SAT Solver Implementation Report

June 12, 2025

Abstract

This report details the implementation of a SAT solver with a user-friendly interface, supporting both resolution-based satisfiability checking and conversion of propositional logic formulas to Conjunctive Normal Form (CNF). The solver is implemented in C, featuring a Windows-based console interface with colored output and ASCII art. We discuss the design choices, optimizations, and results, supplemented by placeholders for screenshots and resolution traces.

1 Introduction

The SAT solver, named *SatiChecker*, is designed to determine the satisfiability of a set of propositional logic clauses and convert formulas to CNF. Developed by Team Logic (Mekentichi NejemEddine, Kahlouche Youcef, Benbada Ayoub, Bouderbala Heythem), the solver provides an interactive interface for users to input clauses via text files, view documentation, and understand the algorithm's workflow.

2 Implementation Choices

The solver is structured into modular components to enhance maintainability and clarity. Below are the key implementation choices:

- **Modular Design**: The code is divided into separate files:
  - fileHandlers.c: Handles file operations (opening, closing, and reading clauses).
  - interface.c: Implements the console-based user interface with colored out- put and navigation.
  - stringHandlers.c: Manages clause parsing, resolution, and CNF conversion.
  - main.c: Orchestrates the program flow and integrates all components.
- **Data Structures**:
  - A Clause structure stores a clause string and an array of variables (up to 26, mapped to letters A-Z).
  - Dynamic memory allocation is used for clauses and variables to handle varying input sizes.

- **User Interface**:
  - Windows-specific console functions (windows.h, conio.h) for colored output and key navigation.
  - ASCII art and centered text for a visually appealing interface.
  - Menu-driven navigation with arrow key support for selecting options.
- **Input Format**:
  - Clauses are read from a text file in CNF (e.g., A|B|!C & D|!E & F).
  - For CNF conversion, inputs can include implications (>) and equivalences (=), with parentheses for grouping.
- **Algorithmic Approach**:
  - Resolution algorithm for satisfiability checking, based on detecting empty clauses.
  - CNF conversion follows a step-by-step process: eliminating equivalences, implications, pushing negations inward, and distributing OR over AND.

3 Optimizations

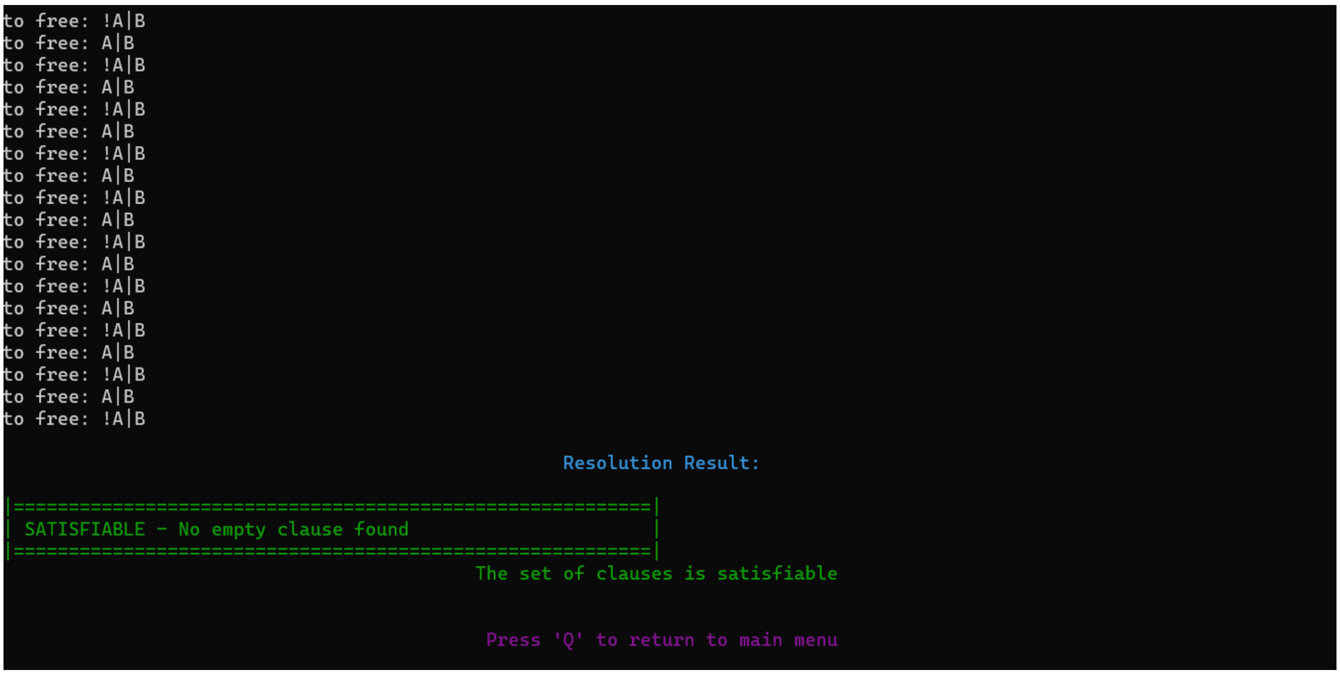
Several optimizations were implemented to improve performance and usability:

