CS 7240 Intermediate Project Report: CDCL SAT Solver Implementation

William Schultz

April 18, 2022

1 Overview

Satisfiability (SAT) solvers have become immensely powerful tools for solving hard, generic constraint satisfaction problems. Even though the SAT problem is known to be fundamentally hard (NP-complete), these tools are now effective at solving large, nontrivial real world problem instances and are applied widely in hardware and software verification, program analysis, electronic design automation, etc. The goal for this project will be to implement a SAT solver based on relatively state of the art techniques, which are mostly based on *conflict driven clause learning* (CDCL) [8, 6, 13], an extension of the foundational DPLL algorithm [9]. The goal is to implement the solver and compare its performance to other state of the art solvers.

2 Project Details

The overall goal is to implement a CDCL based SAT solver in C++ that accepts input in DIMACS CNF format [1]. At a high level a SAT solver takes in a boolean formula F in *conjunctive normal form (CNF)* and returns a result of either *sat*, along with a satisfying assignment that makes the given formula true, or *unsat*, meaning no such assignment exists. In the case that the formula is satisfiable and a satisfying assignment is returned, it is easy to verify whether this result is correct. If a result of *unsat* is returned, though, it may still be desirable to check the validity of this result. So, most modern solvers also return a refutation of F which can be independently verified.

This can be returned in the form of a resolution refutation [7] proof. Our goal will be to also produce such proofs in the case that F is unsatisfiable. We will aim to base our implementation and algorithms off of the treatments found in [8].

Progress Report (April 19, 2022)

Completed milestones:

- Basic working DPLL SAT solver implementation with unit resolution
- Test harness that checks correctness of solver implementation on randomly generated CNF formulas against brute force SAT solving method
- Visualization of termination tree (i.e. search tree) as DOT graph

Remaining work items:

- Implement conflict driven clause learning (CDCL) on top of the basic DPLL backtracking algorithm.
- Implement ability to generate ceritifcates of unsatisfiability via resolution proofs. We plan to generate proofs based on the RUP (Reverse Unit Propagation) format [10, 12].
- Use the DRAT-trim checker tool to test the correctness of our unsat proofs [11].
- Evaluate our CDCL SAT solver implementation on varied benchmark set and compare its performance to Kissat[2], which won the main track of SAT 2020 competition.

3 Evaluation Plan

We will evaluate our solver's performance on a standard set of SAT benchmarks and compare with other state of the art solvers, drawing from winners of the recent SAT 2020 competitions [5]. The solvers *kissat*[2], *cryptominisat* [14], and *minisat*[3] seem to be good targets to compare against. The SAT 2020 competition publishes the benchmarks used for the competition [4], so

we plan to use these as a starting point for evaluation. If all of these are too hard for our solver, though, then we can resort to some other, hand generated benchmarks, or another set of various CNF benchmarks hosted at [1].

References

[1] CNF Files. https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html.

[2] Kissat SAT Solver. http://fmv.jku.at/kissat/.

[3] MiniSat SAT Solver. http://minisat.se/.

[4] SAT Competition 2020: Benchmarks. https://satcompetition.github.io/2020/benchmarks.html.

[5] SAT Competition 2020 Results. https://satcompetition.github.io/2020/results.html.

[6] Conflict-Driven Clause Learning SAT Solvers. https://www.cs.princeton.edu/~zkincaid/courses/fall18/readings/SATHandbook-CDCL.pdf, 2008.

[7] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer Publishing Company, Incorporated, 3rd edition, 2012.

[8] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, NLD, 2009.

[9] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, jul 1960.

[10] E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for cnf formulas. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 886–891, 2003.

[11] Marijn J.H. Heule. DRAT-trim. https://www.cs.utexas.edu/~marijn/drat-trim/.

[12] Marijn J.H. Heule, Warren A. Hunt, and Nathan Wetzler. Trimming while checking clausal proofs. In *2013 Formal Methods in Computer-Aided Design*, pages 181–188, 2013.

[13] J.P. Marques Silva and K.A. Sakallah. GRASP-A new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227, 1996.

[14] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 244–257, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.