ECEN689-602/CSCE689-603 Introduction to Formal Verification Fall 2021

Project Report Phase A

Submitted to the Faculty

Of

Texas A&M University

By

Team 6 I-Group

Brent Arnold Basiano UIN: 130004869

Sri Hari Pada Kodi UIN: 932003040

Under the Guidance of

Professor Jiang Hu

Department of Electrical and Computer Engineering

November 2021

TABLE OF CONTENTS

LIST OF FIGURES

ECEN 689 - Introduction to Formal Verification

Phase A

Team 6 - I-Group

1. Abstract

A road grid with a traffic system was designed for Phase A which required a software program for the road system's signal control. The signal control is a two-dimensional array where each array has at most 1 green light in each intersection, and the road network is a directed graph where vertices are the intersections and edges are road segments. The traffic signal works by rotating the array to simulate a sequence for the traffic signals. The I-Group created its own vehicle behavior to help determine if the traffic signals work.

2. Background

The project is to design a road system and verify its properties. All cars start in a common point in the road called point A which moves to other points labelled B, C, and D. The car must go to all the points in any order but must return to point A. For this project, two groups in a team deal with the road (I-Group) and vehicles (V-Group) separately. The project is also broken down to three phases. This report focuses on Phases A.

For phase A, the V-Group codes the vehicle in the road system. Vehicle behavior can move in any direction except a U-turn, go 30mph or stop, must not collide with any cars, and should not breach a red light. V-Group does not know the traffic signals. The I-Group codes the signal control in the road system. The signal control behavior must be that the traffic signals in each intersection would have at most 1 green light turned on. In an intersection, only one car can go through at a time. I-Group would take vehicle behavior but does not know where the vehicles are heading.

3. Procedures

Before designing the traffic signals for the road, a road must be designed. The road system was coded in Python and was designed using a directed graph. Each edge is the road segment between two intersections which are the vertices. To create the road segment, a nested dictionary is created where the keys are the intersections. Figure 1 shows the road system, Figure 2 shows the road as a graph, and Figure 3 shows the graph code. Full code is in the Appendix.

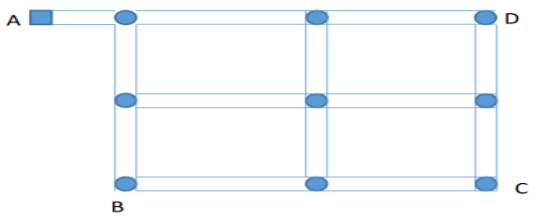

Figure 1. Road System

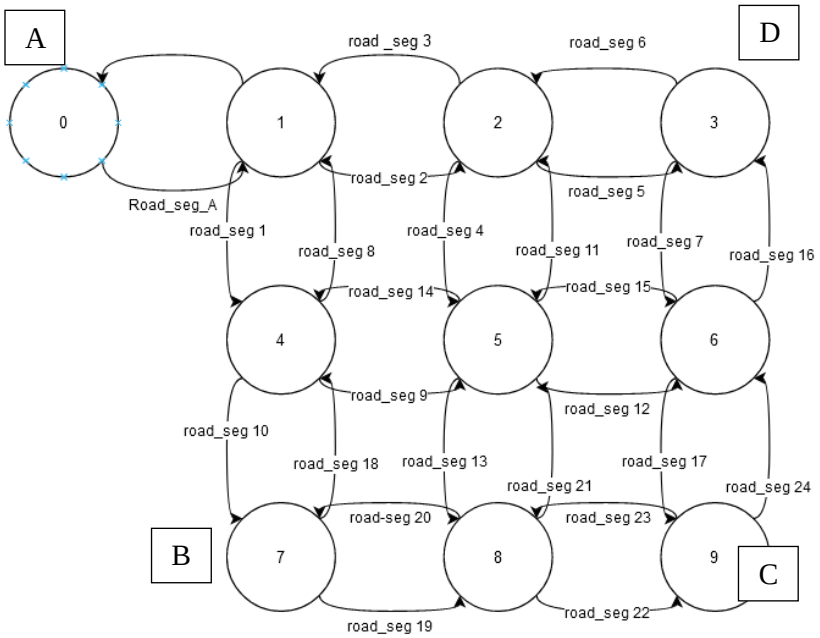

Figure 2: Directed Graph of the Road System

```
graph = {
    0: { 1: [road_seg_A, (0,0)]},
    1: { 0: [road_seg[0]], 2: [road_seg[2], (1,0)], 4: [road_seg[1], (3,2)]},
    2: { 1: [road_seg[3], (0,2)], 5: [road_seg[4], (4,3)], 3: [road_seg[5], (2,0)]},
    3: { 2: [road_seg[6], (1,2)], 6: [road_seg[7], (5,2)]},
    4: { 1: [road_seg[8], (0,1)], 5: [road_seg[9], (4,0)], 7: [road_seg[10], (6,1)]},
    5: { 2: [road_seg[11], (1,1)], 4: [road_seg[14], (3,1)], 6: [road_seg[12], (5,0)], 8: [road_seg[13], (7,2)]},
    6: { 3: [road_seg[16], (2,1)], 5: [road_seg[15], (4,2)], 9: [road_seg[17], (8,1)]},
    7: { 4: [road_seg[18], (3,0)], 8: [road_seg[19], (7,0)]},
    8: { 5: [road_seg[21], (4,1)], 7: [road_seg[20], (6,0)], 9: [road_seg[23], (8,0)]},
    9: { 6: [road_seg[23], (5,1)], 8: [road_seg[24], (7,1)]}
}
```

Figure 3: Graph Implementation

In Figure 3, the graph [0][1] means access the road segment and traffic signal between 0 and 1. Each road segment between the intersections is 0.5 miles while the road between the starting point(A) and the first intersection is 1/30 of a mile. One road segment has 30 uniformed slots. To help keep track of the cars in the road segment, a double-ended queue or deque is used to create a road segment. The 0^{th} index is where the cars enter while the 29^{th} index will be the slot where the car moves out of the segment to another segment. Each road is linked to the directed graph.

