CS8803 (CS4803) - SAT and SMT Solvers: Theory, Practice, and Applications

Vijay Ganesh

April 1, 2026

1 Course Introduction

The course described here is a graduate-level in-depth course on the theory and practice of Boolean SAT and SMT solvers, as well as their applications in software engineering (broadly construed to include testing, formal methods, analysis, synthesis, program understanding, etc.), programming languages design and implementation, and trustworthy AI.

There is wide recognition in academia and industry that solvers have been deeply impactful, have become integral to many PL/SE/FM (programming languages/software engineering/formal methods) approaches over the last two decades, and will likely have a deep impact on the field of trustworthy AI. In addition to their utility in many application settings, solvers constitute an interesting research topic in and of themselves, both from a theory (computability and complexity theory) and algorithmic perspective (solver algorithms, design choices, and important implementation details).

It is clear that a working knowledge of solvers can be very useful to any PL/SE/FM and AI graduate students. Hence, the goal of the course is to give interested graduate students a thorough and in-depth understanding of solvers and their applications.

2 Learning Objectives

The goal of the course is for students to gain mastery the following topics:

1. A deep understanding of SAT solvers, how to extend them, and how to effectively use them to solve their software engineering and trustworthy AI problems.

2. A working understanding of SMT solvers and how to effectively use them to solve their software engineering and AI security and reliability problems.

3. A working understanding of Verification of Neural Network (VNN) solvers and how to effectively use them to solve their AI security and reliability problems.

4. An appreciation of the (complexity and computability) theory that underpins solver technology.

3 Course Syllabus

The course is divided into the following 3 broad sub-topics:

- **Theory - Boolean logic, first-order logic, first-order theories, and proof complexity:** The goal here is to familiarize the students with the theoretical foundations of solvers. I plan to cover Boolean and first-order logic (syntax, semantics, proof systems, and theories), with a detailed set of lectures on properties of proof systems such as soundness, completeness, compactness, decidability, and complexity. I will also discuss certain relevant first-order theories such as Presburger and Peano arithmetic, mixed integer linear arithmetic, bit-vector, and strings) and their properties (e.g., axiom systems, incompleteness, decidability, and complexity). While students are supposed to have a rudimentary knowledge of complexity theory (e.g., classes such as P and NP), I will provide an introduction to proof complexity. Finally, I plan to discuss how these ideas came about, the contexts in which they were first invented, their evolution over time, and how they went from being purely philosophical topics to forming the foundation of mathematics and computer science.

- **Solver algorithms and architectures - DPLL and CDCL SAT solvers, programmatic SAT, SAT+CAS, SMT solvers, lazy and eager, Nelson-Oppen combination, CDCL(T), decision procedures for SMT theories, VNN solvers:** The goal here is to familiarize students with a detailed working knowledge of the internal algorithms and architecture (i.e., implementation details) of solvers. Specifically, there will be a very sharp focus on the CDCL method and its extensions (CDCL(T) and programmatic SAT), as well as decision procedures for certain theories such as bit-vectors and strings. Learning about programmatic SAT can be particularly useful for PLSE students, as it is one of the most powerful ways to extend a SAT solver. Time permitting, I also plan to discuss practical proof formats such as DRAT.

- **Solver applications in PLSE and AI - symbolic execution and testing, model checking, and neural network verification:** A course on solvers is incomplete without a thorough discussion of their applications. Specifically, I plan to discuss symbolic execution and testing, model checking (e.g., PDR), and verification of neural networks. To cover this topic properly one needs to discuss how constraints are generated by these

applications, encoding formats, proof objects, and how solvers are used in these contexts.

4 Marking Scheme

The students will be evaluated based on the following criteria.

4.1 Project

The course is designed to be hands-on and project heavy. I have created a detailed list of projects and open problems from theory to solver algorithms to applications. The students may choose a project from this list or come up with their own. 75% of the total marks will be project related.

- At the beginning of the term they will submit a project proposal (up to 5 pages long) that describes the problem, its importance, and how they plan to solve it. Obviously, they have to argue why their project is meaningful in the context of a SAT/SMT solver course. (5% of total grade.)
- Additionally, the students will be required to write a 10-15 page report in LNCS format (e.g., SAT/CAV papers) and make a 45 minute presentation on their projects towards the end of the term. I will also review their code and proofs. (70% of the total grade.)
- The project will be marked primarily for novelty and execution. I have created a detailed write up on how the students have to go about completing their project proposals, reports, and slides.

4.2 Assignments

I plan to have 2 assignments, and they will constitute 10% of the total marks. The assignments will focus on a mix of theory and practice questions.

4.3 Final Exam

I believe that exams are an important way to evaluate students for any course. Hence, we have a final exam with a significant percentage of the total marks (15%) allocated to the final exam.