- **Memory Management**:
  - Dynamic allocation with malloc and realloc for clauses and variables to handle varying input sizes.
  - Careful freeing of memory in freeClause to prevent leaks.
- **Efficient Clause Resolution**:
  - The setResolution function avoids redundant resolvents by checking for duplicates using setHas.
  - A maximum iteration limit (100) prevents infinite loops in complex cases.
  - A 3-minute timeout ensures the program does not hang on large inputs, returning a ”likely satisfiable” result.
- **Input Validation**:
  - verifySet and verifySetWithConversion ensure valid syntax for resolution and CNF conversion, respectively.
  - Error messages guide users to correct input formats.
- **User Experience**:
  - Colored output distinguishes different types of information (e.g., red for errors, green for satisfiable results).
  - Step-by-step display of CNF conversion enhances transparency.

4 Results

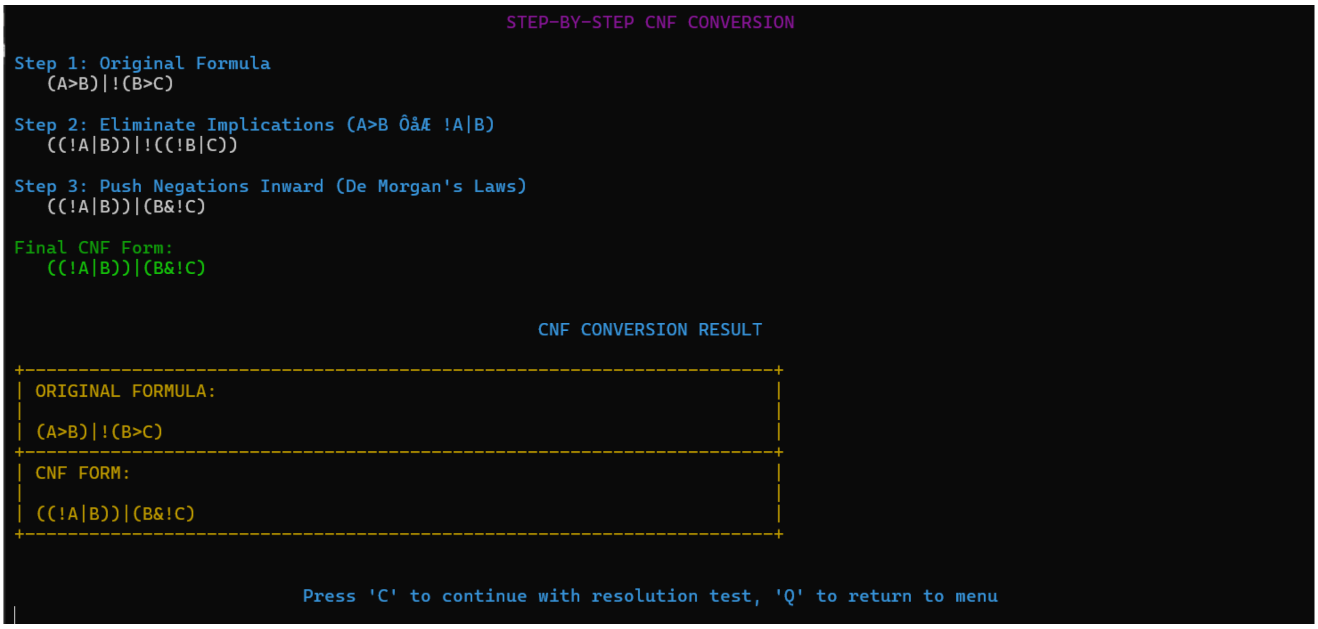
The solver was tested with various inputs to evaluate its functionality. Below are key results, with placeholders for screenshots and traces:

- **Satisfiability Checking**:
  - Input: A|B|!C & D|!E & F
  - Result: Satisfiable (no empty clause derived).



```
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B
to free: A|B
to free: !A|B

                                                            Resolution Result:

|==========================================================|
| SATISFIABLE - No empty clause found                      |
|==========================================================|
                                                    The set of clauses is satisfiable


                                                     Press 'Q' to return to main menu
```

- **CNF Conversion**:
  - Input: (A>B)|!(B>C)
  - Output: (!A|B)|(B&!C)



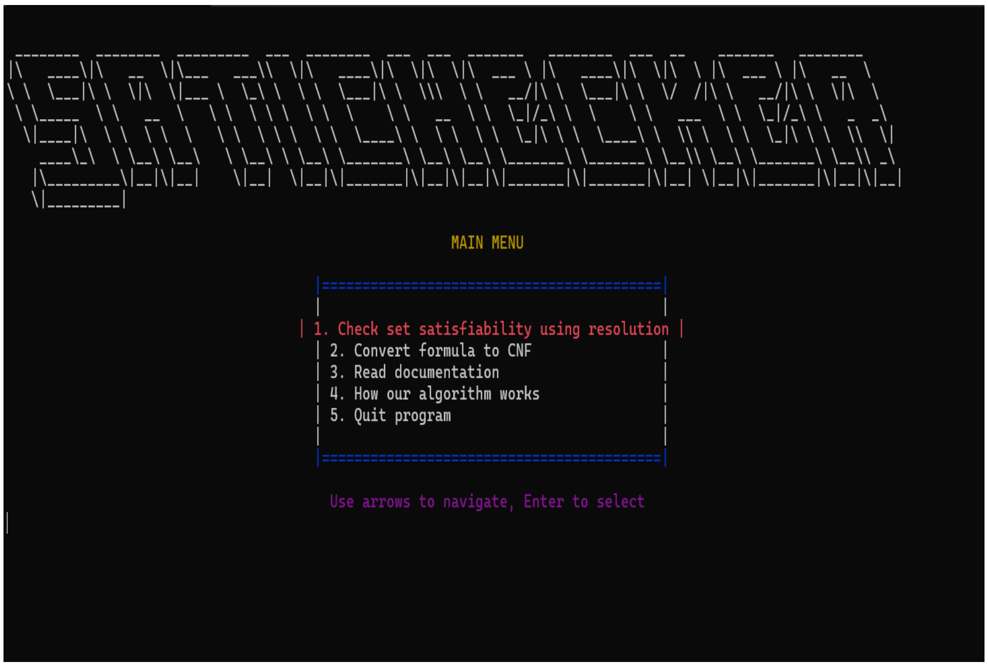
```
                                                 STEP-BY-STEP CNF CONVERSION

Step 1: Original Formula
   (A>B)|!(B>C)

Step 2: Eliminate Implications (A>B ÔåÆ !A|B)
   ((!A|B))|!((!B|C))

Step 3: Push Negations Inward (De Morgan's Laws)
   ((!A|B))|(B&!C)

Final CNF Form:
   ((!A|B))|(B&!C)


                                                    CNF CONVERSION RESULT

+----------------------------------------------------------------------+
| ORIGINAL FORMULA:                                                    |
|                                                                      |
| (A>B)|!(B>C)                                                         |
+----------------------------------------------------------------------+
| CNF FORM:                                                            |
|                                                                      |
| ((!A|B))|(B&!C)                                                      |
+----------------------------------------------------------------------+


                           Press 'C' to continue with resolution test, 'Q' to return to menu
```