For Phase A, I-Group needs to code the traffic signal of the road system. The traffic signal is in a two-dimensional list which contains 9 lists representing the 9 intersections. For red signals, the value is 0 while green signals are 1. At each intersection, at most 1 green light is present. For the initial property of the traffic signal, I-Group determines which light is green while V-Group does not know the information about the current lights. To create a sequence, an array rotation algorithm was used to rotate the traffic signals. In Figure 4, 5, and 6, the traffic signal code, rotation algorithm, and layout of the traffic signals on the grid.

```
traff_sig = [
    [1,0,0],
    [0,1,0],
    [0,1],
    [1,0,0],
    [0,0,1,0],
    [0,1,0],
    [0,1],
    [0,0,1],
    [0,1]
]
```

Figure 4: Traffic Code

```
def traffic_update():
    for i in range(len(traff_sig)):
        traff_sig[i] = traff_sig[i][1:len(traff_sig[i])] + traff_sig[i][0:1]
```

Figure 5: Traffic Signal Rotation Algorithm

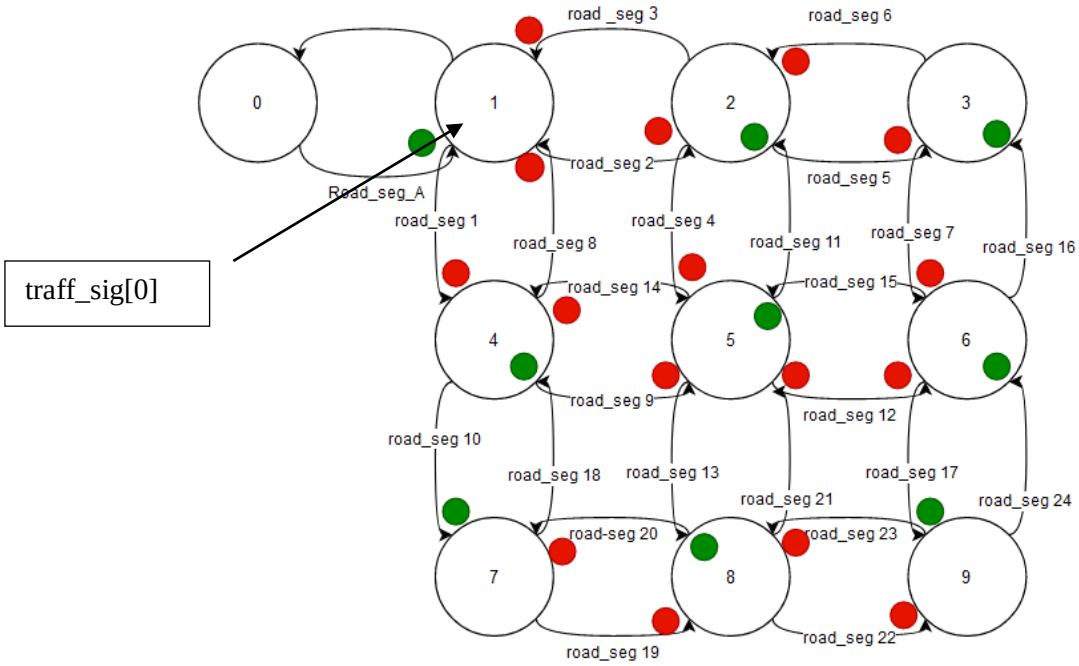

Figure 6: Road System with Intersection based on Figure 4

When running the simulation, the traffic update function would change the lights for every time step which is of two seconds. The output of the traffic signal is shown in Figure 7 - 9.

```
while True:
    sleep(2)
    update()
    print(traff_sig)
```

Figure 7: Print Current Traffic Signal Lights

```
C:\Users\Brent Basiano>python -u "c:\Users\Brent Basiano\Desktop\road_test.py"
[[1, 0, 0], [0, 1, 0], [0, 1], [1, 0, 0], [0, 0, 1, 0], [0, 1, 0], [0, 1], [0, 0, 1], [0, 1]]
[[0, 0, 1], [1, 0, 0], [1, 0], [0, 0, 1], [0, 1, 0, 0], [1, 0, 0], [1, 0], [0, 1, 0], [1, 0]]
[[0, 1, 0], [0, 0, 1], [0, 1], [0, 1, 0], [1, 0, 0, 0], [0, 0, 1], [0, 1], [1, 0, 0], [0, 1]]
[[1, 0, 0], [0, 1, 0], [1, 0], [1, 0, 0], [0, 0, 0, 1], [0, 1, 0], [1, 0], [0, 0, 1], [1, 0]]
```