5 Course Materials

I have an extensive slide deck that covers all of the material outline above (based on work by Isil Dillig and others, extended by me, and used with their generous permission). I have used and extended this slide deck for over 10 years. In addition, students may choose to consult with the following books and reading

materials. However, the slide deck would be more than sufficient to get a good grade in the course.

- Leary and Kristiansen Book: A Friendly Introduction to Mathematical Logic by Leary and Kristiansen
- Sipser Book: An Introduction to the Theory of Computation by Michael Sipser
- Wolf Book: A Tour through Mathematical Logic by Robert S. Wolf
- Decision Procedures Book: Decision Procedures: An Algorithmic Point of View by Daniel Kroening and Ofer Strichman
- Ganesh/Vardi book chapter: On the Unreasonable Effectiveness of SAT Solvers by Vijay Ganesh and Moshe Vardi
- Dennis Gurichev's Excellent Notes – SAT/SMT by Example by Dennis Gurichev

6 Project Requirements

All students who take the course for a grade have to complete the course project requirements. Students can choose to do a project that is primarily empirical (e.g., build a solver or apply solvers to solve an interesting and difficult problem) or theoretical (e.g., prove an interesting theorem related to solvers). A good project should lead to a publishable paper. Every project submission must be accompanied by a short 5 page preliminary proposal (to be submitted in the 3rd week of the term), a (max) 15 page report (end of term), and a 45 minute presentation (end of term). I will personally inspect your code (empirical projects) and/or proofs (theory projects).

The most effective way to succeed with regard to this requirement is to consult with me as often as you can (preferably once a week). I am more than happy to set up a time to chat via Zoom or in-person. If you are already conducting research in the area of solvers or their applications, then that project can be used to complete the course requirements. Having said that, you have to demonstrate that your work was conducted during this term (and not merely reuse previous work of yours).

It goes without saying that all Georgia Tech guidelines regarding plagiarism or cheating of any kind apply here as well. While you are free to reuse and extend previously written code, make sure to cite them appropriately. Also, it is preferable to use open source licenses that encourage reuse and dissemination of code (e.g., MIT license).

6.1 What Constitutes a Good Project?

Below find some details on what I am looking for in a high-quality project:

1. **Novelty:** I want you to research a specific sub-topic in the broad areas of SAT/SMT solvers and their applications in testing, analysis, verification, math, physics, and zero-in on a specific problem and method of interest to you. Then find out how you can add some novelty to previous work.

2. **Execution:** The quality of your code and how well you execute your project will be paramount in maximizing your marks. Similarly, the correctness/quality of your proofs/theorems would determine your grade in a significant way. Quality can be hard to define. However, I do have lots of experience evaluating research projects. I will apply my judgement on how good your project is and whether it is worthy of publication at a top-tier conference/journal.

3. **Potential for Impact:** How impactful can your project be? Does it open a new line of research? Does it solve a long-standing open problem?

6.2 Project Proposal, Report, and Slides

Below find the structure you should follow as you write your project proposal, report, and slides:

6.2.1 Project Proposal (5% of total marks)

You must present your ideas in a project proposal, arguing why the problem you have chosen is interesting, impactful, and your approach is novel. You must present a feasibility analysis, benchmarks used to test your approach, the suitability of these benchmarks (why these benchmarks and not something else), and an outline of your approach.

1. **Title and Abstract:** The proposal must have a descriptive project title, list the names/IDs/Email IDs of the group members (max. of 2), and a short abstract detailing the problem being addressed and a description of the proposed method.

2. **Problem Statement:** A short paragraph detailing the problem being addressed, why it is interesting, and what is the impact of solving said problem. (E.g., improvements to SAT/SMT solvers, a new program analysis tool based on solvers, a new application of solvers to math, test suite minimization, detecting security errors, detecting performance errors, adversarial attacks against ML systems, etc.)

3. **Approach or Algorithm Proposed:** A short description of the proposed method

4. **Framework or tool chain used:** Provide details of the system that you are modifying or using (e.g., LLVM version, Clang, KLEE)

5. **Benchmarks Used:** Describe the benchmarks you plan to use to test your system. I.e., list the programs on which you plan to run your testing or program analysis tool. Why are these benchmarks suitable to expose the strengths and weaknesses of your approach?

6. **Plan of Action and Feasibility Analysis:** Present a plan of action (i.e., deliverables with tentative dates) and why you think you can pull of this project in approx. 10 weeks. For example, you can argue that you have the necessary background and solid understanding of the tool ecosystem for your project. It is very important to think this through carefully.

7. **Explain your Demo:** You have to state your final goal in your proposal, e.g., you plan to run your testing tool on your benchmarks and expose certain class of errors. While having a benchmark is great way to guide your research project, we expect your tool to perform well outside of the benchmark you have created.

