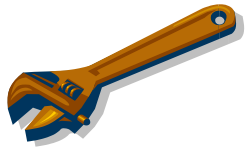
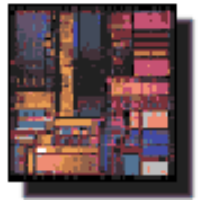






Digital Systems Design Automation

Course Project Overview

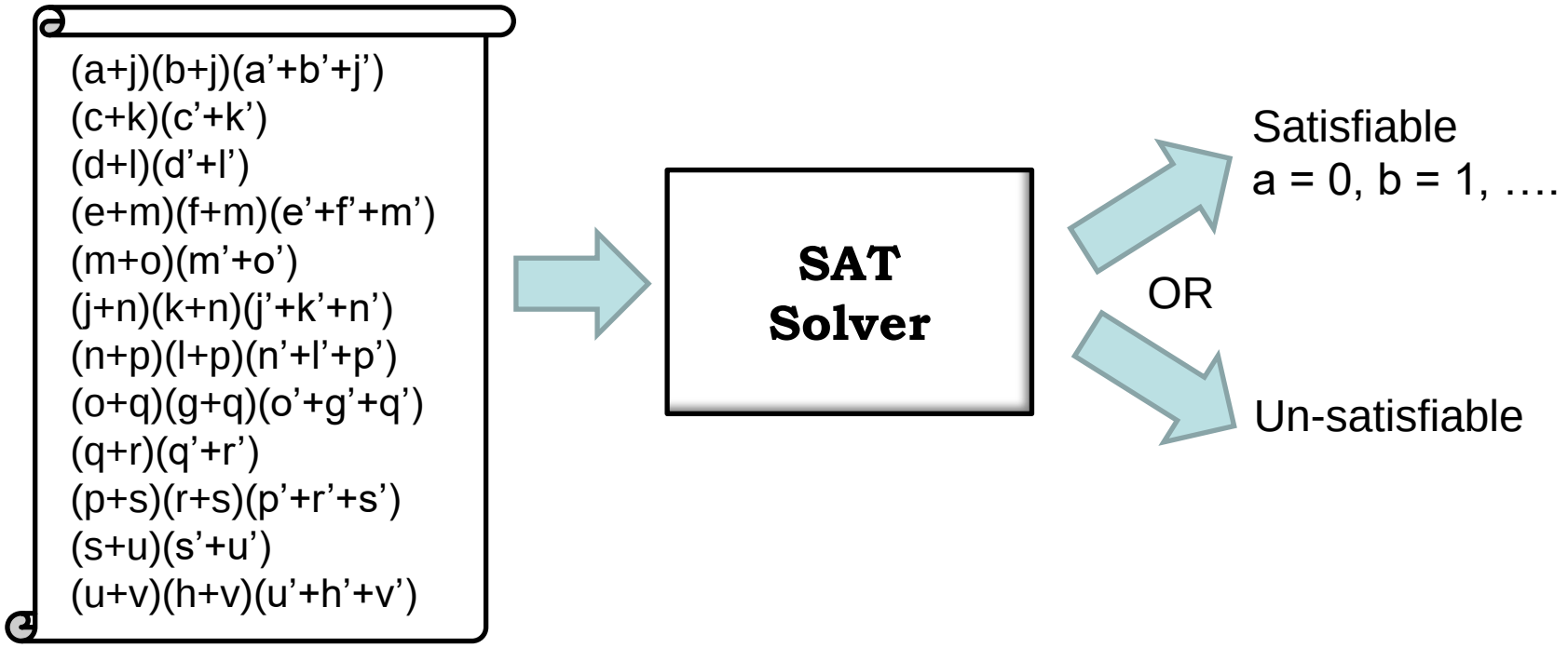


Anand Raghunathan

raghunathan@purdue.edu

Satisfiability: The Search for Truth

- The project will focus on **Boolean Satisfiability (SAT)**, a fundamental problem in EDA (and Computing)
- Given a Boolean formula in CNF form, determine whether there is an assignment of variables that makes the formula TRUE



Two Options

- **Option 1 (build):** Implement a SAT solver based on the DPLL backtracking algorithm
  - DPLL + at least two advanced heuristics
- **Option 2 (improve):** Propose and implement an improvement to an existing state-of-the-art SAT solver
  - Only in exceptional cases, requires discussion and agreement on project scope
- For most students, Option 1 will be the better choice since it is well-defined
  - Option 2 places greater onus on you to define the project scope
  - Teams pursuing option 2 need to schedule a discussion with the instructor to mutually agree on the project scope

