**Author:** [Amr Yasser]

# Table of Contents



---

# 1. Introduction

Truth-teller and liar puzzles are classic logic problems in which each participant makes statements that may be true or false. The goal is to determine which participants are truthful and which are lying, given a set of logical constraints. This report documents the automated solution for a six-person puzzle using Python, detailing the approach, results, and insights.

# 2. Problem Statement

Six individuals (A, B, C, D, E, F) each make a statement about themselves or others. A truthful individual always tells the truth, while a liar always lies. The specific logical constraints encoded in the project are:

1. **A → ¬B:** If A is truthful, then B is lying.
2. **B → (C ∧ D):** If B is truthful, then both C and D are truthful.
3. **C → (A → ¬E):** If C is truthful and A is truthful, then E is lying.
4. **D → (A ↔ B):** If D is truthful, then A and B share the same truth-value.
5. **E → (at least two liars among {A, B, C, D, F}):** If E is truthful, there are at least two liars among the other five.
6. **F → (D ∧ ¬E):** If F is truthful, then D is truthful and E is lying.

Additionally, the solution requires at least one truth-teller and at least one liar among the six individuals.

# 3. Methodology

The solution was implemented in Python using the following steps:

- **Representation:** Each individual's truth-value is represented as a Boolean ( `True` for truthful, `False` for liar).
- **Condition Checking:** A function `satisfies_conditions(A, B, C, D, E, F)` evaluates all six logical constraints plus the requirement of at least one truth-teller and one liar.
- **Enumeration:** The `itertools.product` utility generates all `2^6 = 64` possible truth-value assignments. Each assignment is tested against the constraints function.
- **Testing:** Two predefined cases verify correct behavior:
  - Case 1: (A=True, B=False, C=True, D=True, E=False, F=True) → **Valid**
  - Case 2: (A=False, B=True, C=False, D=False, E=True, F=False) → **Invalid**

# 4. Results

## 4.1 Predefined Test Cases

| Case | A | B | C | D | E | F | Outcome |
|---|---|---|---|---|---|---|---|
| 1 | True | False | True | True | False | True | Valid |
| 2 | False | True | False | False | True | False | Invalid |

## 4.2 Enumerated Valid Configurations

A total of **12 valid configurations** were found. Each row lists the truth-values for (A, B, C, D, E, F).

| # | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | True | False | True | False | False | False |
| 2 | True | False | False | False | True | False |
| 3 | True | False | False | False | False | False |
| 4 | False | False | True | True | True | False |
| 5 | False | False | True | True | False | True |
| 6 | False | False | True | True | False | False |
| 7 | False | False | True | False | True | False |
| 8 | False | False | True | False | False | False |
| 9 | False | False | False | True | True | False |
| 10 | False | False | False | True | False | True |

| # | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 11 | False | False | False | True | False | False |
| 12 | False | False | False | False | True | False |

# 5. Discussion

- **B is always a liar** in every valid configuration, indicating that B's statement cannot be truthful under the puzzle's conditions.
- **Configurations cluster** by the truth-values of C and D. When both C and D are truthful, multiple subcases arise based on E and F.
- **Role of E and F:** E's requirement of at least two liars strongly influences which combinations of others are admissible; F's constraint further restricts by linking D and E.

# 6. Conclusion

The automated approach successfully identified all valid truth-value assignments for the six-person logic puzzle. The exhaustive search confirms that 12 distinct solutions satisfy the encoded constraints, offering insight into the interplay of each participant's statements.

# 7. Recommendations

- **Extend the framework** to handle dynamic puzzles by parsing constraints from a configuration file.
- **Optimize performance** using constraint-satisfaction libraries (e.g., `python-constraint` or `z3`) for larger puzzles.
- **Enhance reporting** by visualizing solution clusters and dependency graphs of statements.

---