**Project Assignment: Formal Methods in Software Engineering**

**Title: Traffic Light Controller**

**Student Information**

* **Full Name:** Muhammad jameel
* **Registration No:** 2021-GU-0832
* **Roll No:** 30

**Ghazi University, Dera Ghazi Khan**

**Overview of the Problem**

The goal is to model and verify a traffic light controller system that validates state transitions. The system must enforce the following requirements:

**Requirements:**

* Sequential Transition: The light must follow the order: Red → Green → Yellow → Red.
* No Skipped States: Direct transitions from Red to Yellow or Green to Red must be prevented.
* Safety Rule: The light must never remain stuck at any state indefinitely.
* Initial State: The initial state must always be Red.

**System Constraints**

**Mutual Exclusivity:**

* Only one state (Red, Green, Yellow) can be active at a time.

**Safety:**

* No direct transition from Red to Yellow.

**Predictable Behavior:**

* The light must cycle through states in the correct order without skipping.

**Formal Representation**

We utilize Z3, a theorem prover, to define the system and verify its correctness.

**Variables:**

1. current\_state: Represents the current light state.
2. transition: Defines the allowed transitions between states.

**Constraints:**

* Initial State: Red
* Transition Rules: Red → Green → Yellow → Red
* Invalid transitions (e.g., Red → Yellow) must be blocked.

**System Inputs and Outputs**

**Inputs:**

* Light states (Red, Green, Yellow).

**Outputs:**

* The next valid state based on the current light state.
* An error message if invalid states are detected.

**Implementation**

**Dynamic Transaction Validation using Z3**

from z3 import \*

class TrafficLightZ3: def **init**(self): self.solver = Solver() self.define\_variables() self.set\_constraints()

def define\_variables(self):

self.Red, self.Green, self.Yellow = Ints('Red Green Yellow')

self.current\_state = Int('current\_state')

self.solver.add(self.current\_state == self.Red)

def set\_constraints(self):

transition = Function('transition', IntSort(), IntSort())

self.solver.add(transition(self.Red) == self.Green)

self.solver.add(transition(self.Green) == self.Yellow)

self.solver.add(transition(self.Yellow) == self.Red)

invalid\_transition = And(self.current\_state == self.Red, transition(self.Red) == self.Yellow)

self.solver.add(Not(invalid\_transition))

def verify(self):

if self.solver.check() == sat:

print("Traffic Light System is Correct. No Invalid States Detected.")

else:

print("Invalid State Detected. Review Transition Rules.")

if **name** == "**main**": system = TrafficLightZ3() system.verify()

