**Formal Methods Lab**

**Experiment-2**

**CS8A**

**1. Simulate a basic CCS process**

This models a CCS process that performs an action (a) and transitions to the next state.

class CCSProcess:

def \_\_init\_\_(self, name):

self.name = name

def perform\_action(self, action):

print(f"{self.name} performs action {action}")

return CCSProcess(f"{self.name}'") # Transition to a new state

# Example

process = CCSProcess("P")

next\_state = process.perform\_action("a")

print(f"New state: {next\_state.name}")

**2. Parallel composition of two CCS processes**

This models two concurrent processes executing independently.

import threading

class ParallelProcess:

def \_\_init\_\_(self, name):

self.name = name

def perform\_action(self, action):

print(f"{self.name} performs action {action}")

def execute\_parallel(process1, action1, process2, action2):

t1 = threading.Thread(target=process1.perform\_action, args=(action1,))

t2 = threading.Thread(target=process2.perform\_action, args=(action2,))

t1.start()

t2.start()

t1.join()

t2.join()

# Example

P1 = ParallelProcess("P1")

P2 = ParallelProcess("P2")

execute\_parallel(P1, "a", P2, "b")

**3. Pi-Calculus Communication (Sender-Receiver)**

This simulates message passing between two processes over a channel.

import queue

import threading

class PiCalculusChannel:

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

self.channel = queue.Queue()

def send(self, message):

print(f"Sender sends: {message}")

self.channel.put(message)

def receive(self):

message = self.channel.get()

print(f"Receiver receives: {message}")

# Example

channel = PiCalculusChannel()

sender = threading.Thread(target=channel.send, args=("Hello",))

receiver = threading.Thread(target=channel.receive)

sender.start()

receiver.start()

sender.join()

receiver.join()

**4. Verify synchronization between two CCS processes using complementary actions**

This ensures that complementary actions (a and ā) synchronize.

class SyncCCSProcess:

def \_\_init\_\_(self, name):

self.name = name

def perform\_action(self, action):

return action

def synchronize(p1, action1, p2, action2):

if action1 == f"{action2}̄" or action2 == f"{action1}̄":

print(f"Synchronization successful: {p1.name} ({action1}) ↔ {p2.name} ({action2})")

else:

print(f"No synchronization: {p1.name} ({action1}) | {p2.name} ({action2})")

# Example

P1 = SyncCCSProcess("P1")

P2 = SyncCCSProcess("P2")

synchronize(P1, "a", P2, "ā")

synchronize(P1, "b", P2, "c") # No sync

**5. Producer-Consumer System with Mutual Exclusion**

This ensures correct handling of shared resources.

import threading

import queue

import time

buffer = queue.Queue(maxsize=5)

lock = threading.Lock()

def producer():

for i in range(5):

time.sleep(1)

with lock:

buffer.put(i)

print(f"Produced: {i}")

def consumer():

for i in range(5):

time.sleep(2)

with lock:

item = buffer.get()

print(f"Consumed: {item}")

# Start producer and consumer threads

prod\_thread = threading.Thread(target=producer)

cons\_thread = threading.Thread(target=consumer)

prod\_thread.start()

cons\_thread.start()

prod\_thread.join()

cons\_thread.join()