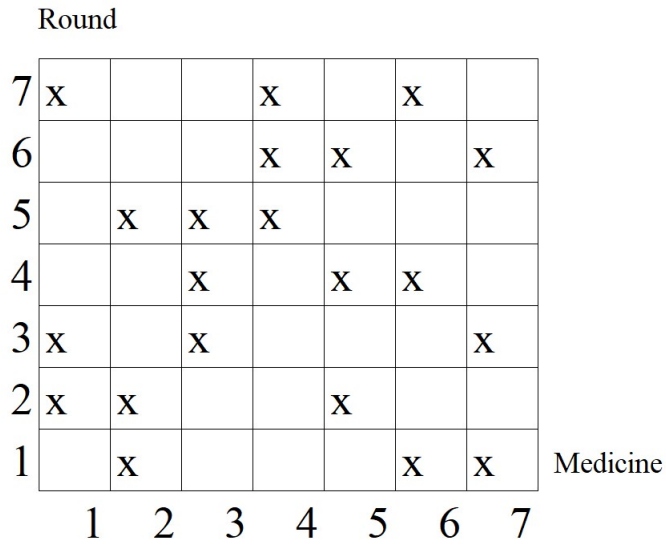
3. For the last constraint the strategy is to set the constraint that if comparing any two row number of coinciding elements is less than 2. We do this check for all the rows (all rounds).

```
(define-fun NoTwoPairsInRound((Round1 Int) (Round2 Int)) Bool
   (and
        (= (N_pairs Round1 Round2) (+
                (ite (and (T Round1 1) (T Round2 1)) 1 0)
                 ...
                (ite (and (T Round1 7) (T Round2 7)) 1 0)
          )
        )
        (< (N_pairs Round1 Round2) 2)
   )
)
```

Finally to make sure that this holds true for all the combination of rows we reuse a fucntion declared before. We add one optimization, we compare two rows only if index of row A is less than index of row B. E.g if we compared row 1 and 2, there is no need to compare row 2 and 1. Thus, first row we compare with 6 other rows, row 2 with 5 other rows etc.

```
(define-fun NoTwoPairsInAllRounds() Bool
    (and
        ;Row 1
        (NoTwoPairsInRound 1 2)
        (NoTwoPairsInRound 1 3)
        (NoTwoPairsInRound 1 4)
        (NoTwoPairsInRound 1 5)
        (NoTwoPairsInRound 1 6)
        (NoTwoPairsInRound 1 7)
        ;Row 2
        (NoTwoPairsInRound 2 3)
        (NoTwoPairsInRound 2 4)
        (NoTwoPairsInRound 2 5)
        (NoTwoPairsInRound 2 6)
        (NoTwoPairsInRound 2 7)
        ;Row 3
        (NoTwoPairsInRound 3 4)
        (NoTwoPairsInRound 3 5)
        (NoTwoPairsInRound 3 6)
        (NoTwoPairsInRound 3 7)
        ;Row 4
        (NoTwoPairsInRound 4 5)
        (NoTwoPairsInRound 4 6)
        (NoTwoPairsInRound 4 7)
        ;Row 5
        (NoTwoPairsInRound 5 6)
        (NoTwoPairsInRound 5 7)
        ;Row 6
        (NoTwoPairsInRound 6 7)))
```

To get a result we retrieve a model. Since this set of constraints is possible to satisfy, we are getting of the possible solutions.



Which essentially states that in round 1 medicines $\{2, 6, 7\}$ are tested, in round 2 medicines $\{1, 2, 5\}$ etc.