Project: Satisfiablity test of clauses and its application

Sruthi Mandalapu - R11906160
Gowtham Edumudi - R11912904

1 Problem Statement

Problem: The N-Queens problem in which, the Queens need to placed on an N×N chessboard so that no two queens threaten each other. That is no 2 queens should be in same row, column or diagnol. The goal is to find the solution considering the above constraints. In detailed it can be explained as:

Given: N×N chessboard
(a) There should be no 2 queens existing in the same row
(b) There should be no 2 queens existing in the same column
(c) There should be no 2 queens existing in the same diagnol

To Prove: To find weather there exists a solution for solving n-queens problem or not [i.e, weather it is satisfiable/unsatisfiable]. If satisfiable gives the solution in the form, where should be the Queen positions are to be placed on the chess board.

2 Mechanism to solve N Queens Problem

Objective: The objective is to find a formula whose satisfiable assignment gives us a solution to the n-queens problem.

Base Idea: To introduce a proposition letter (variable) for each position on the 3×3 board. Let P_{ij} denote the propositional letter for position (i, j) of the board. If P_{ij} is true, there is a queen on (i, j); otherwise, there is no queen on (i, j).

1. Applying on rows:
   To express the constraint that there is no two queens on the first row: if any P1i is true, then other P1j's of first row should be false, i.e.,
   $P_{11} \to \neg P_{12}$,
   $P_{11} \to \neg P_{13}$,
   $P_{12} \to \neg P_{11}$,
   $P_{12} \to \neg P_{13}$,
   $P_{13} \to \neg P_{11}$,
   $P_{13} \to \neg P_{12}$
   We get $(\neg P_{11} \vee \neg P_{12}) \wedge (\neg P_{11} \vee \neg P_{13}) \wedge (\neg P_{12} \vee \neg P_{13})$
   This can be expressed as $(\neg P_{ij} \vee \neg P_{ik})$, where i = row number, j = column number, k ∈ (j+1, j+2, ..., n)

2. Applying on columns:
   To express the constraint that there is no two queens on the first columns: if any Pi1 is true, then other Pj1's of first column should be false, i.e.,
   $P_{11} \to \neg P_{21}$,
   $P_{11} \to \neg P_{31}$,
   $P_{21} \to \neg P_{11}$,
   $P_{21} \to \neg P_{31}$,
   $P_{31} \to \neg P_{11}$,
   $P_{31} \to \neg P_{21}$
   We get $(\neg P_{11} \vee \neg P_{21}) \wedge (\neg P_{11} \vee \neg P_{31}) \wedge (\neg P_{21} \vee \neg P_{31})$
   This can be expressed as $(\neg P_{ij} \vee \neg P_{kj})$, where i = row number, j = column number, k ∈ (i+1, i+2, ..., n)

3. Applying on left diagnols:
   To express the constraint that there is no two queens on the left diagnol: if any Pij is true, then other position in that diagnol should be false, i.e.,
   $P_{21} \rightarrow \neg P_{32}$,
   $P_{32} \rightarrow \neg P_{21}$,
   $P_{11} \rightarrow \neg P_{22}$,
   $P_{11} \rightarrow \neg P_{33}$,
   $P_{22} \rightarrow \neg P_{11}$,
   $P_{22} \rightarrow \neg P_{33}$,
   $P_{33} \rightarrow \neg P_{11}$,
   $P_{33} \rightarrow \neg P_{22}$,
   $P_{12} \rightarrow \neg P_{23}$,
   $P_{23} \rightarrow \neg P_{12}$
   We get $(\neg P_{11} \vee \neg P_{22}) \wedge (\neg P_{11} \vee \neg P_{33}) \wedge (\neg P_{22} \vee \neg P_{33}) \wedge (\neg P_{12} \vee \neg P_{23}) \wedge (\neg P_{21} \vee \neg P_{32})$
   This can be expressed as $(\neg P_{ij} \vee \neg P_{kl})$, where i = row number, j = column number, k $\in$ (i+1, i+2, ..., n), l $\in$ (j+1, j+2, ..., n)