5 Challenges and Solutions

- **Challenge**: Handling complex formulas with nested parentheses in CNF conversion.
  - **Solution**: Implemented `findMatchingParen` to correctly identify matching parentheses and process nested expressions.
- **Challenge**: Preventing memory leaks in dynamic clause allocation.
  - **Solution**: Rigorous memory management with `freeClause` and checks for allocation failures.
- **Challenge**: Ensuring user-friendly error handling.
  - **Solution**: Clear error messages with color coding and references to documentation.

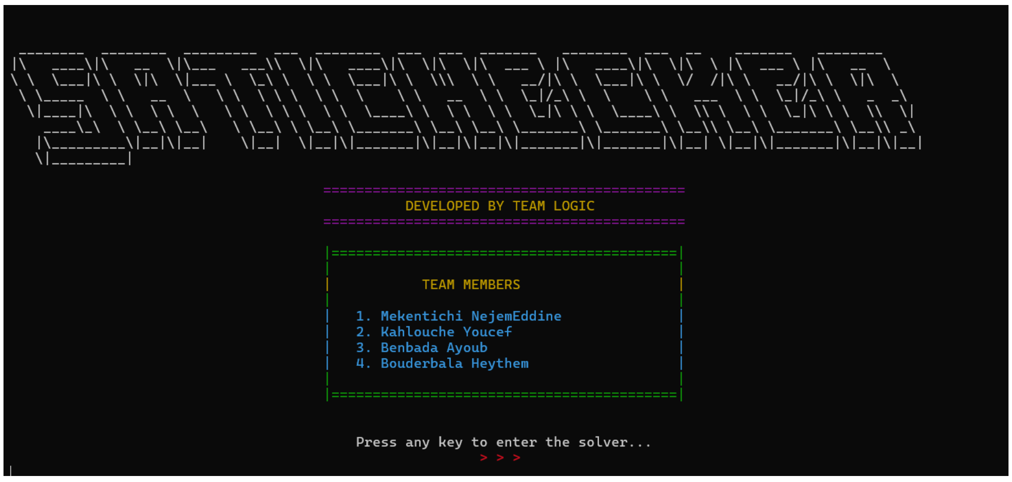
6 Conclusion

The *SatiChecker* SAT solver successfully implements resolution-based satisfiability checking and CNF conversion with a user-friendly interface. The modular design, efficient memory management, and robust error handling make it a reliable tool for propositional logic tasks. Future improvements could include support for larger variable sets and optimized resolutions strategies