Figure 8: Traffic Signal Sequence

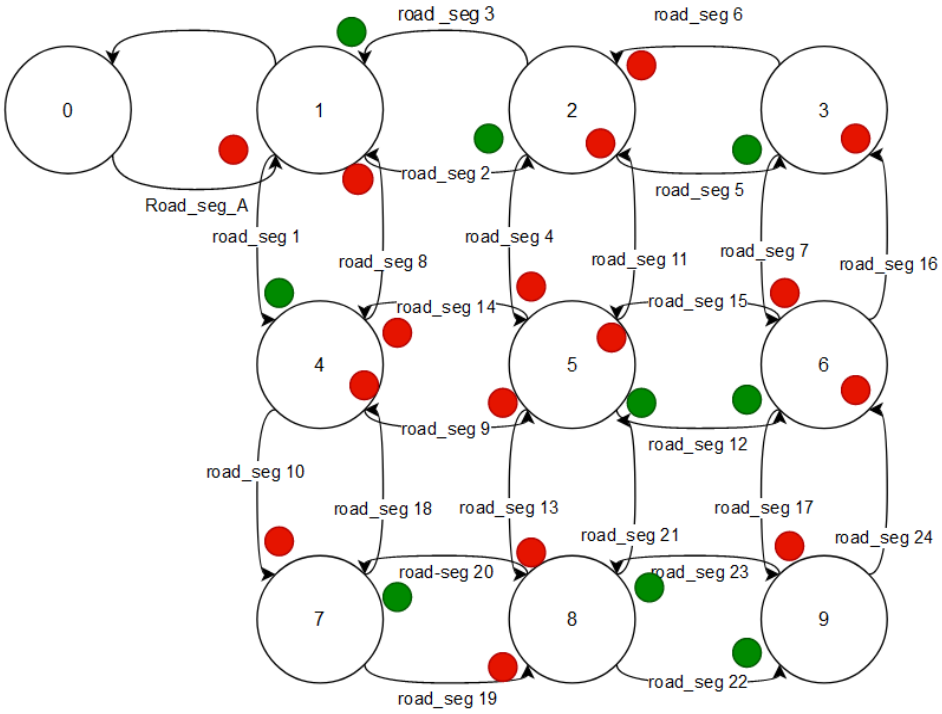

Figure 9: Traffic Signal Sequence After Rotating Once

For testing if the traffic signals work properly, a car was added to test the traffic signals. Since V-Group is creating the vehicle behavior, I-Group assumes a vehicle behavior for testing purposes to see if the vehicle behaves with the traffic signals.

```
[[0, 0, 1], [1, 0, 0], [1, 0], [0, 0, 1], [0, 1, 0, 0], [1,                    0]]
Road seg  0  ->  1
[[0, 1, 0], [0, 0, 1], [0, 1], [0, 1, 0], [1, 0, 0, 0], [0,                    1]]
Road seg  0  ->  1
[[1, 0, 0], [0, 1, 0], [1, 0], [1, 0, 0], [0, 0, 0, 1], [0,                    0]]
Road seg  0  ->  1
[[0, 0, 1], [1, 0, 0], [0, 1], [0, 0, 1], [0, 1, 0, 0], [1,                    1]]
Road seg  1  ->  4
[[0, 1, 0], [0, 0, 1], [1, 0], [0, 1, 0], [0, 1, 0, 0], [0, 0, 1], [1, 0], [1, 0, 0], [1, 0]]
Road seg  1  ->  4
```

Starting in edge 0 - > 1, car waits until traff_sig[0][0] = 1. Once traff_sig[0][0] = 1, the car moves to the next road segment.

Figure 10: Car Movement Test Result

4. Conclusion

Phase A, for I-Group is to design a traffic signal control for the road system. The traffic signal lights are created using a two-dimensional array containing zeroes and ones. Zeroes represent red lights, and ones represent green lights. The traffic signal sequence is created by rotating the array by one increment. A car was added to test the traffic signal, but V-Group creates the vehicle behavior. Therefore, the car behavior in this test was only an assumption. Phase B will be to combine both programs and determine a verification tool which will be used for Phase C.

5. Appendix

```
"""
directed graph
0 -> 1 ::= 0: { 1: [road_seg_A, (0,0)]} -
> previous vertex : next vertex: [deque road segment, tuple (containin
g i and j for traff_sig[i][j])]

traffic signal:
red = 0
green = 1

car capacity_max = 30

1 iteration = 2 seconds
"""
from collections import deque
from time import sleep


# initial traffic signal V-Group does not know
traff_sig = [
    [1,0,0],
    [0,1,0],
    [0,1],
    [1,0,0],
    [0,0,1,0],
    [0,1,0],
    [0,1],
    [0,0,1],
    [0,1]
]

# road segment from point A (0) to first intersection (1)
road_seg_A = deque([0 for _ in range(2)])

# road segments
road_seg = [deque([0 for _ in range(30)])] * 25


# test car (V-group has own design for integration)
car = {
    1: {'path': [0,1,4,7,8,9,6,3,2,1,0], 'current_road_i': 0, 'prev_ro
ad_i': 0, 'next_road_i': 1}
```

```
    # 2: {'path': [0,1,2,3,6,9,8,7,4,1,0], 'current_road_i': 0, 'prev_
road_i': 0, 'next_road_i': 1}
}

# see what cars are in the road systems
car_queue = deque([])

# road network
graph = {
    0: { 1: [road_seg_A, (0,0)]},
    1: { 0: [road_seg[0]], 2: [road_seg[2], (1,0)], 4: [road_seg[1], (
3,2)]},
    2: { 1: [road_seg[3], (0,2)], 5: [road_seg[4], (4,3)], 3: [road_se
g[5], (2,0)]},
    3: { 2: [road_seg[6], (1,2)], 6: [road_seg[7], (5,2)]},
    4: { 1: [road_seg[8], (0,1)], 5: [road_seg[9], (4,0)], 7: [road_se
g[10], (6,1)]},
    5: { 2: [road_seg[11], (1,1)], 4: [road_seg[14], (3,1)], 6: [road_
seg[12], (5,0)], 8: [road_seg[13], (7,2)]},
    6: { 3: [road_seg[16], (2,1)], 5: [road_seg[15], (4,2)], 9: [road_
seg[17], (8,1)]},
    7: { 4: [road_seg[18], (3,0)], 8: [road_seg[19], (7,0)]},
    8: { 5: [road_seg[21], (4,1)], 7: [road_seg[20], (6,0)], 9: [road_
seg[23], (8,0)]},
    9: { 6: [road_seg[23], (5,1)], 8: [road_seg[24], (7,1)]}
}

# Points A,B,C,D
A = 0
B = 7
C = 9
D = 3

# change traffic signals to next sequence
def traffic_update():
    for i in range(len(traff_sig)):
        traff_sig[i] = traff_sig[i][1:len(traff_sig[i])] + traff_sig[i
][0:1]

# car changes to different road segment
def intersect_update(road_seg, traffic_signal):
    if traffic_signal:
        id = road_seg.pop()
        road_seg.appendleft(0)
        car[id]['prev_road_i'] = car[id]['current_road_i']
```

```
        car[id]['current_road_i'] += 1
        car[id]['next_road_i'] += 1
        current_road = car[id]['current_road_i']
        next_road = car[id]['next_road_i']
        graph[car[id]['path'][current_road]][car[id]['path'][next_road
]][0].appendleft(id)
        graph[car[id]['path'][current_road]][car[id]['path'][next_road
]][0].pop()

# car moves to next slot in a road segment
def road_seg_update(road_seg, car_id):
    global traff_sig
    current_road = car[car_id]['current_road_i']
    next_road = car[car_id]['next_road_i']
    intersect, road = graph[car[car_id]['path'][current_road]][car[car
_id]['path'][next_road]][1]

    if car_id in road_seg and car_id != road_seg[-1]:
        road_seg.pop()
        road_seg.appendleft(0)
    elif car_id in road_seg and car_id == road_seg[-1]:
        intersect_update(road_seg, traff_sig[intersect][road])
    else:
        road_seg[0] = car_id

# overall update of the road system
def update():

    for i in range(len(car_queue)-1, -1, -1):
        current_road = car[car_queue[i]]['current_road_i']
        next_road = car[car_queue[i]]['next_road_i']
        if current_road == 0 and car_queue[i] not in road_seg_A and ca
r_queue[i] not in road_seg:
            if road_seg_A[0] == 0:
                road_seg_update(road_seg_A, car_queue[i])
        elif current_road == 0 and car_queue[i] in road_seg_A:
            road_seg_update(road_seg_A, car_queue[i])
        else:
            road_seg_update(graph[car[car_queue[i]]['path'][current_ro
ad]][car[car_queue[i]]['path'][next_road]][0], car_queue[i])


    traffic_update() #change traffic signal at the end of the update
```

```
# input car ids to car queue
for id in car.keys():
    car_queue.appendleft(id)

# main loop
while True:
    sleep(2) #2 second time interval
    update()
    print(traff_sig) # print next traffic signal sequence
    # current_road = car[1]['current_road_i']
    # next_road = car[1]['next_road_i']
    # current_road2 = car[2]['current_road_i']
    # next_road2 = car[2]['next_road_i']
    # print("Road seg ", car[1]['path'][current_road], " -
> ", car[1]['path'][next_road])
    # print("Road seg ", car[2]['path'][current_road2], " -
> ", car[2]['path'][next_road2])
    # print(road_seg[0])
```

