SMT Solvers and Their Applications

Aaryan Vaishya
Mentor: Brynmor Chapman

October 19, 2025
MIT PRIMES Conference

SAT

Definition

The boolean satisfiability (SAT) problem asks if a boolean formula (using ANDs, ORs, and NOTs) has an assignment of its variables that evaluates to TRUE.

- Example: x_1 AND x_2 is satisfiable, by letting both variables be true.
- Example: (x_1 AND (NOT x_1)) AND x_2 is not satisfiable, as x_1 AND (NOT x_1) is always false.
- SAT is an NP problem; there is no known solution in polynomial time, but any solution can be verified in polynomial time.
- Fastest known solutions are exponential.
- SAT is the prototypical NP-complete problem; meaning every problem in NP can be converted to SAT in polynomial time.
- Yet, heuristic solvers are extremely fast for most practical instances of SAT.

How to Solve SAT

It is desirable to convert formulas into a standardized format that is easy to work with.

Conjunctive Normal Form

A formula is in **Conjunctive Normal Form (CNF)** if it is a conjunction (AND) of n **clauses**, where the i^{th} is a disjunction (OR) of a_i **literals** $\ell_{i,j}$ (variables or their negations).

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{a_i} \ell_{i,j}$$

A CNF is satisfied if and only if *every* clause is true.

Exercise (trivial)

Show that every boolean formula has a representation as a CNF, in the sense that the CNF is equivalent to the original formula for all inputs.

How to Solve SAT

Tseitin's transformation allows us to get a CNF A whose size is linear in the size of the original expression B such that the A is satisfiable iff B is satisfiable. In particular, A and B are NOT the same for all inputs.

Tseitin's Transformation

Introduce a variable for the result of each operator, starting from the deepest nested clauses. Then there exist three clauses whose conjunction forces satisfiability to hold only if and only if the new variable actually represents the result of its designated operator. The CNF is the variable assigned to the final operator conjuncted with the aforementioned clauses.

Transformation Rules

Each operator is replaced by a new variable and encoded as follows:

$$a \leftrightarrow (b \wedge c): \quad (\neg a \vee b) \ \wedge \ (\neg a \vee c) \ \wedge \ (a \vee \neg b \vee \neg c)$$

$$a \leftrightarrow (b \vee c): \quad (\neg b \vee a) \ \wedge \ (\neg c \vee a) \ \wedge \ (\neg a \vee b \vee c)$$

How to Solve SAT

Tseitin's Transformation: Example

Transform $\varphi = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$

Introduce Variables

$$y_1 \leftrightarrow (x_1 \wedge x_2), \quad y_2 \leftrightarrow (\neg x_1 \wedge x_3), \quad y_3 \leftrightarrow (y_1 \vee y_2)$$

Encode Each Equivalence In Clauses

$$y_1 \leftrightarrow (x_1 \wedge x_2) \Rightarrow (\neg y_1 \vee x_1),\ (\neg y_1 \vee x_2),\ (\neg x_1 \vee \neg x_2 \vee y_1)$$

$$y_2 \leftrightarrow (\neg x_1 \wedge x_3) \Rightarrow (\neg y_2 \vee \neg x_1),\ (\neg y_2 \vee x_3),\ (x_1 \vee \neg x_3 \vee y_2)$$

$$y_3 \leftrightarrow (y_1 \vee y_2) \Rightarrow (\neg y_1 \vee y_3),\ (\neg y_2 \vee y_3),\ (\neg y_3 \vee y_1 \vee y_2)$$

All clauses above together with the unit clause (y_3) form a CNF formula equisatisfiable with the original φ.

How to Solve SAT

Davis–Putnam–Logemann–Loveland (DPLL) Algorithm

The DPLL procedure explores a **search tree of all possible variable assignments**. Each level of the tree corresponds to assigning one variable.

Algorithm Sketch

1. **Decision:** Choose a variable (a **decision variable**) and assign it a value. This creates two branches in the search tree: one for `True`, one for `False`.
2. **Unit Propagation:** If any clause has only one unassigned literal with all assigned literals false, that literal must be true.
3. **Check for Contradiction:** A **contradiction** occurs when all the literals of a clause are assigned to be false.
4. **Backtracking:** When a contradiction occurs, the solver **backtracks** to the most recent decision point, flips that variable's value, and resumes exploration from there.

How to Solve SAT

Conflict-Driven Clause Learning (CDCL)

CDCL extends DPLL by learning from conflicts instead of re-exploring the same mistakes. Whenever a clause becomes unsatisfied (a **conflict**), the solver analyzes which assignments caused it and derives a new **learned clause** that blocks the same combination in the future.

Conflict Handling

- The solver traces implications in a graph to find the root cause of the conflict.
- A new clause is learned by resolution.
- The solver then **backjumps non-chronologically:**
  - Instead of undoing one decision at a time, it jumps directly to the most recent earlier decision level that is actually relevant to the conflict.
  - This skips over many unnecessary branches in the search tree.