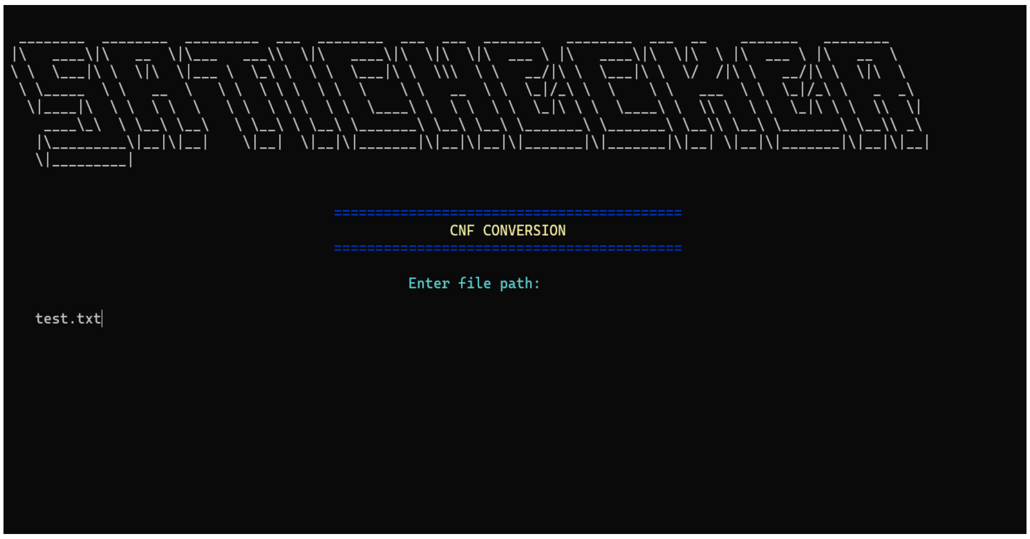
```
                              MAIN MENU

                |=========================================|
                |                                         |
               | 1. Check set satisfiability using resolution |
                | 2. Convert formula to CNF               |
                | 3. Read documentation                   |
                | 4. How our algorithm works              |
                | 5. Quit program                         |
                |                                         |
                |=========================================|

                  Use arrows to navigate, Enter to select
```

Figure 1: Main Menu Interface



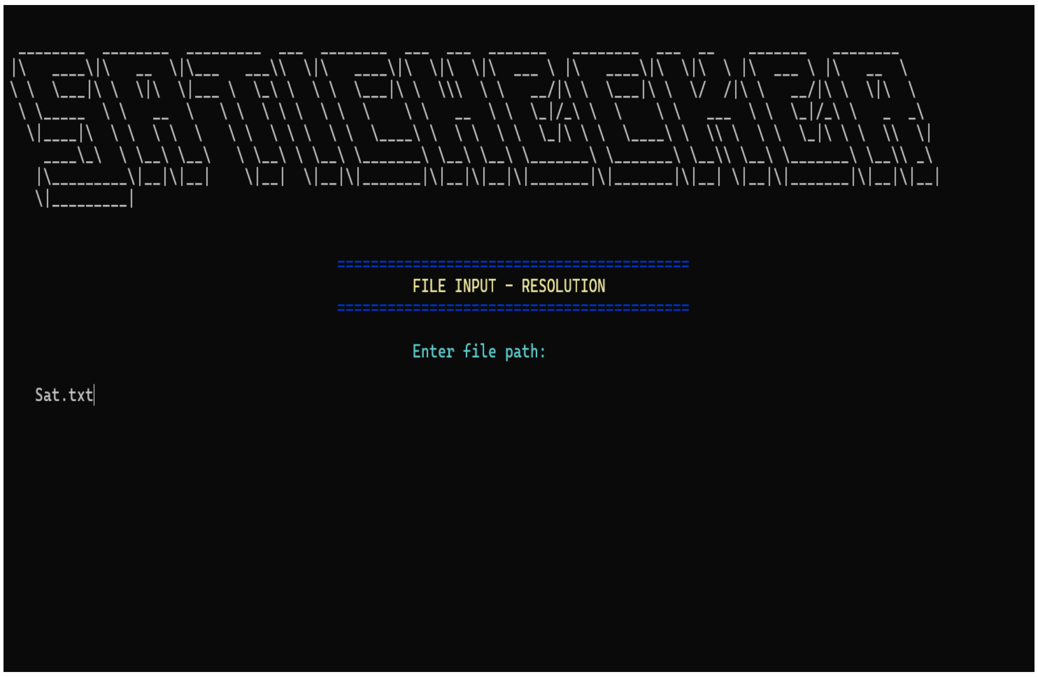
```
              ============================================
                         DEVELOPED BY TEAM LOGIC
              ============================================

              |=========================================|
              |                                         |
              |              TEAM MEMBERS               |
              |                                         |
              |   1. Mekentichi NejemEddine             |
              |   2. Kahlouche Youcef                   |
              |   3. Benbada Ayoub                      |
              |   4. Bouderbala Heythem                 |
              |                                         |
              |=========================================|

                    Press any key to enter the solver...
                                  > > >
```