ECEN689-602/CSCE689-603 Introduction to Formal Verification Fall 2021

Project Report Phase B

Submitted to the Faculty

Of

Texas A&M University

By

Team 6 I-Group Integration

Brent Arnold Basiano UIN: 130004869

Sri Hari Pada Kodi UIN: 932003040

Under the Guidance of

Professor Jiang Hu

Department of Electrical and Computer Engineering

November 2021

TABLE OF CONTENTS

LIST OF FIGURES

ECEN 689 - Introduction to Formal Verification

Phase B

Team 6 - I-Group

1. Abstract

Phase B integrated software to create the full simulation software of the road system. The traffic signal lights were converted to a one-dimensional array to allow the array to be allocated to shared memory for the vehicles. The traffic signal sequence algorithm was modified to allow subarrays to rotate with respect to their intersections. The road was converted to a Numpy array which contained the current car slot and the associated the car id. Vehicle behavior stayed the same. After integration, the software was successfully simulated with 30 cars with no traffic violation, collisions, and U-turns.

2. Background

The project is to design a road system and verify its properties. All cars start in a common point in the road called point A which moves to other points labelled B, C, and D. The car must go to all the points in any order but must return to point A. For this project, two groups in a team deal with the road (I-Group) and vehicles (V-Group) separately. The project is also broken down to three phases. This report focuses on Phases B.

For phase B, the I-Group focused on integration of both programs. Vehicle behavior provided by V-Group would know traffic light signals made by I-Group. The road system created by I-Group should know where vehicles are located. The V-Group focused on identifying a model checking software that will be used in this project. A demonstration of the model checker will also be given by V-Group

3. Procedures

To allow integration, certain aspects of the program were changed. For the road system, a directed graph was being used to access the traffic signal keys while the road segments are converted into a Numpy two-dimensional array which would contain the 30 slots and the vehicle IDs. For the traffic signals, the data structure was converted from a two-dimensional array to a one-dimensional array. A subarray within the array specifies a certain intersection. Traffic signal sequence follows the same algorithm but done for each subarray.