How to Solve SAT

Practical Techniques That Power SAT Solvers

- **Branching heuristics:** Pick the variables that keep showing up in conflicts as decision variables.
- **Restarts:** Periodically restart the search to avoid deep unproductive branches, keeping learned clauses.
- **Clause deletion:** Remove stale or weak learned clauses based on activity or LBD (literal block distance - how far apart in the tree literals in a clause are assigned) metrics.
- **Phase saving:** Remember the last successful polarity (true/false state) for each variable.
- **Preprocessing:** Simplify CNF before solving (e.g., subsumption, variable elimination).

These heuristics and optimizations make modern solvers like `MiniSAT`, `Glucose`, and `Kissat` able to solve industrial-scale problems with millions of variables in seconds.

Mathematics with SAT

Boolean Pythagorean Triples Problem

Question: Can the natural numbers up to n be colored red/blue so that no monochromatic triple (a, b, c) satisfies $a^2 + b^2 = c^2$?

Encoding: Each number becomes a Boolean variable, and each Pythagorean triple adds clauses forbidding all three from sharing one color.

Result: SAT solvers proved this is impossible for $n > 7824$.

Schur Number Five

Question: What is the largest n such that $\{1 \dots n\}$ can be 5-colored without a monochromatic solution to $a + b = c$?

Encoding: Boolean variables represent number–color assignments, with clauses ensuring each number has one color and forbidding same-color additive triples.

Result: SAT solvers proved $n = 160$; any 5-coloring of $\{1 \dots 161\}$ fails.

From SAT to SMT

Limitations of SAT

- SAT reasoning is limited to Boolean variables and propositional logic.
- It cannot directly represent constraints involving numbers, arrays, strings, or structured data.
- Extending SAT to such domains requires bit-level encodings, which are inefficient and lose structural information.

Satisfiability Modulo Theories (SMT) extends SAT to handle richer, non-Boolean domains by incorporating mathematical **theories**.

Theories

Theories

- A **theory** defines a **logical domain** (universe of objects), a set of functions and relations, and axioms that describe how they behave.
- SMT solvers use such theories to reason about structured domains beyond propositional logic.

Common Theories

- **Linear arithmetic:** reasoning over numerical domains using addition and inequalities.
- **Arrays:** reasoning about indexed data through read and write operations.
- **Bitvectors:** reasoning about fixed-width binary values and low-level arithmetic.

Theory Solvers

Role

Theory solvers are at the core of SMT solvers. They check the satisfiability of conjunctions of basic constraints within a theory.

Reasoning Process

A theory solver maintains a model of its domain and refines it as new constraints arrive:

1. **Normalize** : rewrite constraints into canonical internal form.
2. **Propagate** : derive additional constraints implied by the current set.
3. **Check consistency** : test if all constraints can hold simultaneously.
4. **Explain conflict / construct model** : report contradictions or return a satisfying assignment.

Our Experience With SMT Solvers: Setup

Problem Statement

We seek to **maximize the number of edges** of the n-dimensional hypercube $\{-1, 1\}^n$ intersected an odd number of times by a polynomial curve p of degree k.

Signing Representation

- Each vertex $\mathbf{v} \in \{-1, 1\}^n$ is assigned the sign of the polynomial p evaluated at v
- An edge $(\mathbf{u}, \mathbf{v})$ is **intersected** if and only if $p(\mathbf{u})$ and $p(\mathbf{v})$ have opposite signs.
- A signing of the cube determines a unique configuration of intersected edges.

Our Experience with SMT Solvers: Method

Combinatorial Pruning

- Candidate signings are generated vertex by vertex and pruned using known weaker bounds on intersection-maximizing subcubes.
- Any intersection-maximizing subcube with too few edges implies that the entire hypercube will not have enough edges to beat known examples.
- Any subcube with too many intersections cannot give a working polynomial of degree k. These follow from smaller cases of the given problem.

Our Experience with SMT Solvers: Method

SMT Verification

- For each partial signing not yet eliminated, an SMT Solver (Z3) checks whether there exist coefficients of a degree k polynomial actually giving the desired signing (this is just solving a system of linear inequalities).
- Signings passing this test are then given another bit.

This technique has helped us correct previous bounds in the case where $n = 6, k = 3$.

Theorem (**V.** 2025)

The Gotsman-Linial conjecture for $(n, d) = (6, 3)$ is false. There exist cubics that intersect more than 150 edges of the 6 dimensional hypercube an odd number of times.

Acknowledgements

- I thank my mentor Brynmor Chapman for his guidance and support in our research.
- I thank the MIT PRIMES Program for this amazing opportunity to conduct research.
- I thank the organizers of this PRIMES conference for making it possible for us to present these results.

Thank You

Questions?