6.2.2 Project Report, Demo, and Slides (70% of the total marks)

At the end of the term, you will make a approx. 45 minute presentation of your project as well as submit a publishable paper complete with title, abstract, introduction (that includes a crisp problem statement and list of contributions), related work, your algorithm/theorems, example illustrating your algorithm, experimental evaluation on a meaningful benchmark for empirical projects, conclusions, and future work.

6.2.3 Structure of the Final Project Report

The final project report should be in standard font used in IEEE or ACM papers with a font size of 11 pt. Both 1-column and 2-column are acceptable. The total number of pages (excluding references) may not exceed 10 (2-column) or 15 (1-column). I urge you to use Latex to typeset your reports. The report should follow the structure given below:

1. Title

2. Author name, affiliations, ID

3. Abstract: crisp definition of problem and overview of solution (300 or so words)

4. Section: Introduction

   (a) Problem statement: briefly state and describe the problem you address in the paper

   (b) Context: describe some previous addressing the problem, the importance/relevance of the problem and also why it is non-trivial

(c) Overview of solution: provide an overview of the solution

5. Section: Background. Provide any theoretical or empirical background, if necessary
6. Section: Detailed solution description
   (a) Describe your algorithm(s), theorem(s), and/or implementation
   (b) Provide examples
7. Section: Experimental Evaluation/Results
   (a) Describe how you evaluated your method/algorithm
   (b) The setup (e.g., hardware/software/OS used)
   (c) The benchmark suite (testcases used to test your method)
   (d) Metrics used to measure the quality of your method against state-of-the-art
   (e) Actual result tables and graphs (e.g., cactus plots)
   (f) Make this section as detailed as possible
8. Section: Related Work
   (a) What tools are comparing against. Why these tools, and why not some tools (why did you choose a certain set of competing tools over others)?
   (b) How is your tool/method/algorithm better than the related work
9. Section: Conclusions and Future Work
   (a) Crisply restate your contributions and argue why it is an improvement over the state-of-the-art
   (b) Describe your next steps

6.2.4 Talk Slides

Aim for a 45 minute talk, with no more than 40 slides. Follow the final project report/paper structure closely.

6.3 Potential Projects

Below is a list of **classes of projects** that you can use as inspiration. Note that you still have to do research by reading the references below (and consulting with me) to narrow down the exact project you want to work on.

7 Potential Projects

Below is a list of **classes of projects** that you can use as inspiration. Note that you still have to do research by reading the references below (and consulting with me) to narrow down the exact project you want to work on.

1. **Machine learning based solver heuristics:** SAT and SMT solvers use a lot of heuristics for branching (variable and value selection), restarts, sequencing and selection of rules. These heuristics can be implemented using machine learning (ML) techniques. There is a wealth of information about these techniques in the book by Sean Holden [H$^+$21]. You can also refer to papers on MapleSAT (and its variants) [LGPC16, LOM$^+$18], AlphaMapleSAT [JLL$^+$24], and Z3-Alpha [LSJ$^+$24].

2. **Algorithm selection:** Another line of research that is closely related to the use of ML inside solvers is the use of ML for (adaptive) algorithm selection. This is a very rich space with many tools starting with the work of SATzilla [XHHLB08] to the recent Verification of Neural Networks (VNN) solver Goose [SPKG22].

3. **Proof complexity of solvers:** You may choose a theory project by diving into proof complexity of solvers [BN21]. For example, you could analyze a proof system such as resolution and study lower bounds to it. You may also want to do research on the restarts problem [BBJ14]. Another avenue could be characterizing class of instances that are easy for SAT solvers [GV20].

4. **Applications of solvers:** In my previous offerings of this course, applications of solvers were often the most sought-after project directions. A few directions include extending KLEE [CN21, CS13].

5. **Profiler and visualization for SMT solvers:** This project involves instrumenting SMT solver code to better understand its internal processes, collecting data, and visualizing the transformation of an SMT formula (viewed as a graph). Existing profilers for Z3 can be used as a starting point.

6. **Extended resolution SAT solver:** Implement an extended resolution SAT solver, a proof system stronger than resolution. Extended resolution allows the introduction of definitions using new variables.

7. **Parameterized CDCL verified in Coq/LEAN:** Implement a verified SAT solver in Coq or LEAN. This project requires understanding DPLL or CDCL SAT solvers in terms of invariants and proving adherence to them. Another option is to implement a SAT solver in LEAN and automatically generate a proof of correctness.

8. **Synthesizing Boolean expressions using CEGIS:** Automatically synthesize Boolean expressions using counterexample-guided inductive synthesis (CEGIS). CEGIS can be used to generate code from specifications. Explore if large language models (LLMs) can be used to generate Boolean expressions.

9. **String solvers:** Extend an existing string solver in Z3 to improve performance or enhance its capabilities.

10. **Fuzzers for solvers:** Develop a fuzzer to test SAT/SMT/VNN solvers, ensuring robustness and correctness under a variety of inputs.