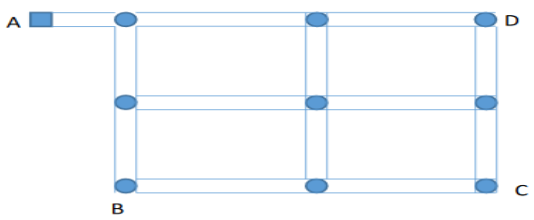

Figure 1. Road System

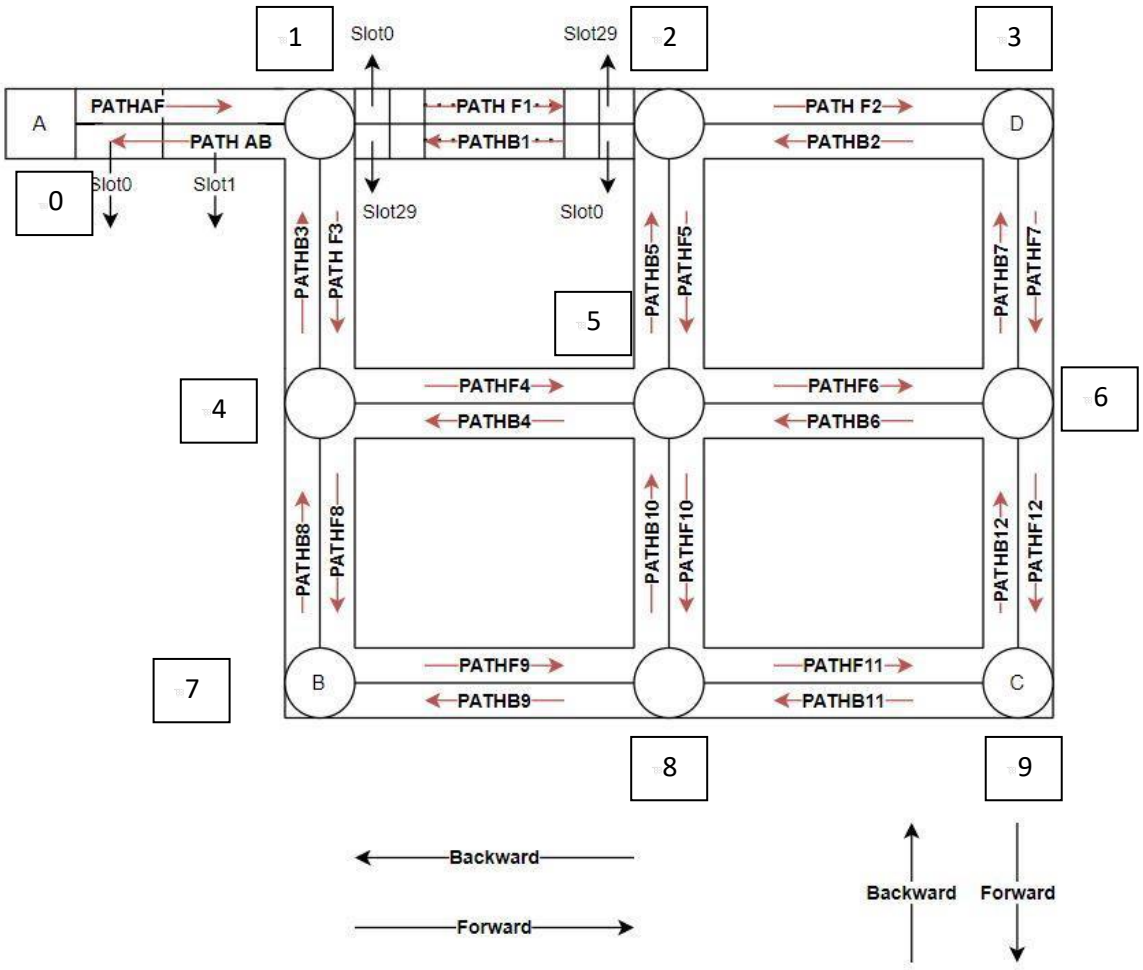

Figure 2: Directed Graph of the Road System

```
graph = {
    0: { 1: [road_seg_A, 0]},
    1: { 0: [road_seg[0], 25], 2: [road_seg[2], 3], 4: [road_seg[1], 10]},
    2: { 1: [road_seg[3], 2], 5: [road_seg[4], 14], 3: [road_seg[5], 6]},
    3: { 2: [road_seg[6], 5], 6: [road_seg[7], 17]},
    4: { 1: [road_seg[8], 1], 5: [road_seg[9], 11], 7: [road_seg[10], 19]},
    5: { 2: [road_seg[11], 4], 4: [road_seg[14], 9], 6: [road_seg[12], 15], 8: [road_seg[13], 22]},
    6: { 3: [road_seg[16], 7], 5: [road_seg[15], 13], 9: [road_seg[17], 24]},
    7: { 4: [road_seg[18], 8], 8: [road_seg[19], 20]},
    8: { 5: [road_seg[21], 12], 7: [road_seg[20], 18], 9: [road_seg[23], 23]},
    9: { 6: [road_seg[23], 16], 8: [road_seg[24], 21]}
}
```

Figure 3: Graph Implementation

In Figure 3, the graph [0][1] means access the road segment and traffic signal between 0 and 1. Each road segment between the intersections is 0.5 miles while the road between the starting point(A) and the first intersection is 1/30 of a mile. One road segment has 30 uniformed slots. To

help keep track of the cars in the road segment, a Numpy two-dimensional array is used to keep track of the slots and vehicle IDs.

For the traffic signal, the original implementation was to utilize a two-dimensional array where each ith index is the intersection and jth index is the current light signal of the road segment. Instead, the array has been converted to a one-dimensional array where a specified subarray is an intersection. This conversion is to allow the multiprocessing module in Python to put the array into a shared memory for the vehicles. To create a sequence, a subarray rotation algorithm was used to rotate the traffic signals. In Figure 4, 5, and 6, old the traffic signal code, the new traffic signal code, and the new rotation algorithm.

```
traff_sig = [
    [1,0,0],
    [0,1,0],
    [0,1],
    [1,0,0],
    [0,0,1,0],
    [0,1,0],
    [0,1],
    [0,0,1],
    [0,1]
]
```

Figure 4: Old Traffic Code

```
traff_sig_1d = [1,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,1,1]

traff_sig = mp.Array('i', traff_sig_1d)
```

Figure 5: New Traffic Code

```
def TrafficUpdate():
  while car_count.value:
    time.sleep(2)
    traff_sig[0:3] = traff_sig[1:3] + traff_sig[0:1]
    traff_sig[3:6] = traff_sig[4:6] + traff_sig[3:4]
    traff_sig[6:8] = traff_sig[7:8] + traff_sig[6:7]
    traff_sig[8:11] = traff_sig[9:11] + traff_sig[8:9]
    traff_sig[11:15] = traff_sig[12:15] + traff_sig[11:12]
    traff_sig[15:18] = traff_sig[16:18] + traff_sig[15:16]
    traff_sig[18:20] = traff_sig[19:20] + traff_sig[18:19]
    traff_sig[20:23] = traff_sig[21:23] + traff_sig[20:21]
    traff_sig[23:25] = traff_sig[24:25] + traff_sig[23:24]
    traff_sig[25] ^= 1
```

Figure 6: New Rotation Algorithm

Looking at Figure 4 and 5, the traffic signal is converted to a one-dimensional array to allow the traffic signal to be allocated in shared memory for the road system. For the traffic signal rotation in Figure 6, the rotation is based on the subarray which represents the intersection. Figure 7 shows the visual representation of the Figure 5 traffic signals.