Figure 2: Team Splash Screen



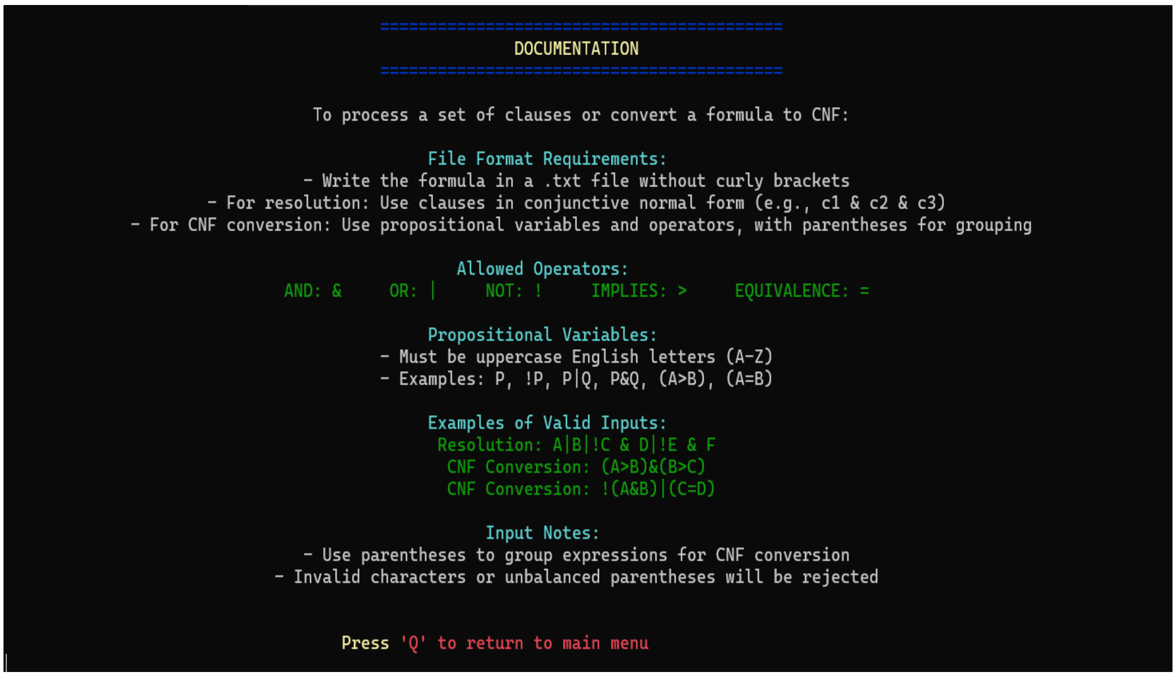
```
CNF CONVERSION

Enter file path:

test.txt
```