**Verification and Testing**

**Testing Scenarios:**

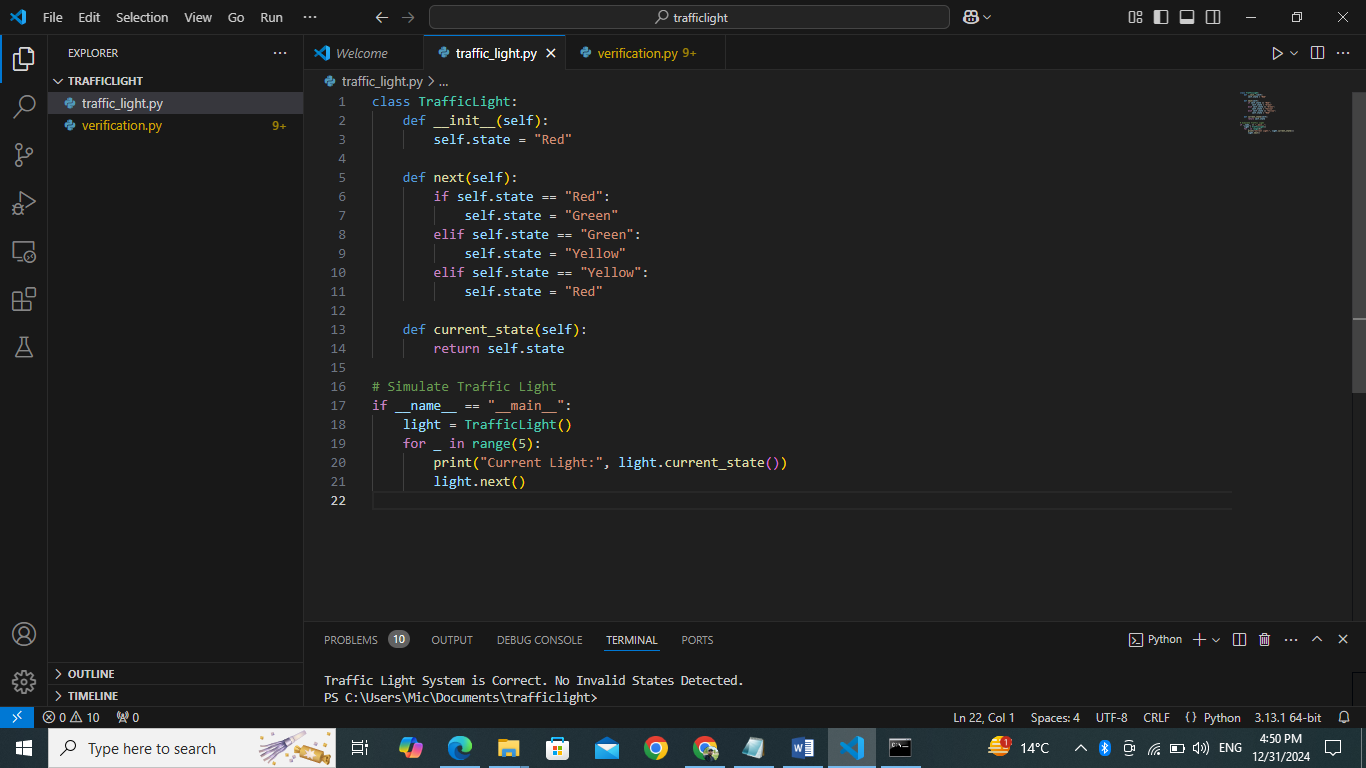
* Simulate sequential light transitions.
* Attempt invalid transitions (e.g., Red → Yellow).
* Verify that Z3 detects and blocks invalid states.

