An efficient SAT solver verified in Lean

Kristoffer Aandahl

June 2025

1 Introduction

Formal verification of code is the field of computer science trying to prove properties about software systems. These properties might include aspects such as functional correctness, termination or memory safety. To do this, we compile code into logical statements, with which we can validate our desired properties. Proving these properties, however, can be difficult even with the logical statements in front of us. In the pursuit of automation, we therefore turn to boolean satisfiability solvers (SAT) [1]. Lean is a combined automated theorem prover and a programming language. It is therefore frequently used to make formally verified code, due to its ability to reason about its own code. During this reasoning, the use of SAT solvers are often used to prove certain facts. However, the solvers used are all in external languages. This means that verified code in Lean sometimes relies on unverified code to reason about correctness. It is therefore the goal of this thesis to make a verified SAT solver in Lean, that performs well enough that it can be used in exchange for these nonnative solutions.

This will not be the first Sat solver in lean, as there already exists at least one other SAT solver in Lean (Saturn)[2]. However, it does not use the more modern conflict driven clause learning (CDCL) algorithm, which the currently best performing solvers are based on. Implementing a CDCL based Lean solver will hopefully make it perform well enough that it in some cases makes sense to switch over to this thesis's solver.

It is not expected that this solver, will be the most efficient or best performing solver. But rather make a solver performing in the same class as other solvers. Which, might make it an option to the other solvers, in cases where, bleeding edge performance is not mandatory.

2 Related works

2.1 CaDiCaL

CaDiCaL[2] is a CDCL incremental SAT solver. CaDiCaL is made with the purpose of being a well documented, clean and easy to work with SAT solver. It is also built with the purpose to be easy to expand and as a result a lot of the top performing SAT solvers in the 2024 sat solving competitions, including the winner Kissat-sc2024, were derivatives of CaDiCaL [3]. CaDiCaL will be used as a reference implementation for this SAT solver, and the algorithms used should mirror what CaDiCaL does. CaDiCaL should therefore be helpful in providing well performing heuristics and acceleration techniques, as well as the fundamental CDCL algorithms. Lastly, Lean can also make calls directly to CaDiCaL through the BVDecide proof tactic.

2.2 IsaSAT

IsaSAT[4] is another formally verified sat solver. This one is made in Isabelle and is compiled using the Isabelle LLVM. IsaSAT is currently the best performing verified SAT solver, and is also based on CDCL. Even though IsaSAT is built in Isabelle, it will likely have encountered similar difficulties as this thesis's solver. It will therefore likely be relevant to consult with IsaSAT to find strategies to prove certain facts or deal with other difficulties in the quest of verifying a SAT solver.

2.3 SATurn

SATurn[5] is a verified SAT solver in Lean, based on the Davis–Putnam–Logemann–Loveland (DPLL) algorithm. While the algorithm is different, the fact that it is made in Lean makes it interesting, as it should be possible to learn from the approaches used in this implementation. DPLL is a simpler and less optimized algorithm than CDCL, but shares fundamental similarities. It is therefore not unlikely that some concepts from Saturn can be reused. SATurn is made by Siddhartha Gadgil, a professor in mathematics at the Indian Institute of Science. However, no scientific papers on the solver, has been found.

3 Approach

This project will attempt to make a verified SAT solver in Lean, based on the existing CDCL algorithm. As a first step to achieve this, the solver will have to be implemented and the Isabell based solver, IsaSAT, will likely be a source of inspiration, for how to implement it in a verifiable manner. Proving correctness of the solver, will rely on proving two aspects, soundness, that the solver only return true in cases where there exists a solution, and secondly proving completeness, that the solver always return an answer. Both of these aspects have been proven before, and this thesis will most likely use similar proof outlines as SATurn and IsaSAT. While SATurn uses a different algorithm than will be used in this project, the fact that it is made in Lean, might provide helpful insight into the process of making a solver in Lean. Once a solver has been made and verified, it should be benchmarked. Benchmarks will be pulled from the annual SAT competition. It is therefore important that the solver, supports the input and output formats used in these benchmarks. During benchmarking, it is a goal to quantify how big of a performance penalty switching to Lean is, and the thesis aims, therefore, to benchmark against similar solvers, in other languages. If the project still have time after implementation and benchmarking, it would be of interest to implement and verify some acceleration techniques. There exists, multiple heuristics and preprocessing techniques to accelerate SAT solvers and implementing some in a verified manner. We know that some of these techniques are crucial in the top performing solvers. In particular, is preprocessing techniques crucial for efficiently finding and proving unsatisfiability.

4 Goals

1. **Implement a CDCL based SAT solver in Lean.** The main goal is to implement a verifiable SAT solver in lean. The solver should follow the conflict driven clause learning (CDCL) approach.

2. **Verify soundness of the SAT solver.** The most important aspect to verify for a solver is that it only return true if a given problem has a solution.

3. **Verify completeness of the SAT solver.** The second most important aspect is to verify that it always terminates with an answer.

4. **Benchmark the algorithm.** While the performance of the solver is not the main goal, knowing the relative performance, especially to related solvers, will be interesting. While the solver will be proven to always return the correct answer, this might be uninteresting if the time it takes to provide answers is significantly worse than other solvers.

5. **Optimize the solver.** The solver will likely never be as efficient as the top performing solver. Sill the thesis aims to make a efficient enough solver that it makes sence to adopt it. Doing this it should at least perform as well or better than exisiting lean based solutions and ideally as well the best performing fomrally verified solvers.

6. **Support standard input and output formats.** SAT solvers use standardized formats for inputs and outputs, the solver should support these, such that it can be used without extra effort in place of other solvers.

5 Extension goals

1. **Accelerate the solver.** There exists multiple heuristics and preprocessing techniques used to accelerate CDCL based SAT solvers. To achieve as good performance as possible, incorporating some of these tactics, will be vital.

2. **Support the SAT competition specifications.** Ideally, the solver, should match the specifications of the SAT competition. And if timelines for the competition and thesis align, the solver should be signed up to the competition.

3. **Integration with other Lean packages.** Lean supports multiple packages, in which this project might be of interest. Packages such as BVdecide, for example, already call external SAT solvers, to prove sat problems. Integrating a Lean based solver with this package would therefore make sense. Grind a package aiming at satisfiability modulo theories (SMT), might also make sense to make integrations towards.

4. **Simple SMT theories.** If the main SAT solver is completed, attempting to make formal verification over theories, such as equality logic, would be interesting. This would make the solver capable of solving some SMTs, making the solver more capable and potentially usable.

References

[1] Kroening, *Decision Procedures*, eng. Springer Berlin Heidelberg, 2016, ISBN: 3662504960, 3662504979, 9783662504963, 9783662504970.

[2] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL 2.0," in *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, A. Gurfinkel and V. Ganesh, Eds., ser. Lecture Notes in Computer Science, vol. 14681, Springer, 2024, pp. 133–152. DOI: 10.1007/978-3-031-65627-9_7.

[3] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024," in *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., ser. Department of Computer Science Report Series B, vol. B-2024-1, University of Helsinki, 2024, pp. 8–10.

[4] M. Fleury, J. C. Blanchette, and P. Lammich, "A verified sat solver with watched literals using imperative hol," in *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ser. CPP 2018, Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 158–171, ISBN: 9781450355865. DOI: `10.1145/3167080`. [Online]. Available: `https://doi.org/10.1145/3167080`.

[5] S. Gadgil and A. Rao. "Saturn." (), [Online]. Available: `https://github.com/siddhartha-gadgil/Saturn/`.