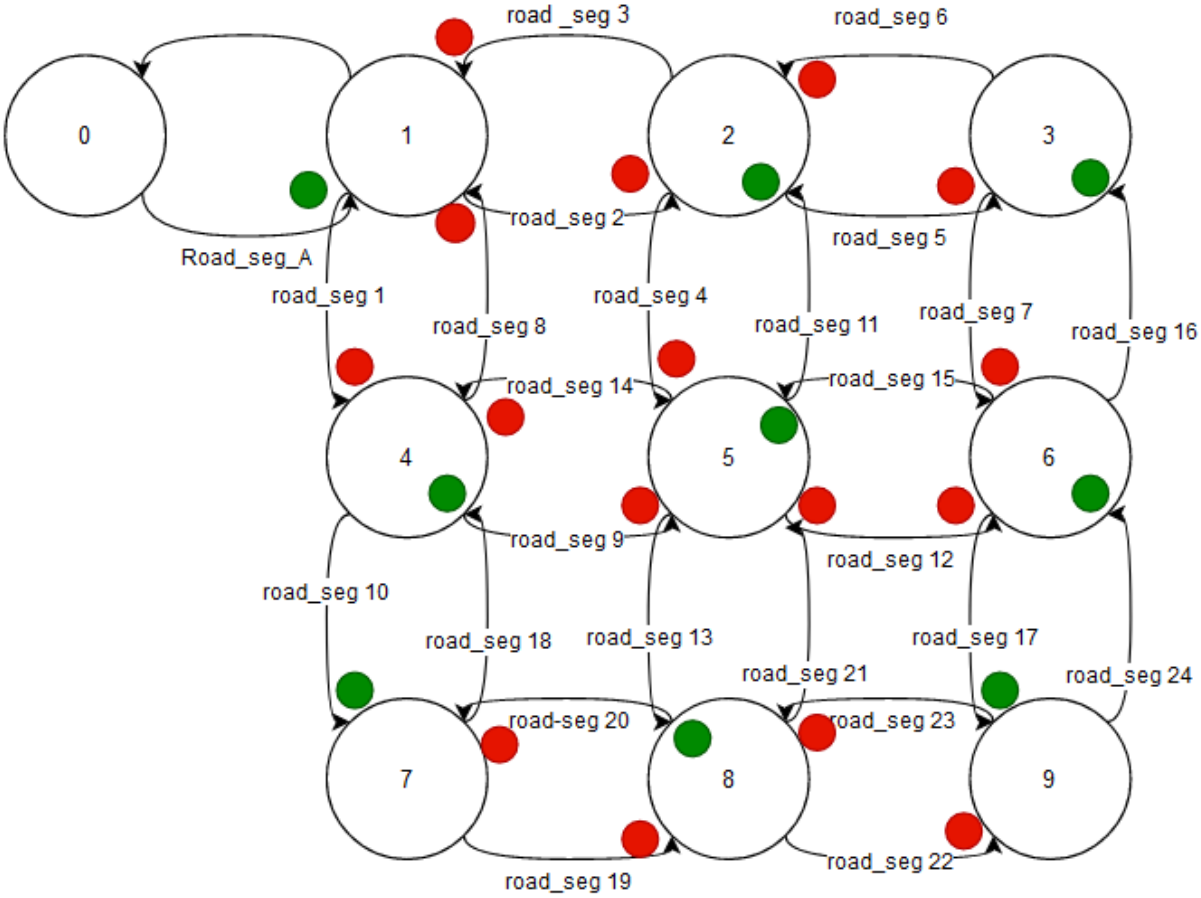

Figure 7: Representation of Traffic Signals based on Figure 5

For the simulation of the software, the original 15 cars have been increased to 30 cars. The simulation successfully finished as shown in Figure 8 with a throughput of 115 cars/hr, no red-light violation and collisions.

```
vehicle 12 is in pathB 1 and slot 11
vehicle 6 returned to A covering B C and D
vehicle 13 is in pathB 3 and slot 25
vehicle 10 is in pathAB and slot 0
vehicle 15 is in pathB 3 and slot 19
vehicle 11 is in pathB 1 and slot 29
vehicle 14 is in pathB 3 and slot 22
0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0
6
```

Car ID 6 Finished the course.

Next Traffic Signal Sequence

6 Cars Remaining

```
1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 1
30
vehicle 9 is in pathB 11 and slot 28
vehicle 12 is in pathB 11 and slot 22
vehicle 16 is in pathB 12 and slot 21
vehicle 1 is in pathB 1 and slot 4
vehicle 4 is in pathB 2 and slot 26
vehicle 18 is in pathB 12 and slot 19
vehicle 23 is in pathB 11 and slot 16
vehicle 2 is in pathB 1 and slot 1
for vehicle 3 to move to next pathB 1 signal is red
```

For vehicle 3 to move to path B1, light must be green (1). Therefore, red light violation is avoided.

```
for vehicle 2 Slot 0 of pathAF is full
vehicle 1 is in pathAF and slot 0
for vehicle 3 Slot 0 of pathAF is full
for vehicle 4 Slot 0 of pathAF is full
for vehicle 5 Slot 0 of pathAF is full
for vehicle 6 Slot 0 of pathAF is full
for vehicle 7 Slot 0 of pathAF is full
for vehicle 8 Slot 0 of pathAF is full
for vehicle 9 Slot 0 of pathAF is full
for vehicle 10 Slot 0 of pathAF is full
for vehicle 11 Slot 0 of pathAF is full
for vehicle 12 Slot 0 of pathAF is full
for vehicle 13 Slot 0 of pathAF is full
for vehicle 14 Slot 0 of pathAF is full
for vehicle 15 Slot 0 of pathAF is full
for vehicle 16 Slot 0 of pathAF is full
for vehicle 17 Slot 0 of pathAF is full
for vehicle 19 Slot 0 of pathAF is full
for vehicle 18 Slot 0 of pathAF is full
for vehicle 20 Slot 0 of pathAF is full
for vehicle 21 Slot 0 of pathAF is full
for vehicle 23 Slot 0 of pathAF is full
for vehicle 22 Slot 0 of pathAF is full
for vehicle 24 Slot 0 of pathAF is full
for vehicle 25 Slot 0 of pathAF is full
for vehicle 27 Slot 0 of pathAF is full
for vehicle 26 Slot 0 of pathAF is full
for vehicle 28 Slot 0 of pathAF is full
for vehicle 29 Slot 0 of pathAF is full
for vehicle 30 Slot 0 of pathAF is full
```

While car 1 is in path AF slot 0, no cars cannot go to the 0^{th} slot. Therefore, collision is avoided.

```
vehicle 25 is in pathAB  and slot 1
1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0
1
vehicle 25 returned to A covering B C and D
0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1
0
Finished in 936.2 second(s)
```

