Formal Methods in Software Engineering

Boolean Satisfiability — Spring 2025

Konstantin Chukharev

§1 Boolean Satisfiability

Boolean Satisfiability Problem (SAT)

SAT is the classical NP-complete problem of determining whether a given Boolean formula is *satisfiable*, that is, whether there exists an assignment of truth values to its variables that makes the formula true.

$$\exists X . f(X) = 1$$

SAT is a *decision* problem, which means that the answer is either "yes" or "no". However, in practice, we are mainly interested in *finding* the actual satisfying assignment if it exists — this is a *functional* SAT problem.

Historically, SAT was the first problem proven to be NP-complete, independently by Stephen Cook [1] and Leonid Levin [2] in 1971.

Stephen Cook

Leonid Levin

Cook–Levin Theorem

Theorem 1 (Cook–Levin): SAT is NP-complete.

That is, SAT is in NP, and *any* problem in NP can be *reduced* to SAT in polynomial time.

The proof is due to Richard Karp [3], who introduced the concept of *polynomial-time many-one reductions*, also known as *Karp reductions*. The earlier proof by Cook was based on a weaker type of reduction called *Turing reduction* or *Cook reduction*.

Definition 1 (Karp's many-one reduction): A polynomial-time *many-one reduction* from a problem A to a problem B is a polynomial-time computable function f such that for every instance x of A, x is a "yes" instance of A if and only if $f(x)$ is a "yes" instance of B. A reduction of this kind is denoted as $A \leq_p B$ and called a *polynomial transformation* or *Karp reduction*.

Cook–Levin Theorem [2]

Proof sketch: A problem L is in NP if there exists a polynomial-time verifier (Turing machine) $V(x, c)$ that verifies whether a certificate c is a valid proof that $x \in L$.

A *Karp reduction* from L to SAT is a polynomial-time computable function f mapping instances x of L to propositional formulas φ_x, such that $x \in L$ iff φ_x is satisfiable.

For input x, simulate $V(x, c)$ computation (with certificate c) as a *Turing machine* run. Encode its execution over $T = \mathcal{O}(p(|x|))$ steps into a propositional formula φ_x:

- Variables represent the machine's state, tape cells, and head position at each step t.
- Clauses enforce the initial configuration (input x and empty certificate c), valid transitions between steps (per V's rules), and the acceptance at step T.

A satisfying assignment to φ_x corresponds to a valid certificate c causing $V(x, c)$ to accept.

The encoding $x \mapsto \varphi_x$ is computable in polynomial time. Since $L \in \text{NP}$ was arbitrary, *all NP problems can be reduced to SAT*, proving SAT is **NP-hard**. As SAT is also in **NP**, it is **NP-complete**. $\square$

<u>This foundational result shows that SAT is a "universal" problem for NP.</u>

Solving General Search Problems with SAT

Modelling and solving general search problems:

1. Define a finite set of possible *states*.
2. Describe states using propositional *variables*.
3. Describe *legal* and *illegal* states using propositional *formulas*.
4. Construct a propositional *formula* describing the desired state.
5. Translate the formula into an *equisatisfiable* CNF formula.
6. If the formula is *satisfiable*, the satisfying assignment corresponds to the desired state.
7. If the formula is *unsatisfiable*, the desired state does not exist.

Example: Graph Coloring

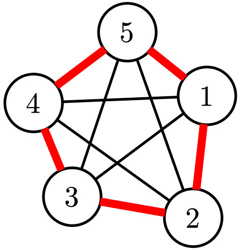
Recall that a graph $G = (V, E)$ consists of a set V of vertices and a set E of edges, where each edge is an unordered pair of vertices.

A complete graph on n vertices, denoted K_n, is a graph with $|V| = n$ such that E contains all possible pairs of vertices. In total, K_n has $\frac{n(n-1)}{2}$ edges.

Given a graph, color its vertices such that no two adjacent vertices have the same color.

Given a complete graph K_n, color its edges using k colors without creating a monochromatic triangle. What is the largest complete graph for which this is possible for a given number of colors?

- For $k = 1$, the answer is $n = 2$.
  - The graph K_2 has only one edge, which can be colored with a single color.
- For $k = 2$, the answer is $n = 5$.
  - See the example of 2-colored K_5 on the right.
- For $k = 3$, the answer is $n = 16$.
  - This is the work for a SAT solver. See the next slides.



Modelling and Solving the Graph Coloring Example

1. *Define a finite set of possible states.*
   - Each possible edge coloring is a state. There are $3^{|E|}$ possible states.
2. *Describe states using propositional variables.*
   - A simple (*one-hot*, or *direct*) encoding uses three variables for each edge: e_1, e_2, and e_3. There are 8 possible combinations of values of three variables, which given a state space of $8^{|E|}$. This is larger than necessary, but keeps the encoding simple.
3. *Describe legal and illegal states using propositional formulas.*
   - For each edge $e \in E$, the formula $e_1 + e_2 + e_3 = 1$ (so called "cardinality constraint") ensures that each edge is colored with exactly one color. This reduces the state space to $3^{|E|}$.
4. *Construct a propositional formula describing the desired state.*
   - The desired state is one in which there are no monochromatic triangles. For each triangle (e, f, g), we explicitly forbid it from being colored with the same color:

$$\neg((e_1 \leftrightarrow f_1) \wedge (f_1 \leftrightarrow g_1) \wedge (e_2 \leftrightarrow f_2) \wedge (f_2 \leftrightarrow g_2) \wedge (e_3 \leftrightarrow f_3) \wedge (f_3 \leftrightarrow g_3))$$

Modelling and Solving the Graph Coloring Example [2]

5. *Translate the formula into an equisatisfiable CNF formula.*
   - This can be done using the Tseitin transformations.

6. *If the formula is satisfiable, the satisfying assignment corresponds to the desired state.*
   - The satisfying assignment corresponds to a valid edge coloring. Among variables e_1, e_2, and e_3, the single one with the value of 1 corresponds to the color of the edge.

7. *If the formula is unsatisfiable, the desired state does not exist.*
   - If the formula is unsatisfiable, there is no valid edge coloring.

Now, run a SAT solver for increasing values of n, and find the largest n for which the formula is satisfiable. The answer is $n = 16$ for $k = 3$.

TODO

- [ ] Encodings
- [ ] SAT Solvers
- [ ] Applications
- [ ] Exercises

Bibliography

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 1971, pp. 151–158. doi: 10.1145/800157.805047.

[2] L. A. Levin, "Universal sequential search problems," *Problemy Peredachi Informatsii*, vol. 9, no. 3, pp. 115–116, 1973, [Online]. Available: http://mi.mathnet.ru/ppi914

[3] R. M. Karp, "Reducibility among Combinatorial Problems," *Complexity of Computer Computations*. Springer, pp. 85–103, 1972. doi: 10.1007/978-1-4684-2001-2_9.