**Formal Methods Lab**

**1. Simple State Transition System**

**This program models a finite state machine with defined states and transitions.**

**class StateMachine:**

**def \_\_init\_\_(self):**

**self.state = "START" # Initial state**

**self.transitions = {**

**"START": {"next": "PROCESS"},**

**"PROCESS": {"next": "END", "reset": "START"},**

**"END": {"reset": "START"},**

**}**

**def transition(self, action):**

**if action in self.transitions[self.state]:**

**self.state = self.transitions[self.state][action]**

**return f"Transitioned to state: {self.state}"**

**else:**

**return f"Invalid action '{action}' for state '{self.state}'"**

**def get\_state(self):**

**return self.state**

**# Example usage**

**machine = StateMachine()**

**print(machine.transition("next")) # PROCESS**

**print(machine.transition("next")) # END**

**print(machine.transition("reset")) # START**

**2. Truth Table Generator and Verifier**

**This program generates truth tables for a given Boolean expression and verifies correctness.**

**import itertools**

**def generate\_truth\_table(expression, variables):**

**results = []**

**for values in itertools.product([False, True], repeat=len(variables)):**

**env = dict(zip(variables, values))**

**result = eval(expression, {}, env)**

**results.append((\*values, result))**

**return results**

**def verify\_truth\_table(user\_table, generated\_table):**

**return user\_table == generated\_table**

**# Input example**

**expression = input("Enter a Boolean expression (e.g., A and B or not A): ")**

**variables = sorted(set(filter(str.isalpha, expression)))**

**print("Variables:", variables)**

**truth\_table = generate\_truth\_table(expression, variables)**

**print("\nTruth Table:")**

**print(" | ".join(variables + ["Result"]))**

**for row in truth\_table:**

**print(" | ".join(map(str, row)))**

**# Example for verification**

**user\_table = [**

**(False, False, False), # Replace with user-provided table**

**(False, True, True),**

**(True, False, True),**

**(True, True, True),**

**]**

**print("\nVerification:", verify\_truth\_table(user\_table, truth\_table))**

**3. LTL Formula Verification Against FSM**

**This program checks if an LTL formula holds on all paths of a finite-state machine.**

**class FSM:**

**def \_\_init\_\_(self, states, transitions, start\_state):**

**self.states = states**

**self.transitions = transitions**

**self.start\_state = start\_state**

**def simulate(self):**

**current\_state = self.start\_state**

**while current\_state in self.transitions:**

**yield current\_state**

**current\_state = self.transitions[current\_state]**

**def verify\_ltl(formula, fsm):**

**for state in fsm.simulate():**

**if not eval(formula.replace("state", f"'{state}'")):**

**return False**

**return True**

**# Example FSM and LTL verification**

**states = {"A", "B", "C"}**

**transitions = {"A": "B", "B": "C", "C": "A"}**

**start\_state = "A"**

**fsm = FSM(states, transitions, start\_state)**

**ltl\_formula = "state == 'A' or state == 'B'" # Example LTL formula**

**print("LTL Verification:", verify\_ltl(ltl\_formula, fsm))**

**4. Traffic Light Controller Simulation**

**Simulates a reactive traffic light system.**

**import time**

**class TrafficLight:**

**def \_\_init\_\_(self):**

**self.states = ["RED", "YELLOW", "GREEN"]**

**self.current\_state = 0**

**def next\_state(self):**

**self.current\_state = (self.current\_state + 1) % len(self.states)**

**return self.states[self.current\_state]**

**def simulate(self, cycles=5):**

**for \_ in range(cycles):**

**print(f"Light is {self.states[self.current\_state]}")**

**time.sleep(1)**

**self.next\_state()**

**# Example usage**

**traffic\_light = TrafficLight()**

**traffic\_light.simulate()**

**5. Process Communication Using CSP**

**Implements process communication using Python's queue module to model message passing.**

**import threading**

**from queue import Queue**

**def process\_a(queue):**

**for i in range(5):**

**print(f"Process A: Sending {i}")**

**queue.put(i)**

**time.sleep(0.5)**

**def process\_b(queue):**

**while True:**

**message = queue.get()**

**if message is None: # Termination signal**

**break**

**print(f"Process B: Received {message}")**

**# Main simulation**

**queue = Queue()**

**thread\_a = threading.Thread(target=process\_a, args=(queue,))**

**thread\_b = threading.Thread(target=process\_b, args=(queue,))**

**thread\_a.start()**

**thread\_b.start()**

**thread\_a.join()**

**queue.put(None) # Send termination signal to process B**

**thread\_b.join()**