Implementing a SAT Solver with Advanced Heuristics

Geromy Cunningham
Jonathan Lloyd
ECE51216: Digital Systems Design Automation – Group 22

I. Introduction & Problem Statement

Propositional satisfiability (SAT) solvers are a fundamental tool in computational logic and artificial intelligence. The Davis-Putnam-Logemann-Loveland (DPLL) algorithm is the basis of many modern SAT solvers, and its efficiency can be significantly improved through heuristic optimizations.

The goal of the project is to develop a SAT solver that takes input as a Boolean formula in Conjunctive Normal Form (CNF) following the DIMACS format and outputs either a satisfying variable assignment or confirmation that the formula is unsatisfiable. We propose implementing a SAT solver based on the DPLL backtracking algorithm with two advanced heuristics: watched literals and non-chronological backtracking. If time permits, we will extend our implementation to include conflict-driven clause learning (CDCL), further enhancing performance.

II. Motivation

Implementation of the DPLL algorithm from scratch with heuristics provides valuable insights into search algorithms and logical inference. The heuristics that we chose allow for clear performance improvements while keeping the implementation manageable within the project timeline. Watched literals assists in unit propagation by maintaining "watched literals" per clause, reducing the number of checks needed. Non-chronological backtracking allows the program to jump directly to the deepest level responsible for a conflict instead of sequentially stepping back. CDCL extends the DPLL by learning from conflicts. When a contradiction is encountered, a new clause is derived that prevents future mistakes.

III. Methodology

Our SAT solver will be implemented in C/C++ and possibly some Python, leveraging efficient data structures for heuristic optimizations.

The key components of our approach include:

- **DPLL Algorithm:** standard DPLL with unit propagation and pure literal elimination that will allow simple backtracking-based search for satisfying assignments.
- **Watched Literals:** Optimize unit propagation by tracking only two literals per clause.
- **Non-Chronological Backtracking:** Conflict analysis to determine the correct backtracking point.
- **CDCL (Time-Permitting):** Include clause learning or generating new clauses to prevent repeated conflicts.
- **Input and Output handling:** Parse CNF formulas from DIMACS format using an existing parser. Output a satisfying assignment if one exists; otherwise, declare unsatisfiability.
- **Benchmarking and Testing:** Use public benchmark repositories to evaluate solver performance. Compare results against the basic DPLL implementation to measure heuristic effectiveness.

IV. Expected Challenges

- **Data Structures:** Implementing watched literals will require careful management of data structure manipulation to efficiently and correctly track watched literals and their original clauses.
- **Memory Management:** Efficiently store and manage modified/original clauses avoiding excessive memory usage.
  - CDCL will also need to store and manage learned clauses similarly.
- **Conflict Analysis:** Ensuring correct non-chronological backtracking requires updated and accurate conflict analysis methodologies and mechanisms.
  - Further if we get to CDCL, there will need to be a method determining the first Unique Implication Point (UIP).
- **Balancing heuristics** to optimize performance without excessive overhead costs.

V. Expected Outcomes

A functional SAT solver implementing DPLL with watched literals and non-chronological backtracking will be delivered. If time permits, an extension to CDCL for further optimization. Performance evaluation of heuristics will show improvements upon the standard DPLL datasets. A well-documented codebase and report summarizing findings and unexpected challenges encountered.

VI. Timeline

Week	Milestone
17Mar – 23Mar	Research further into methodologies and planning.
24Mar – 30Mar	Standard DPLL implementation.
31Mar – 6Apr	Implement Watched Literals & Non-Chronological Backtracking.
7Apr – 10Apr	Work on Intermediate Report.
11Apr – 20Apr	Finish Non-Chronological Backtracking (CDCL if time).
21Apr – 27Apr	Debug (finish CDCL).
28Apr – 4May	Test data analytics.
5May – 9May	Complete Final Report.