Option 1: Implement a basic SAT solver

- **Input**: CNF formula in "DIMACS" format
  - Several public benchmark repositories (https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html, http://archive.dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf/)
- **Output**: Satisfying variable assignment, or assertion that the formula is un-satisfiable
- **Expected scope:** DPLL + at least two advanced heuristics (choose from conflict-driven learning, non-chronological backtracking, watched literals, decision heuristic such as DLIS or VSIDS)
- Why this option?
  - Learn how to implement a fundamental search algorithm and heuristics to improve its performance
  - Well-defined scope
- Why not?
  - Already know the basic algorithm, want to try something else
  - Do something more cutting-edge or closer to research
- Other questions
  - Is this a lot of code? Not necessarily – for example, MiniSat (http://minisat.se/), which was a winner in the SAT 2005 and 2007 competitions, started off as ~600 lines of code!
  - Can I re-use code? **Only for the input parser**, otherwise defeats the whole purpose of doing this

Option 2: Improve an existing state-of-the-art SAT solver

- **Starting point**: Existing public-domain SAT solver of your choice (MiniSat recommended)
- **Deliverable**: An improved version
  - Examples
    - Parallelize on multi-core or GPU or cluster
    - Implement new heuristic based on a publication
    - Optimize for specific classes of problem instances (e.g., easily satisfiable)
    - Forego completeness to gain speed (e.g., use Neural Nets)
    - **<Your idea goes here!>**
- Why this option?
  - See if you can improve the state-of-the-art
- Why not?
  - Greater onus on you to define the project
- **Critical to success: Clearly define and manage scope**
- Teams pursuing option 2 need to schedule a discussion with the instructor to mutually agree on the project scope

Team configurations

- On-campus students (both in-person and async online, i.e., sections ECE-51216-001 and ECE-51216-004)
  - Project teams **should consist of two students**
  - Single student teams will be allowed only in exceptional circumstances (with instructor approval)
- Remote students (section ECE-51216-EPE)
  - Project teams can be comprised of one or two students

- **Students are responsible for forming teams**
  - This spreadsheet tells you who's available: https://docs.google.com/spreadsheets/d/1tFH_3m26hWBvOC66gCVpopT1wZ-FtjaoaBj8nk2mEiQ/edit#gid=902967585
  - Once you form teams, update the spreadsheet
- All guidelines are the same for both single and two-student teams, but grading will be separate

Timeline

- March 5th, 2024: Project overview released
- March 5th – March 12th, 2024 : Form teams, decide on option
- March 12th, 2024: Preliminary project proposal due (no more than 2 pages)
  - Option 1 or 2
  - In case of option 1: Heuristics you plan to implement
  - In case of option 2: Motivation, existing baseline solver you will improve, how you will improve it
  - Evaluation & testing plan, Timeline & milestones
- April 7th, 2024: Intermediate report
  - Implementation status, Issues encountered and how you plan to address them
- May 4th, 2024: Final project submission
  - Code + input data sets + documentation
  - Project report
- For Option 2 teams: Project presentations (20 mins) will be scheduled during finals week (April 29th – May 4th)

I strongly recommend that you get started as soon as possible and have a clear plan (with timeline and milestones)

References (SAT)

- L. Zhang and S. Malik, “The Quest for Efficient Boolean Satisfiability Solvers”, CAV 2002
  - http://www.princeton.edu/~chaff/publication/cade_cav_2002.pdf
- “SAT and MAX-SAT for the Lay-Researcher”, http://www.mqasem.net/sat/sat/
- SAT introduction slides at UC Berkeley
  - https://people.eecs.berkeley.edu/~sseshia/219c/lectures/SATSolving.pdf

ECE 51216, Spring 2024
Digital Systems Design Automation

Project Submission Guidelines

Please read the guidelines and information in this document carefully – they will help you prepare your project report and code submission so as to receive highest credit.

DEADLINE

The project is due on Saturday, May 4th, 2024 (11:59PM EST). This is a hard deadline due to the cutoff time to submit grades to the registrar's office.

1. WHAT TO SUBMIT

You will need to submit the following:

- A project report that clearly documents the motivation/rationale, algorithms and data structures chosen, experimental methodology, results obtained, and an analysis of the results and conclusions. The requirements for the project report will vary based on which option you have chosen (Option 1- build or Option 2 – improve). Please read the guidelines provided in Section 2.
- For project teams pursuing Option 1 (build) or Option 2 (improve), a self-contained code package (.zip file) containing the code that you have implemented for the SAT solver, makefiles/compilation scripts, and benchmarks used for evaluation. The code must follow the guidelines specified in Sections 3/4. ***We cannot evaluate (and therefore cannot give credit for) code that does not compile or run, so please make sure that you provide a submission that is self-contained and works on the specified machines***.

All code and project report submissions must be made on Brightspace (note that Brightspace allows you to include multiple attachments with your submission, so use one for the report and one for the code).

2. PROJECT REPORT

You are free to choose your own writing style for the report, provided that it includes at least the following information. Feel free to add additional material that you think is relevant.