Figure 3: CNF Conversion Screen



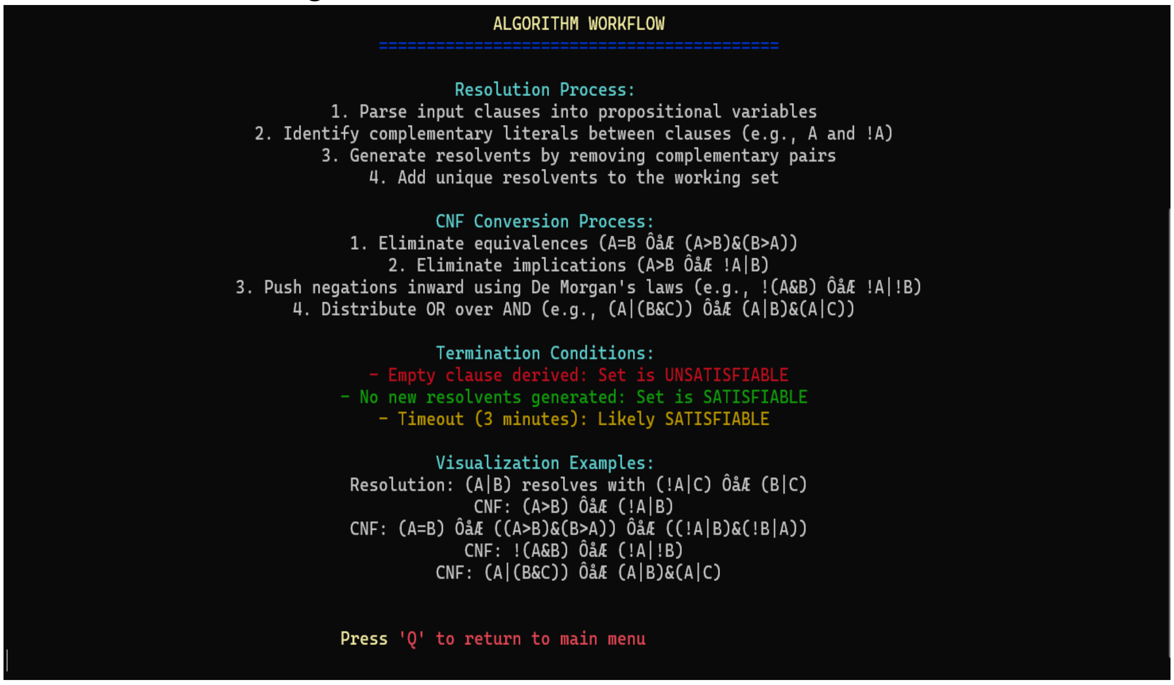
```
FILE INPUT - RESOLUTION

Enter file path:

Sat.txt
```

Figure 4: Resolution Satisfiability Checker Screen



```
==========================================
              DOCUMENTATION
==========================================

To process a set of clauses or convert a formula to CNF:

File Format Requirements:
- Write the formula in a .txt file without curly brackets
- For resolution: Use clauses in conjunctive normal form (e.g., c1 & c2 & c3)
- For CNF conversion: Use propositional variables and operators, with parentheses for grouping

Allowed Operators:
AND: &     OR: |     NOT: !     IMPLIES: >     EQUIVALENCE: =

Propositional Variables:
- Must be uppercase English letters (A-Z)
- Examples: P, !P, P|Q, P&Q, (A>B), (A=B)

Examples of Valid Inputs:
Resolution: A|B|!C & D|!E & F
CNF Conversion: (A>B)&(B>C)
CNF Conversion: !(A&B)|(C=D)

Input Notes:
- Use parentheses to group expressions for CNF conversion
- Invalid characters or unbalanced parentheses will be rejected


Press 'Q' to return to main menu
```

Figure 5: Documentation Screen



```
ALGORITHM WORKFLOW
==========================================

Resolution Process:
1. Parse input clauses into propositional variables
2. Identify complementary literals between clauses (e.g., A and !A)
3. Generate resolvents by removing complementary pairs
4. Add unique resolvents to the working set

CNF Conversion Process:
1. Eliminate equivalences (A=B ÔåÆ (A>B)&(B>A))
2. Eliminate implications (A>B ÔåÆ !A|B)
3. Push negations inward using De Morgan's laws (e.g., !(A&B) ÔåÆ !A|!B)
4. Distribute OR over AND (e.g., (A|(B&C)) ÔåÆ (A|B)&(A|C))

Termination Conditions:
- Empty clause derived: Set is UNSATISFIABLE
- No new resolvents generated: Set is SATISFIABLE
- Timeout (3 minutes): Likely SATISFIABLE

Visualization Examples:
Resolution: (A|B) resolves with (!A|C) ÔåÆ (B|C)
CNF: (A>B) ÔåÆ (!A|B)
CNF: (A=B) ÔåÆ ((A>B)&(B>A)) ÔåÆ ((!A|B)&(!B|A))
CNF: !(A&B) ÔåÆ (!A|!B)
CNF: (A|(B&C)) ÔåÆ (A|B)&(A|C)


Press 'Q' to return to main menu
```

Figure 6: Workflow Screen