4. Applying on right diagnols:
   To express the constraint that there is no two queens on the left diagnol: if any Pij is true, then other position in that diagnol should be false, i.e.,
   $P_{23} \rightarrow \neg P_{32}$,
   $P_{32} \rightarrow \neg P_{23}$,
   $P_{13} \rightarrow \neg P_{22}$,
   $P_{13} \rightarrow \neg P_{31}$,
   $P_{22} \rightarrow \neg P_{13}$,
   $P_{22} \rightarrow \neg P_{31}$,
   $P_{31} \rightarrow \neg P_{13}$,
   $P_{31} \rightarrow \neg P_{22}$,
   $P_{12} \rightarrow \neg P_{21}$,
   $P_{21} \rightarrow \neg P_{12}$
   We get $(\neg P_{12} \vee \neg P_{21}) \wedge (\neg P_{13} \vee \neg P_{22}) \wedge (\neg P_{13} \vee \neg P_{31}) \wedge (\neg P_{22} \vee \neg P_{31}) \wedge (\neg P_{23} \vee \neg P_{32})$
   This can be expressed as $(\neg P_{ij} \vee \neg P_{kl})$, where i = row number, j = column number, k $\in$ (i+1, i+2, ..., n), l $\in$ (j-1, j-2, ..., n)

3 Pseudocode of Implementation

- Pseudocode for generating input to SAT SOLVER:

  1. Initializations:
     n = Size of Chess Board [Chess Board of Dimensions n×n]
     lst = list defined for the positions on chess board [size of array is n×n]

  2. Display position of Queens in chess board:
     ```
     while(i!=(n*n)+1):
         p=[]
         for j in range(n):
             p.append(i)
             print(i,end=" ")
             i+=1
         print(0)
     ```

  3. Verifying the row positions:
     ```
     for i in range(n):
         for j in range(n):
             for k in range(j+1,n):
                 print(-lst[i][j],-lst[i][k],0)
     ```

4. Verifying the column positions:
```
for j in range(n):
    for i in range(n):
        for k in range(i+1,n):
            print(-lst[i][j],-lst[k][j],0)
```

5. Verifying the left diagnols:
```
for i in range(n):
    if(i!=0):
        q=[(0,i),(i,0)]
        for j in q:
            l=j[0]
            r=j[1]
            p=[]
            while(r!=n and l!=n):
                p.append(lst[l][r])
                l+=1
                r+=1
            for it in range(len(p)):
                for it1 in range(it+1,len(p)):
                    print(-p[it],-p[it1],0)
    else:
        l=0
        r=0
        p=[]
        while(r!=n and l!=n):
            p.append(lst[l][r])
            l+=1
            r+=1
        for it in range(len(p)):
            for it1 in range(it+1,len(p)):
                print(-p[it],-p[it1],0)
```

6. Verifying the right diagnols:
```
for i in range(1,n):
    q=[(0,i),(i,n-1)]
    for j in q:
        l=j[0]
        r=j[1]
        p=[]
        while(r!=-1 and l!=n):
            p.append(lst[l][r])
            l+=1
            r-=1
        for it in range(len(p)):
            for it1 in range(it+1,len(p)):
                print(-p[it],-p[it1],0)
```

- DIMACS File providing to SAT SOLVER:
  The below file is provided to the sat solver. This produces weather the set of clauses is satisfiable or not. The below given example, is used for solving 2-queens problem which produces unsatisfiability, that means there exists no solution for solving 2 queens.
  p cnf 2 8
  1 2 0
  3 4 0
  -1 -2 0

-3 -4 0
-1 -3 0
-2 -4 0
-1 -4 0
-2 -3 0

- Solving via Python using library "pycryptosat":
  There is a python library where SAT SOLVER can also be implemented which is "pycryptosat". The below is the usage of pycryptosat in python:

  ```
  from pycryptosat import Solver
  s = Solver()
  s.add_clause([ //clauses to be added]) //add_clause() is a method, which takes input parameter as list
  sat, solution = s.solve()
  ```

  Here sat describes weather it is SATISFIABLE or NOT, if sastisfiable produces True, else produces False. Solution is a list which is of boolean format, if there exists a solution i.e solution is satisfiable, all queens in a position tends to put out True, else False. If there is no Solution it returns None

  Eg1: for n = 5
  sat = True
  solution = (None, False, False, False, True, False, False, True, False, False, False, False, False, False, False, True, False, False, True, False, False, True, False, False, False, False)