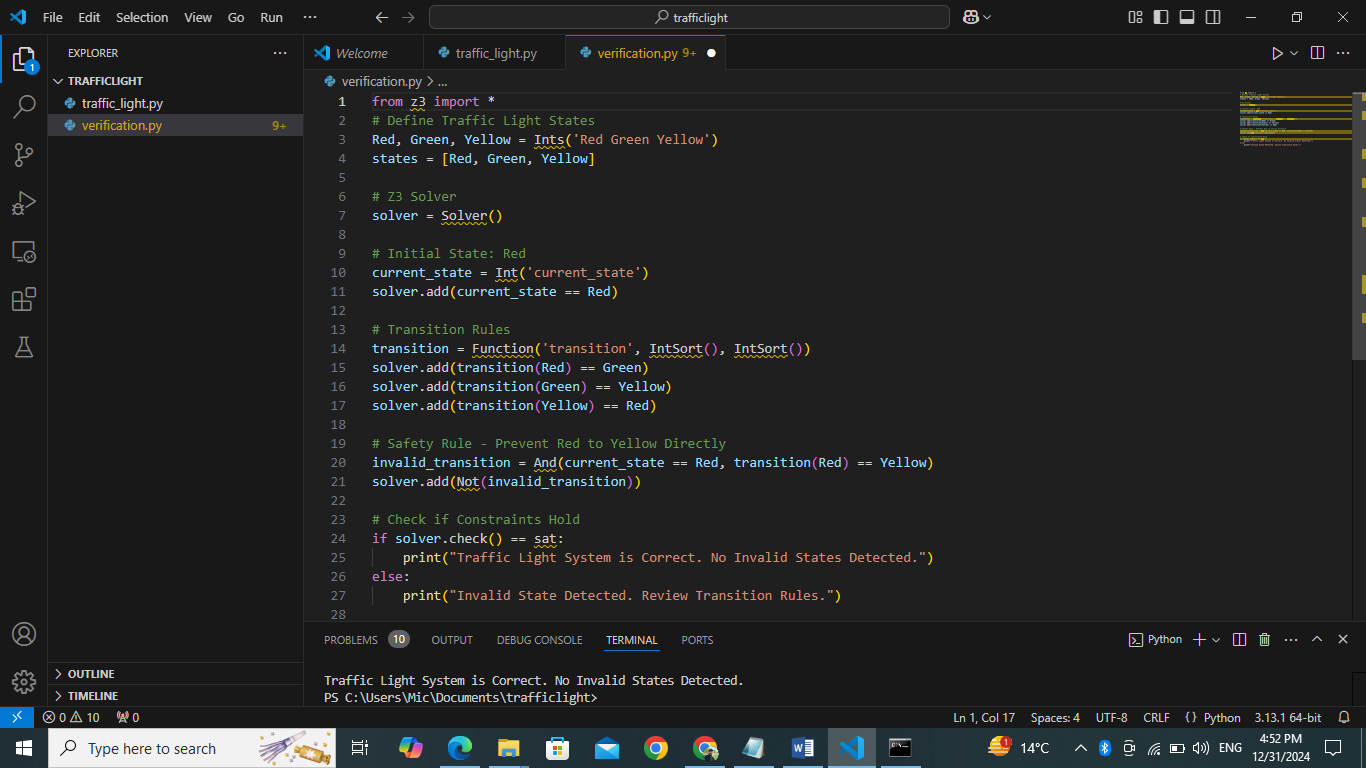
**Tools Used:**

* Z3 Theorem Prover for formal verification.
* Python for dynamic interaction and simulation.

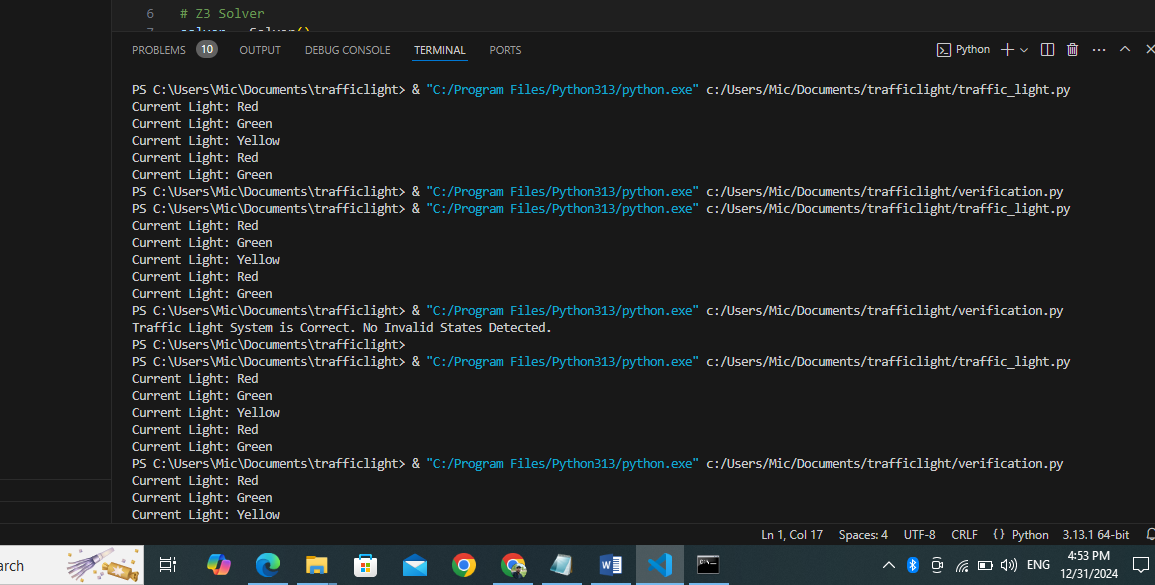
**Screenshots and Results**

**Code:**

****

****

**Output:**

****

**Challenges and Lessons Learned**

**Challenges:**

* Defining constraints that dynamically update during simulation.
* Preventing invalid transitions while ensuring predictable state changes.

**Lessons Learned:**

* The importance of formal methods in ensuring system safety.
* How theorem provers like Z3 can dynamically verify state transitions.

**Future Improvements**

* Implement a visual interface to simulate the traffic light system.
* Extend the model to include pedestrian signals.
* Add time constraints for light duration.

**References**

* Z3 Documentation: https://github.com/Z3Prover/z3
* Formal Methods Tutorials.