ESI Logical

Resolution Solver

Advanced Mathematical Logic Course

Practical Work Report

Team Members

Developed by students of 2CP06:

- **Boukeha Mohamed Akram**
- **Kharchi Merouane Abderrahmane**
- **Dali Braham Mohamed**
- **Kihal Ayoub**

Academic Year: 2024–2025

Supervisors: Meziani Lila, Khelifati Si Larbi, Faical Touka, Charabi Leila

1. Introduction

The goal of this project is to develop an automated resolution engine for propositional logic formulas expressed in Conjunctive Normal Form (CNF). The resolution method used is refutation by resolution, introduced by J.A. Robinson in 1965. This method allows determining whether a formula is satisfiable (SAT) or unsatisfiable (UNSAT) by attempting to derive the empty clause, a contradiction indicating inconsistency.

This report details the architecture of our solution, the design decisions taken, optimizations applied, and the results obtained, including screenshots and examples illustrating the resolution steps.

2. Implementation Choices

2.1 Language and Architecture

The application is divided into two main components:

- **Core Solver (`logic_solver.exe`):**
    - **Written in C.**
    - **Reads formulas in CNF from a text file.**
    - **Applies the resolution by refutation algorithm.**

- **Graphical User Interface (`gui_solver.exe`):**
    - **Developed in C.**
    - **Manages user interactions: formula creation, formula browsing, and testing.**
    - **Communicates with the solver through subprocess calls.**

2.2 Clause Representation

- **A clause is stored as a list of literals.**
- **Each literal is represented as a string (e.g., `P`, `!Q`).**
- **Clauses are stored in memory.**
- **The CNF file format contains one clause per line, literals separated by spaces.**

2.3 File Format (CNF)

Example file content:

P !Q R
!P S
T !U

This represents: (P v ¬Q v R) ^ (¬P v S) ^ (T v ¬U)

3. Resolution Algorithm

We implemented the resolution by refutation as follows:

1. **All clauses are loaded from the `.cnf` file.**
2. **For each pair of clauses, the algorithm checks if they can be resolved (if they contain complementary literals).**
3. **If a resolvent is generated, it is added to the set of clauses.**
4. **If the empty clause `[ ]` is derived, the formula is unsatisfiable.**
5. **If no new clauses can be produced, and the empty clause has not been found, the formula is satisfiable.**

4. Optimizations

4.1 Clause Redundancy Elimination

Duplicate clauses are ignored to reduce processing overhead.

4.2 Conflict Detection

Resolution terminates early when the empty clause is found.

4.3 Literal Indexing

Indexing of literals is used to quickly identify complementary pairs.

5. Graphical Interface

5.1 Features

- **Create and save new CNF formulas.**
- **Browse and preview stored formulas.**
- **Test formulas using the core solver.**
- **View satisfiability results.**