  Eg2: for n=3
  sat = False
  solution = None

4 Output Screenshots

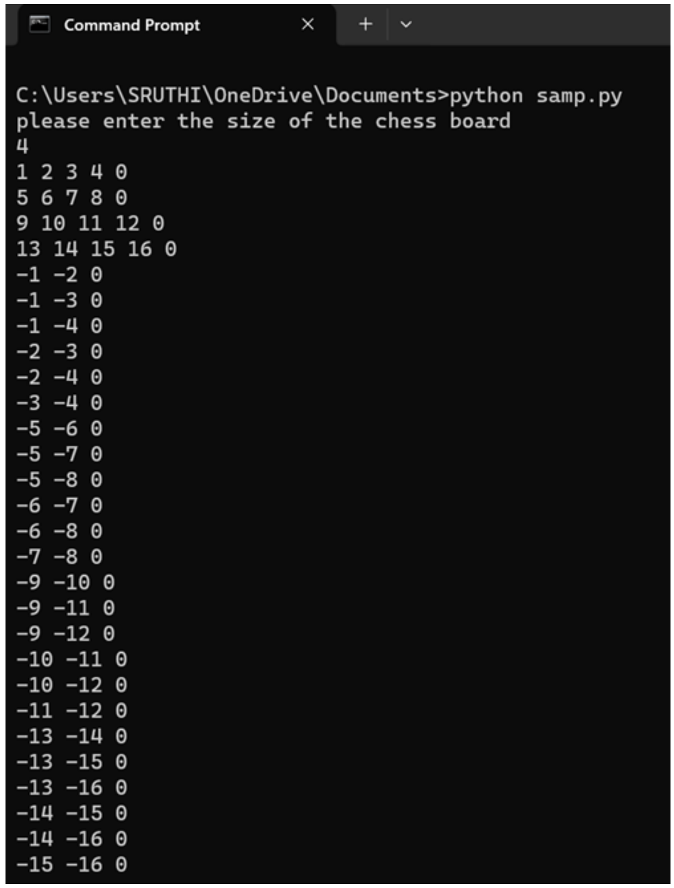

Figure 1: Input file to the sat solver

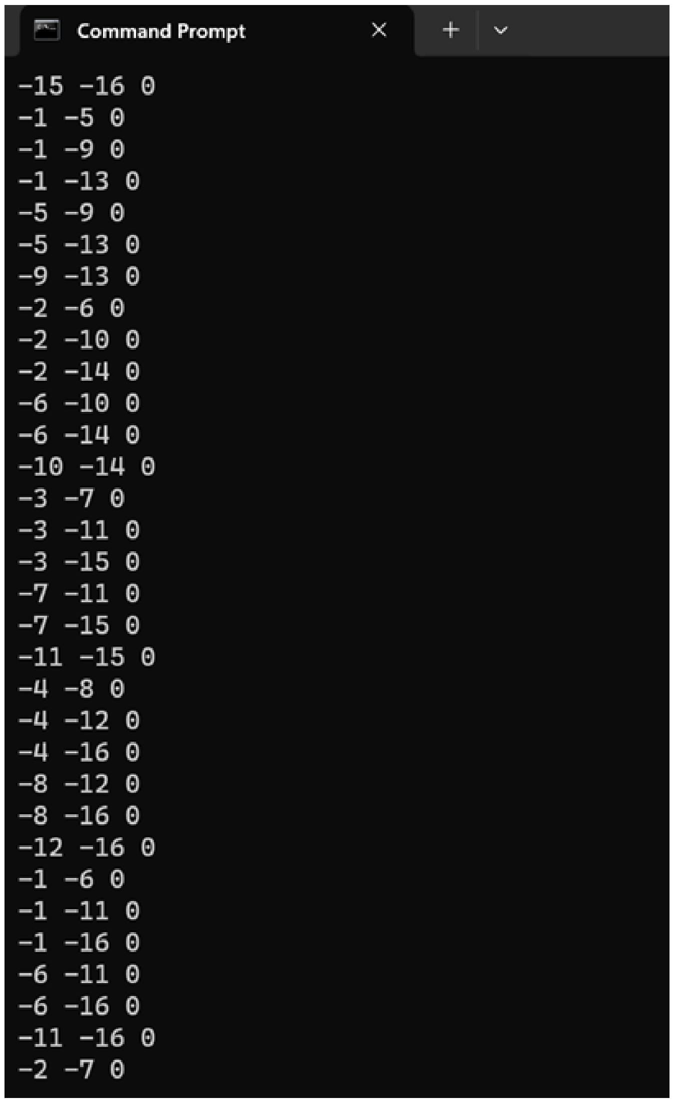

Figure 2: Input file to the sat solver

```
Command Prompt
-2 -7 0
-2 -12 0
-7 -12 0
-5 -10 0
-5 -15 0
-10 -15 0
-3 -8 0
-9 -14 0
-2 -5 0
-8 -11 0
-8 -14 0
-11 -14 0
-3 -6 0
-3 -9 0
-6 -9 0
-12 -15 0
-4 -7 0
-4 -10 0
-4 -13 0
-7 -10 0
-7 -13 0
-10 -13 0
```

Figure 3: Input file to the sat solver

```
Satisfiability: True
(None, False, True, False, False, False, False, False, True, True, False, False, False, False, False, True, False)
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
-1 2 -3 -4 -5 -6 -7 8 9 -10 -11 -12 -13 -14 15 -16
```

Figure 4: Output from the sat solver