1. Introduction

So far the project has been about learning the language of rigorous constraint. Before we can ask a computer to solve a Sudoku puzzle, we have to teach it how to understand the rules.

2. The Core Concept: What Satisfiability Really Means

The most significant takeaway from our study is the concept of Satisfiability (SAT). Conceptually, Satisfiability is the study of possibility.

When we write down a set of logical rules (formulas), Satisfiability asks a fundamental question: "Is there a world where all these rules can be true at the same time?"

- **Satisfiability vs. Validity:** We learned to distinguish between something being possible versus something being universal. A formula is **Satisfiable** if there is at least one specific scenario (an assignment of variables) where it evaluates to True. In contrast, a formula is **Valid** (a tautology) only if it is True in every possible scenario.
- **The "Model":** In logic, finding a "model" is the same as finding a solution. For our upcoming project, a solved Sudoku board is simply a "model" that satisfies all the constraints we encoded. If a set of rules is "Unsatisfiable," it means the puzzle is broken—the constraints contradict each other, and no solution exists.

3. Speaking the Language: Syntax and Semantics

To reach that understanding, we first had to master the building blocks:

- **Syntax (The Grammar):** We learned how to construct well-formed formulas using variables and connectives like AND, OR, NOT, and IMPLIES. We realized that converting English constraints into these symbols is an art form in itself.
- **Semantics (The Meaning):** We explored how truth values propagate through a formula. We also touched on **Entailment**—the idea that if we know certain premises are true, logical laws guarantee that certain conclusions must also be true.

4. Standardizing the Problem: Normal Forms

We learned that computers struggle with the messiness of human-readable logic. To build a solver, we must standardize our inputs using Normal Forms.

- **CNF (Conjunctive Normal Form):** This is the standard format for SAT solvers. It structures the entire problem as a series of requirements (clauses) that must all be true (AND), where each requirement is a set of options (OR).
- **The Trade-off:** Converting a formula to CNF can sometimes cause the file size to explode efficiently. We studied **Tseitin Encoding**, a clever technique that introduces new temporary variables. This allows us to convert complex formulas into CNF without the file size blowing up, keeping the problem solvable.

5. How the Machine Thinks: DPLL and Resolution

Finally, we moved from representation to action. How do we actually find the solution?

- **Resolution:** We looked at the mathematical rule of Resolution. It's a way of simplifying logic by canceling out opposites. If one rule says "A or B" and another says "Not A or C," we can logically conclude "B or C."
- **The DPLL Algorithm:** This is the engine we will build next. We learned that brute-force guessing is too slow. DPLL is a "smart" search.
    1. It makes a decision (guesses a variable).
    2. It uses **Unit Propagation** (if a rule has only one option left, the computer is forced to take it).
    3. If it hits a contradiction, it **Backtracks** and tries a different path.
- **Clause Learning:** We also touched on how modern solvers "learn" from their mistakes. When they hit a dead end, they remember exactly what combination of variables caused the conflict so they never waste time testing that combination again.

6. Conclusion and Next Steps

We have successfully bridged the gap between abstract logic and algorithmic thinking. We understand not just how to write constraints, but why solvers work the way they do. Using this, we can implement the required Sudoku solver.