5.2 Usage

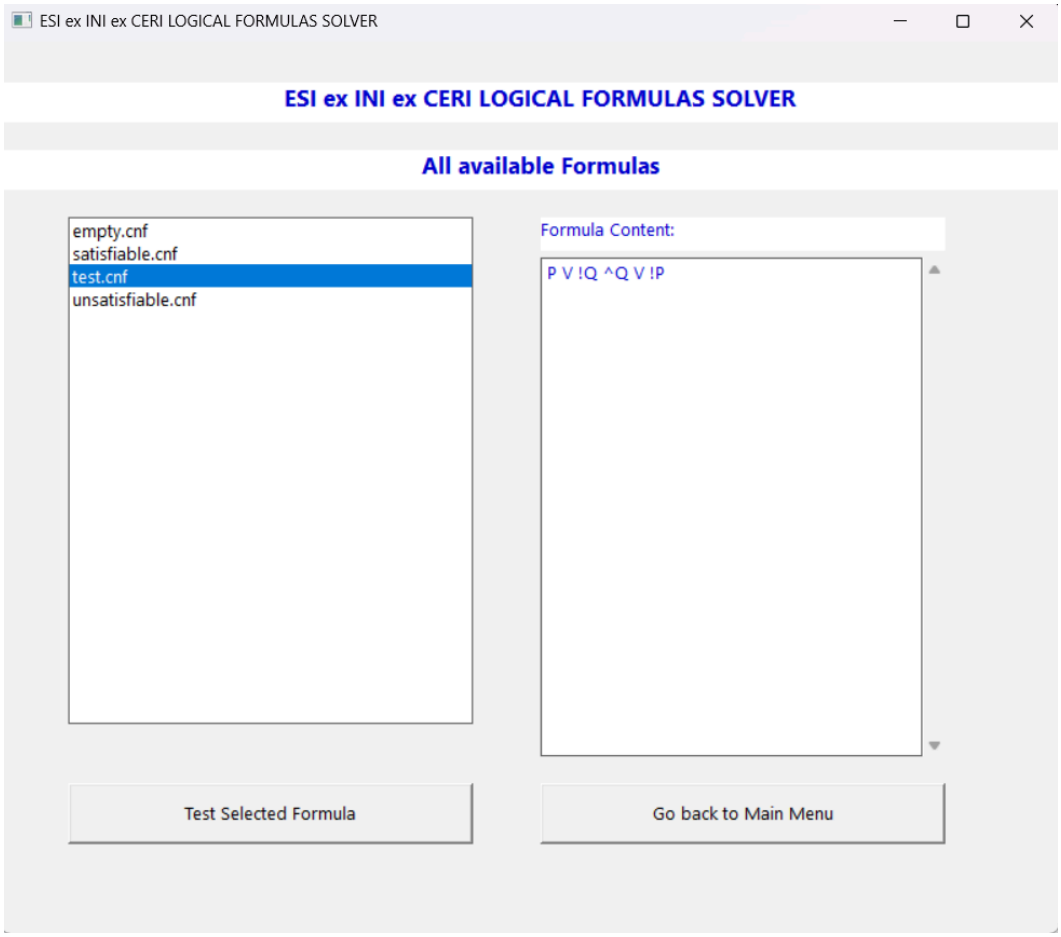
- **Run `gui_solver.exe`.**
- **Use the menu to create or test formulas.**
- **Formulas are saved as `.cnf` files in the same directory so there is no need to do it manually.**

6. Results and Demonstration

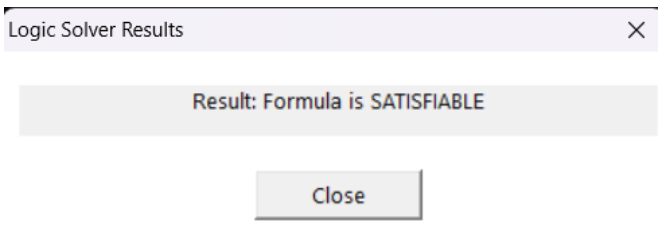
6.1 Example 1: Satisfiable Formula

Formula:

