Brief Recall: 3SAT, 2SAT, P, and NP

The **3SAT problem** is a classic Boolean satisfiability problem where each clause has exactly three literals; it is known to be **NP-complete**, meaning it is as hard as the hardest problems in NP and no polynomial-time algorithm is known for it. In contrast, **2SAT**, where each clause has only two literals, is solvable in polynomial time and thus belongs to the class **P**. This distinction illustrates the fundamental complexity difference between 2SAT and 3SAT, highlighting key concepts in computational complexity theory: **P** (problems solvable efficiently) versus **NP** (problems verifiable efficiently), with 3SAT being a canonical NP-complete problem.

Truth Table for a 3SAT Clause

CLAUSE (l_1, l_2, l_3) = l_1 ∨ l_2 ∨ l_3

l_1	**l_2**	**l_3**	**CLAUSE**
F	F	F	F
F	F	T	T
F	T	F	T
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

If there are NOTs in the clause, it doesn't affect the reasoning, because we could just replace the literal in question by a new variable l_4 such that l_4 = ¬l_x, for example.

Proposed Transformation from a 3SAT Clause to a 2SAT Clause

The result of this transformation is **not logically equivalent** to the original clause, as the truth table below shows.

What does this transformation do?
It replaces the disjunction of the first two literals with a new variable `a` and enforces `($l_1$ ∨ $l_2$) ⇒ a`. The implication in the opposite direction seems to require a ternary clause.

Transformer Definition

Transformer(l_1,l_2,l_3,a) = (¬l_1∨a) ∧ (¬l_2∨a) ∧ (a∨l_3)

l_1	l_2	l_3	a	Transformer
F	F	F	F	F
F	F	F	T	T
F	F	T	F	T
F	F	T	T	T
F	T	F	F	F
F	T	F	T	T
F	T	T	F	F
F	T	T	T	T
T	F	F	F	F
T	F	F	T	T
T	F	T	F	F
T	F	T	T	T
T	T	F	F	F
T	T	F	T	T
T	T	T	F	F
T	T	T	T	T

By analyzing the truth table (after omitting the extra variable a), we observe:

- **Green** rows: results match the original 3SAT clause.
- **Blue** rows: irrelevant cases, as they would be unsatisfiable in any case — we can ignore them.
- **Red** and **Yellow** rows: these are problematic — either **false** (red) or **true for the wrong reasons** (yellow).

I considered using a second transformer and taking the intersection of results, but that would probably introduce **exponential complexity** depending on the number of clauses.

These problematic cases stem from the approximation in the transformation — specifically, the situation where $l_1 = l_2 = FALSE$ and $a = l_1 \lor l_2 = TRUE$. Since I couldn't eliminate this elegantly, I decided to remove it manually.

Algorithm

Let **TWO_SAT** be a program that solves 2SAT in polynomial time.

1. Receive a list of 3-literal clauses (3SAT).

2. Apply the **Transformer** to each clause.
   The complexity of this operation should be linear with respect to the number of clauses, i.e., $O(m)$ or something similar.
   When adding each new variable, record the corresponding clause index so we can remove superfluous options later. Let's imagine a list of dictionaries:

```
conditions = [
    {'l': 0, 'm': 1, 'n': 1, 'o': 0},
    {'a': 1, 'b': 0, 'c': 1, 'd': 0},
    {'x': 1, 'y': 1, 'z': 0, 'w': 1},
]
```

3. Apply `TWO_SAT` to the transformed clauses and collect satisfying assignments.
   - If there are no satisfying assignments, the original 3SAT formula is also unsatisfiable, because this transformation produces *more* models than the original.

4. **Filter the satisfying assignments**:
   - Using the clause-variable indices, check for each satisfying assignment whether the exception $\{l_1 = F, l_2 = F, a = TRUE\}$ is present. If yes, discard the assignment; otherwise, keep it.
   - Remove the added variables.

```
def condition_matches(entry, condition):
------return all(entry.get(var) == val for var, val in condition.items())

for assignment in the assignments_list:
------matching_conditions = [cond for cond in conditions if condition_matches(entry, cond)]
------
------if matching_conditions:
------------delete(assignment)
```

5. Return the final valid assignments.
   If no valid assignments remain after filtering, then the original 3SAT formula is unsatisfiable.

Conclusion:
Since the added operations are only linear or polynomial at worst, the complexity of solving 3SAT depends only on the complexity of solving 2SAT.

We can hence conclude that P=NP.