30 cars/936s * 3600s/hr = 115 cars/hr

Figure 8: Simulation Snippets

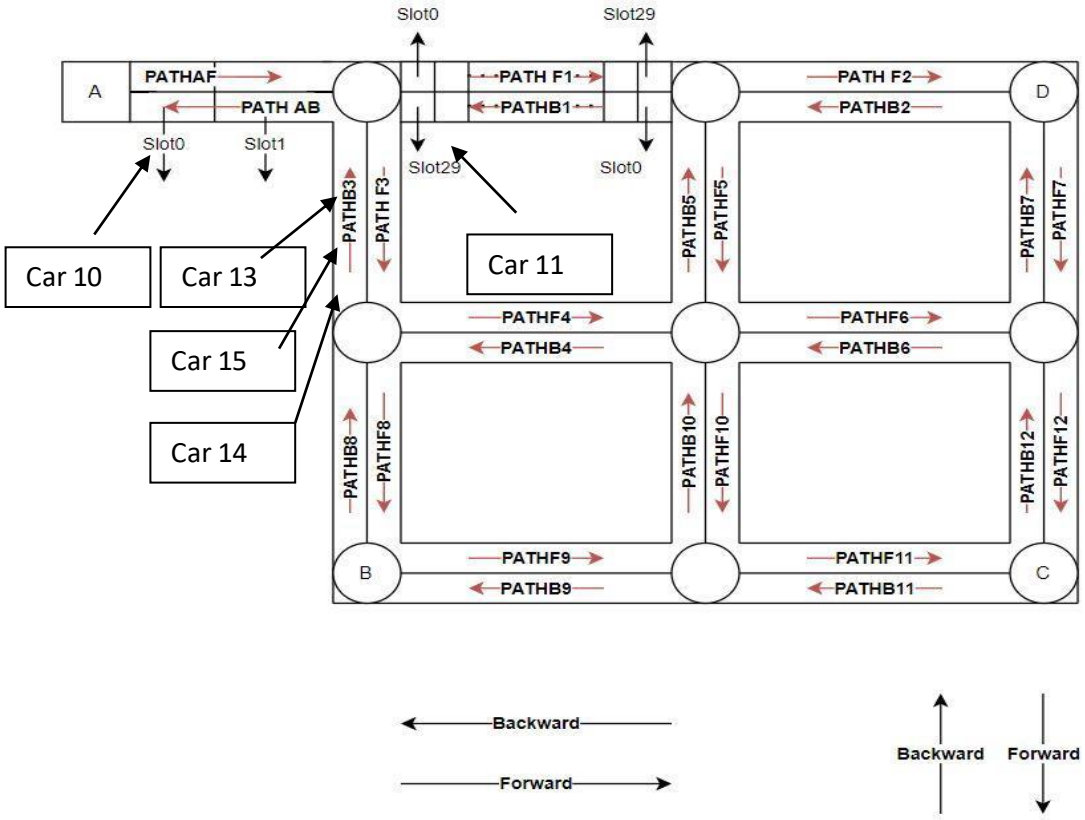

Figure 9: Visual Representation of Cars from Figure 8

4. Conclusion

For Phase B, I-Group was to integrate both programs. The traffic signal lights were converted to a one-dimensional array to allow the multiprocessor module to allocate it in shared memory for the vehicles. The road was converted to a Numpy array which makes the road compatible to vehicle behaviors. The vehicle behaviors provided by V-Group was not modified, but the number of vehicles was increased to 30 vehicles. The software successfully simulated with no violations meaning the integration was successful.