- **Option 1 (Build)**: In this case, your report should (i) provide a pseudo-code description of the basic algorithm (DPLL) and all the major heuristics implemented (BCP, conflict clause learning, non-chronological backtracking, *etc.*), (ii) explain the data structures used and justify your choices, (iii) describe the experimental methodology, benchmarks used to evaluate your implementation, and the results obtained, (iv) analyze the results (*e.g.*, how does the run-time of your implementation vary with problem size, what is the effect of

turning on/off each of the heuristics, *etc.*), and (v) summarize the significant conclusions that can be drawn from your results.

- **Option 2 (Improve)**: For this case, your report should (i) introduce and motivate the advanced heuristic that you chose to implement (why did you think the chosen technique would be useful, and under what scenarios), (ii) describe at a conceptual level (*e.g.*, pseudo-code) the changes you made to the existing SAT solver, (iii) describe the experimental methodology (which SAT solver was used, benchmarks used, and platform used) and the results obtained, (iv) analyze the results and discuss the impact of the implemented technique (when / by how much does the performance of the original SAT solver improve, and why), and (v) summarize any significant conclusions that can be drawn from your results, including how well the improvement works and when it works or does not work best.

You are strongly encouraged to use figures, graphs, and tables whenever appropriate to support your explanation. For example, consider drawing figures that illustrate your data structures, and presenting your results in graphs and/or tables.

3. INTERFACE FOR YOUR SAT SOLVER

Your SAT solver **must adhere to** one of two options for the command line interface (depending on whether it is compiled into an executable or implemented as a Python script):

```
mySAT benchmark.cnf
```

where "mySAT" is the name of the executable, and "benchmark.cnf" is the name of the CNF benchmark.

OR

```
python3 mySAT.py benchmark.cnf
```

where "mySAT.py" is the name of the main file, and "benchmark.cnf" is the name of the CNF benchmark that it will accept as input.

Your SAT solver **MUST print out the following information to standard output (stdout)** if it finds the CNF formula to be satisfiable:

```
RESULT: SAT
ASSIGNMENT: a=1 b=0 c=1 …
```

The first line simply says the formula is satisfiable. The second line is the assignment of values to the variables that makes the CNF formula TRUE (replace a, b, c, … by the actual names of the variables in the CNF formula). **Be sure to follow this exact format. Failure to do this may cause you to not receive proper credit since the grading script may not recognize your output.**

Your SAT solver MUST print out the following information it finds the CNF formula to be unsatisfiable:

```
RESULT: UNSAT
```

Be sure to follow this exact format. Failure to do this may cause you to not receive proper credit since the grading script may not recognize your output.

Your program should produce no additional output than specified above (any debugging or other information should be disabled in your submission).

4. FINAL CODE SUBMISSION

- Please submit your code on Brightspace as a single file named GroupNumber_{Language}.zip
    - The group number will be assigned to you once the teams are formed.
    - Language refers to the programming language your solver is implemented in (C, C++, Java, Python)
- Only one submission per team (any one of the group members can submit)
- ***The submission must be self-contained***, *i.e.*, it should include all packages used other than standard tools like the compiler, standard libraries and utilities already installed on most UNIX systems. In particular, the submission must include **all source code, benchmarks used in your testing, and makefiles or compilation scripts**.
- We will test (compile and run) all code submissions on the ECECOMP cluster (ececomp.ecn.purdue.edu). Your code must compile and run on these machines.
    - Your submission must include both a pre-compiled executable (if using a compiled programming language) as well as instructions to compile the code
    - **Under special circumstances (Option 2 only)**: We can run your code on other ECN machines provided you clearly specify which one. For example, if you need a platform with a larger number of processors/cores or a machine with a GPU. **In this case, you must inform the instructor and TA at least 4 weeks prior to the final submission deadline so that we can ensure we have a suitable machine.**
- **Documentation**: **The submission must contain a README file** explaining (i) the directory structure and what code is contained in each source/header file, (ii) how to compile your code, and (iii) how to run your code (you must adhere to the interface described in Section 3). You must document important data structures and functions in your source code. Please note that 10% of the project score will be allocated for documentation.
- All benchmarks that you used to test the code must be included in your submission. In addition, we will execute your code on additional benchmarks.

4. GRADING

All project submissions will be tested on a common set of benchmarks, and graded on a scale of 100, which will be broken down as given below.

For Option 1 (build), the breakdown of credit will be as follows:

- 50% for functionality and correct implementation – the algorithm and heuristics implemented, and whether the code compiles and runs correctly on the specified platform.

- 25% for performance – how efficient your implementation is in run time and memory usage, how these requirements scale with problem size/difficulty.
- 25% for project report, clear source code documentation and README file.

For Option 2 (improve), the breakdown of credit will be as follows:

- 50% for functionality and correct implementation – the improvement methods, and whether the code compiles and runs correctly on the specified platform.
- 25% for performance – improvements achieved over the baseline in run time and memory usage, how these requirements scale with problem size/difficulty.
- 25% for project report, code documentation and README file

GOOD LUCK!