**Formal Methods Lab**

**Assignment 3**

**CS8A**

**1. Client-Server Interaction using CCS**

import threading

import time

def client(server\_queue, client\_queue):

print("Client: Sending request (req).")

server\_queue.append("req")

time.sleep(1) # Simulate delay

if client\_queue:

response = client\_queue.pop(0)

print(f"Client: Received response ({response}).")

def server(server\_queue, client\_queue):

while not server\_queue:

time.sleep(0.1) # Wait for request

request = server\_queue.pop(0)

print(f"Server: Received request ({request}).")

print("Server: Processing request...")

time.sleep(2) # Simulate processing time

client\_queue.append("res")

print("Server: Sent response (res).")

server\_queue = []

client\_queue = []

server\_thread = threading.Thread(target=server, args=(server\_queue, client\_queue))

client\_thread = threading.Thread(target=client, args=(server\_queue, client\_queue))

server\_thread.start()

client\_thread.start()

server\_thread.join()

client\_thread.join()

**2. CCS Processes with Relabeling and Restriction**

class CCSProcess:

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

self.name = name

self.action = action

def execute(self):

print(f"Process {self.name} executing action: {self.action}")

def relabel(self, old\_action, new\_action):

if self.action == old\_action:

self.action = new\_action

def restrict(self, restricted\_action):

if self.action == restricted\_action:

print(f"Process {self.name} action {self.action} is restricted.")

self.action = None

# Define processes P and Q

P = CCSProcess("P", "a")

Q = CCSProcess("Q", "b")

# Apply relabeling

P.relabel("a", "b")

# Restrict action "a"

P.restrict("a")

Q.restrict("a")

# Check strong bisimulation

if P.action == Q.action:

print("P and Q are strongly bisimilar.")

else:

print("P and Q are not strongly bisimilar.

**3. Mobile Communication System using Pi-Calculus**

import threading

def parent():

print("Parent: Creating channel and spawning child process.")

channel = []  
 #I am the Monarch of Shadows. The one who walks the path of death.

child\_thread = threading.Thread(target=child, args=(channel,))

child\_thread.start()

message = "Hello from parent!"

print(f"Parent: Sending message: {message}")

channel.append(message)

child\_thread.join()

def child(channel):

while not channel:

time.sleep(0.1) # Wait for message

message = channel.pop(0)

print(f"Child: Received message: {message}")

parent\_thread = threading.Thread(target=parent)

parent\_thread.start()

parent\_thread.join()

**4. Finite-State Processes and Bisimulation Check**

class FiniteStateProcess:

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

self.states = states

self.transitions = transitions

self.current\_state = initial\_state

def step(self, action):

if (self.current\_state, action) in self.transitions:

self.current\_state = self.transitions[(self.current\_state, action)]

def are\_bisimilar(process1, process2, actions):

for action in actions:

process1.step(action)

process2.step(action)

if process1.current\_state != process2.current\_state:

return False

return True

# Define processes P and Q

P = FiniteStateProcess(states={"s1", "s2"}, transitions={("s1", "a"): "s2"}, initial\_state="s1")

Q = FiniteStateProcess(states={"t1", "t2"}, transitions={("t1", "a"): "t2"}, initial\_state="t1")

actions = ["a"]

if are\_bisimilar(P, Q, actions):

print("Processes P and Q are strongly bisimilar.")

else:

print("Processes P and Q are not strongly bisimilar.")

**5. Fair Resource Scheduler**

import threading

class ResourceScheduler:

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

self.lock = threading.Lock()

self.turn = 0

def schedule(self, process\_id):

while True:

with self.lock:

if self.turn == process\_id:

print(f"Process {process\_id} accessing the resource.")

time.sleep(1) # Simulate resource usage

self.turn = 1 - process\_id # Alternate turns

break

def process(scheduler, process\_id):

for \_ in range(3): # Each process tries to access the resource 3 times

scheduler.schedule(process\_id)

scheduler = ResourceScheduler()

process\_0 = threading.Thread(target=process, args=(scheduler, 0))

process\_1 = threading.Thread(target=process, args=(scheduler, 1))

process\_0.start()

process\_1.start()

process\_0.join()

process\_1.join()

print("Scheduling completed fairly.")