Project. Satisfiability test of clauses and its application

Submit your program(s) and report to blackboard by **11:59pm Nov 7**. No late submission will be accepted.

We have studied several methods to check (prove) the unsatisfiability of propositions. In the last decades, significant progress has been made to build efficient systems, also called *SAT solvers*, to test the satisfiability of a given proposition in Conjunctive Normal Form (we know that any proposition can be translated into CNF). SAT solvers have found wide applications including hardware/software verification problems, diagnosis and planning problems.

1 Objectives and deliverable

In this project, you are expected to

- Form a team of exactly three people by **11:59pm Oct 3**. If you cannot find a partner, email our TA Vamsi by **11:59pm Sept 30** so that he can help you.
- Download a SAT solver and know how to install it and use it on any Unix (or Linux) machine of our department.
- Encode the n-queens (n is number input by users of your program) problem into the DIMACS CNF format and solve it by the SAT solver you have installed. Since the CNF format could be very large and complex, you may want to write a program to generate the CNF coding of the problem.
- Write a compact while clear report on how you encode the problem into the CNF form. Don't include your source code. You can use pseudo-code for your algorithm to generate CNF for n queens.
- Interview. There will be an interview of about 10 minute long. Our TA Vamsi will contact your team for the interview time later.

2 Background materials: file format and SAT solvers

The CNF input to a SAT solver is usually stored in a file with DIMACS CNF format. The input file starts with comments (each line starts with c). The number of variables and the number of clauses is defined by the line `p cnf #variables #clauses`. Each of the next lines specifies a clause: a positive literal is denoted by the corresponding number, and a negative literal is denoted by the corresponding negative number. The last number in a line should be zero. Here is an example.

```
c An example SAT formula:  {{A, ¬ B}, {¬ A, B}}.
c We have 2 propositional letters (variables) A and B.
c We use number 1 for A and 2 for B.
c p CNF #variables #clauses
p CNF 2, 2
1, -2, 0
-1, 2, 0
```

You may use the SAT solver miniSAT (http://minisat.se/Main.html) or whatever solver you like. From the github page (whose link is on the miniSAT website main page) you can find how a binary miniSAT solver is used.

3 Apply logic and a proof procedure to solve interesting problems

Here are some hints on encoding the n-queens problem into propositional logic. Consider 3 queens problem. We need to encode the following constraints 1) no two queens on the same row, 2) no two queens on the same column, and 3) no two queens on the same diagonal.

The objective is to find a formula whose satisfiable assignment gives us a solution to the n-queens problem. One idea is to introduce a proposition letter (variable) for each position on the 3X3 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).

To express the observation that there is a queen for first row:

$$P_{11} \vee P_{12} \vee P_{13}.$$

To express the constraint that there is no two queens on the first row: if any P_{1i} is true, then other P_{1j}'s of first row should be false, i.e.,

$$\begin{gathered} P_{11} \rightarrow \neg P_{12}, \\ P_{11} \rightarrow \neg P_{13}, \\ P_{12} \rightarrow \neg P_{11}, \\ P_{12} \rightarrow \neg P_{13}, \\ P_{13} \rightarrow \neg P_{11}, \\ P_{13} \rightarrow \neg P_{12}. \end{gathered}$$

You know how to translate $\rightarrow$ connective to $\vee$ connective.

You may note that to solve the n-queens problem, we are specifying the problem (using logic) instead of trying to find an algorithm to solve it. Once we work out the formula, the SAT solver can be employed to produce a solution to the formula (and thus the original problem) automatically. This project shows how logic and the proof procedures can be applied to solve academic (and real life) problems.