P V !Q
Q V !P



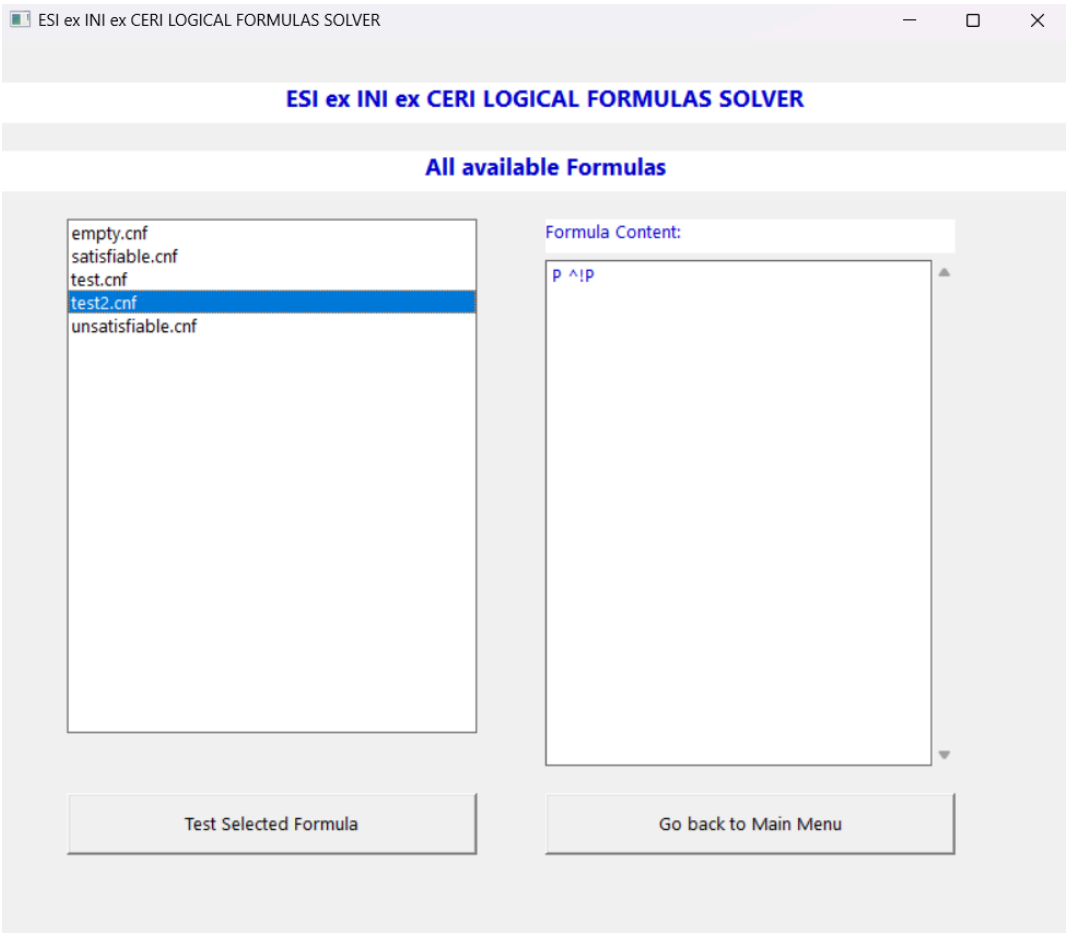
Result: Satisfiable



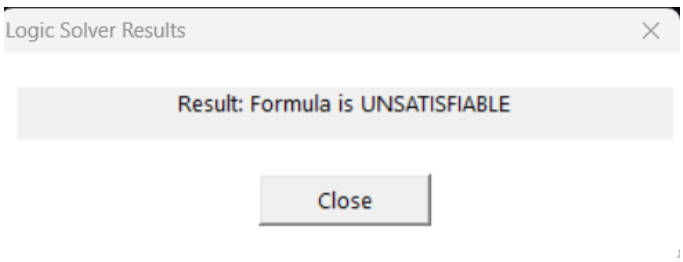
6.2 Example 2: Unsatisfiable Formula

Formula:

P
!P



Result: Unsatisfiable
Trace: Resolution of P and !P yields an empty clause.



7. Files and Artifacts

- `Program` :
  - `logic_solver.exe`: compiled C logic engine
  - `gui_solver.exe`: C GUI
  - `satisfiable.cnf`, `unsatisfiable.cnf`: sample formula files
- `Pictures-Screenshots` : folder containing interface captures
- `Livrable-Documents`: containing all necessary documentation

8. Conclusion

This project has allowed us to put into practice concepts from propositional logic and automatic theorem proving. We implemented a working resolution engine and a graphical tool for handling formulas efficiently. The results demonstrate the correctness of the algorithm and the usefulness of the tool for logic learning and problem modeling.