11. **Extend KLEE for program repair or path explosion problem:** Implement program repair techniques using symbolic/concolic execution. Another option is to use static analysis to construct function summaries, aiding KLEE in addressing path explosion problems, or to guide symbolic analysis more effectively.

12. **Symmetry breaking techniques for SAT solvers:** Implement a symmetry breaking technique as a pre-processor or through the programmatic SAT interface.

8 Teaching Style

My teaching style is very interactive and I am very passionate about teaching. I am also flexible and work closely with my students to help them overcome challenges that they may face while taking my course. Finally, I believe that one of my primary roles is to enable my students to succeed in their careers. Consequently, I tailor the content of my courses to be aligned with their career aspirations to the extent possible.

9 Tentative Weekly Schedule

SAT/SMT Solver Course (Tentative) Weekly Schedule

Week	Topics	Book(s) chapters (Self-contained slides will be provided for all topics)
Week 1	Introduction to SAT/SMT solvers and their applications. Boolean logic, propositional proof systems – part I	Slides
Week 2	Boolean logic, propositional proof systems, soundness, completeness – part II	Slides
Week 3	Algorithms for the Boolean satisfiability problem – DPLL SAT Solvers	Slides
Week 4	Algorithms for the Boolean satisfiability problem – CDCL SAT Solvers. Use of solvers in symbolic execution-based testing and analysis	Slides
Week 5	First-order logic, first-order proof systems, soundness, completeness – part I	Slides
Week 6	First-order logic, first-order proof systems, soundness, completeness – part II	Slides
Week 7	MID-TERM WEEK	
Week 8	SMT Solvers – architecture, CDCL(T)	Slides
Week 9	Uninterpreted functions	Slides
Week 10	Combination of theories – Nelson-Oppen	Slides
Week 11	Decision procedures for theories over bit-vectors and arrays	Slides
Week 12	Decision procedures for integers and real linear arithmetic	Slides
Week 13	Decision procedures for theories over strings	Slides
Week 14	Decision procedures for theories over strings	Slides
Week 15	Applications of SAT/SMT solvers: symbolic execution and testing	
Week 16	More applications or other topics (e.g., verification of neural network solvers, LLMs)	
Week 17	FINALS WEEK	

References

[BBJ14] Maria Luisa Bonet, Sam Buss, and Jan Johannsen. Improved separations of regular resolution from clause learning proof systems. *J. Artif. Intell. Res.*, 49:669–703, 2014.

[BN21] Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 233–350. IOS Press, 2021.

[CN21] Cristian Cadar and Martin Nowack. KLEE symbolic execution engine in 2019. *Int. J. Softw. Tools Technol. Transf.*, 23(6):867–870, 2021.

[CS13] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: three decades later. *Commun. ACM*, 56(2):82–90, 2013.

[GV20] Vijay Ganesh and Moshe Y. Vardi. On the unreasonable effectiveness of SAT solvers. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 547–566. Cambridge University Press, 2020.

[H+21] Sean B Holden et al. Machine learning for automated theorem proving: Learning to solve sat and qsat. *Foundations and Trends® in Machine Learning*, 14(6):807–989, 2021.

[JLL+24] Piyush Jha, Zhengyu Li, Zhengyang Lu, Curtis Bright, and Vijay Ganesh. Alphamaplesat: An mcts-based cube-and-conquer sat solver for hard combinatorial problems. *arXiv preprint arXiv:2401.13770*, 2024.

[LGPC16] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19*, pages 123–140. Springer, 2016.

[LOM+18] Jia Hui Liang, Chanseok Oh, Minu Mathew, Ciza Thomas, Chunxiao Li, and Vijay Ganesh. Machine learning-based restart policy for CDCL SAT solvers. In *Theory and Applications of Satisfiability Testing–SAT 2018, Oxford, UK, July 9–12, 2018, Proceedings 21*, pages 94–110. Springer, 2018.

[LSJ+24] Zhengyang Lu, Stefan Siemer, Piyush Jha, Joel Day, Florin Manea, and Vijay Ganesh. Layered and staged monte carlo tree search for smt strategy synthesis. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*, 2024. Accepted.

[SPKG22] Joseph Scott, Guanting Pan, Elias B. Khalil, and Vijay Ganesh. Goose: A meta-solver for deep neural network verification. In David Déharbe and Antti E. J. Hyvärinen, editors, *Proceedings of the 20th Internal Workshop on Satisfiability Modulo Theories co-located with the 11th International Joint Conference on Automated Reasoning (IJCAR 2022) part of the 8th Federated Logic Conference (FLoC 2022), Haifa, Israel, August 11-12, 2022*, volume 3185 of *CEUR Workshop Proceedings*, pages 99–113. CEUR-WS.org, 2022.

[XHHLB08] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.