5. Appendix

```
import numpy as np
import ctypes as c
import multiprocessing as mp
from multiprocessing import Process,Array,Value
import time
from collections import deque
start=time.perf_counter()
#randon signals used for testing code
# signal=[0,1,0,1,0,1,1,1,1,1,1,1]

# traff_sig = [
#     [1,0,0], 0:3
#     [0,1,0], 3:6
#     [0,1],   6:8
#     [1,0,0], 8:11
#     [0,0,1,0], 11:15
#     [0,1,0], 15:18
#     [0,1],   18:20
#     [0,0,1], 20:23
#     [0,1],   23:25
#     [1]      25
# ]

traff_sig_1d = [1,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,1,1]

traff_sig = mp.Array('i', traff_sig_1d)

road_seg_A = deque([0 for _ in range(2)])
road_seg = [deque([0 for _ in range(30)])] * 25
car_count = mp.Value('i', 30)

graph = {
    0: { 1: [road_seg_A, 0]},
    1: { 0: [road_seg[0], 25], 2: [road_seg[2], 3], 4: [road_seg[1], 10]},
    2: { 1: [road_seg[3], 2], 5: [road_seg[4], 14], 3: [road_seg[5], 6]},
    3: { 2: [road_seg[6], 5], 6: [road_seg[7], 17]},
    4: { 1: [road_seg[8], 1], 5: [road_seg[9], 11], 7: [road_seg[10], 19]},
    5: { 2: [road_seg[11], 4], 4: [road_seg[14], 9], 6: [road_seg[12], 15]
, 8: [road_seg[13], 22]},
    6: { 3: [road_seg[16], 7], 5: [road_seg[15], 13], 9: [road_seg[17], 24
]},
    7: { 4: [road_seg[18], 8], 8: [road_seg[19], 20]},
```

```
    8: { 5: [road_seg[21], 12], 7: [road_seg[20], 18], 9: [road_seg[23], 2
3]},
    9: { 6: [road_seg[23], 16], 8: [road_seg[24], 21]}
}


#pathAF[slot]
m =  3
mp_pathAF = mp.Array(c.c_double, m) # shared, can be used from multiple pr
ocesses
# then in each new process create a new numpy array using:
b = np.frombuffer(mp_pathAF.get_obj())
pathAF = b.reshape(m)

#pathAB[slot]
q = 3
mp_pathAB = mp.Array(c.c_double, q) # shared, can be used from multiple pr
ocesses
# then in each new process create a new numpy array using:
a = np.frombuffer(mp_pathAB.get_obj())
pathAB = a.reshape(q)

#pathF[path_segment][slot]
r, s = 13, 30
mp_pathF = mp.Array(c.c_double, r*s) # shared, can be used from multiple p
rocesses
# then in each new process create a new numpy array using:
e = np.frombuffer(mp_pathF.get_obj())
pathF = e.reshape((r,s))

#pathb[path_segment][slot]
t,u  = 13, 30
mp_pathB = mp.Array(c.c_double,t*u) # shared, can be used from multiple pr
ocesses
# then in each new process create a new numpy array using:
d = np.frombuffer(mp_pathB.get_obj())
pathB = d.reshape((t,u))

def TrafficUpdate():
  while car_count.value:
    time.sleep(2)
    traff_sig[0:3] = traff_sig[1:3] + traff_sig[0:1]
    traff_sig[3:6] = traff_sig[4:6] + traff_sig[3:4]
    traff_sig[6:8] = traff_sig[7:8] + traff_sig[6:7]
    traff_sig[8:11] = traff_sig[9:11] + traff_sig[8:9]
```

```
    traff_sig[11:15] = traff_sig[12:15] + traff_sig[11:12]
    traff_sig[15:18] = traff_sig[16:18] + traff_sig[15:16]
    traff_sig[18:20] = traff_sig[19:20] + traff_sig[18:19]
    traff_sig[20:23] = traff_sig[21:23] + traff_sig[20:21]
    traff_sig[23:25] = traff_sig[24:25] + traff_sig[23:24]
    traff_sig[25] ^= 1
    print(*traff_sig)
    print(car_count.value)

#vehicle generation and insertion
def GenerateVehicle(id,pnext_num,tl):
  count=0
  count_a=0
  count_b=0
  intersect = tl
  for i in range(0,2):
    while(pathAF[i]!=id):
      if (pathAF[i]==0):
        pathAF[i] = id
        pathAF[i-1]=0
        print("vehicle %d is in pathAF and slot %d" %(id,i))
      else:
        count=count+1
        if(count==1):
         print("for vehicle %d Slot %d of pathAF is full"%(id,i))
      time.sleep(2)
    if(i == 1):
      while(pathF[pnext_num][0]!=id):
        if (traff_sig[intersect]==1):
          if(pathF[pnext_num][0]==0):
            pathF[pnext_num][0]=id
            pathAF[1]=0
            print("vehicle %d is in pathF %d and slot 0" %(id,pnext_num))
          else:
            count_a=count_a+1
            if(count_a==1):
             print("for vehicle %d signal is green but Slot %d of pathF %d
 is full"%(id,0,pnext_num))
        else:
          count_b=count_b+1
          if(count_b==1):
           print("for vehicle %d to move to next pathF %d signal is red"%(
id,pnext_num))
        time.sleep(2)
```

```
#vehicle returning to A '''
def ReturnVehicle(id,p_num,tl):
   global car_count
   intersect = tl
   count=0
   for i in range (1,30):
      while(pathB[p_num][i]!=id):
        if (pathB[p_num][i])==0:
            pathB[p_num][i]=id
            pathB[p_num][i-1]=0
            print("vehicle %d is in pathB %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathB %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathAB[0]!=id):
            if(traff_sig[intersect]==1):
                if(pathAB[0]==0):
                     pathAB[0]=id
                     pathB[p_num][29]=0
                     print("vehicle %d is in pathAB and slot 0" %(id))
                else:
                  count=count+1
                  if(count==1):
                     print("for vehicle %d signal is green but next pathAB
  slot %d is not empty"%(id,0))
            else:
              count=count+1
              if(count==1):
               print("for vehicle %d to move to next pathAB %d signal is r
ed"%(id, 25) )
            time.sleep(2)
   if(pathAB[0]==id):
     while(pathAB[1]!=id):
      if(pathAB[1]==0):
        pathAB[1]=id;
        pathAB[0]=0;
        print("vehicle %d is in pathAB  and slot 1" %(id))
      else:
        count=count+1
        if(count==1):
         print("for vehicle %d next pathAB slot %d is not empty"%(id,1))
```

```
      time.sleep(2)
   if(pathAB[1]==id):
        pathAB[1]=0
        print("vehicle %d returned to A covering B C and D" %(id))
        with car_count.get_lock():
          car_count.value -= 1



def TravF(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
      while(pathF[p_num][i]!=id):
        if (pathF[p_num][i])==0:
            pathF[p_num][i]=id
            pathF[p_num][i-1]=0
            print("vehicle %d is in pathF %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathF %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathF[pnext_num][0]!=id):
            if(traff_sig[intersect]==1):
                if(pathF[pnext_num][0]==0):
                     pathF[pnext_num][0]=id
                     pathF[p_num][29]=0
                     print("vehicle %d is in pathF %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathF
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
               print("for vehicle %d to move to next pathF%d signal is red
"%(id,pnext_num) )
            time.sleep(2)
```

```
def TravB(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
      while(pathB[p_num][i]!=id):
        if (pathB[p_num][i])==0:
            pathB[p_num][i]=id
            pathB[p_num][i-1]=0
            print("vehicle %d is in pathB %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathB %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathB[pnext_num][0]!=id):
            if(traff_sig[intersect] ==1):
                if(pathB[pnext_num][0]==0):
                     pathB[pnext_num][0]=id
                     pathB[p_num][29]=0
                     print("vehicle %d is in pathB %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathB
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
               print("for vehicle %d to move to next pathB %d signal is re
d"%(id,pnext_num))
            time.sleep(2)


def TravFtoB(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
```

```
      while(pathF[p_num][i]!=id):
        if (pathF[p_num][i])==0:
            pathF[p_num][i]=id
            pathF[p_num][i-1]=0
            print("vehicle %d is in pathF %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathF %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathB[pnext_num][0]!=id):
            if(traff_sig[intersect] ==1):
                if(pathB[pnext_num][0]==0):
                     pathB[pnext_num][0]=id
                     pathF[p_num][29]=0
                     print("vehicle %d is in pathB %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathB
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
                print("for vehicle %d to move to next pathB %d signal is r
ed"%(id,pnext_num) )
            time.sleep(2)

def TravBtoF(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
      while(pathB[p_num][i]!=id):
        if (pathB[p_num][i])==0:
            pathB[p_num][i]=id
            pathB[p_num][i-1]=0
            print("vehicle %d is in pathB %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
```

```
            print("for vehicle %d next pathB %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathF[pnext_num][0]!=id):
            if(traff_sig[intersect] ==1):
                if(pathF[pnext_num][0]==0):
                     pathF[pnext_num][0]=id
                     pathB[p_num][29]=0
                     print("vehicle %d is in pathF %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathF
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
                print("for vehicle %d to move to next pathF %d signal is r
ed"%(id,pnext_num) )
            time.sleep(2)


def path1_vehicle1():
   GenerateVehicle(1,3,graph[0][1][1])
   TravF(1,3,graph[1][4][1],8)
   TravF(1,8,graph[4][7][1],9)
   TravF(1,9,graph[7][8][1],11)
   TravFtoB(1,11,graph[8][9][1],12)
   TravB(1,12,graph[9][6][1],7)
   TravB(1,7,graph[6][3][1],2)
   TravB(1,2,graph[3][2][1],1)
   ReturnVehicle(1,1,graph[2][1][1])
def path1_vehicle2():
   GenerateVehicle(2,3,graph[0][1][1])
   TravF(2,3,graph[1][4][1],8)
   TravF(2,8,graph[4][7][1],9)
   TravF(2,9,graph[7][8][1],11)
   TravFtoB(2,11,graph[8][9][1],12)
   TravB(2,12,graph[9][6][1],7)
   TravB(2,7,graph[6][3][1],2)
   TravB(2,2,graph[3][2][1],1)
   ReturnVehicle(2,1,graph[2][1][1])
def path1_vehicle3():
```

```
    GenerateVehicle(3,3,graph[0][1][1])
    TravF(3,3,graph[1][4][1],8)
    TravF(3,8,graph[4][7][1],9)
    TravF(3,9,graph[7][8][1],11)
    TravFtoB(3,11,graph[8][9][1],12)
    TravB(3,12,graph[9][6][1],7)
    TravB(3,7,graph[6][3][1],2)
    TravB(3,2,graph[3][2][1],1)
    ReturnVehicle(3,1,graph[2][1][1])
def path2_vehicle4():
    GenerateVehicle(4,1,graph[0][1][1])
    TravF(4,1,graph[1][2][1],5)
    TravF(4,5,graph[2][5][1],10)
    TravF(4,10,graph[5][8][1],11)
    TravFtoB(4,11,graph[8][9][1],12)
    TravB(4,12,graph[9][6][1],7)
    TravB(4,7,graph[6][3][1],2)
    TravBtoF(4,2,graph[3][2][1],5)
    TravF(4,5,graph[2][5][1],10)
    TravFtoB(4,10,graph[5][8][1],9)
    TravB(4,9,graph[8][7][1],8)
    TravB(4,8,graph[7][4][1],3)
    ReturnVehicle(4,3,graph[4][1][1])
def path2_vehicle5():
    GenerateVehicle(5,1,graph[0][1][1])
    TravF(5,1,graph[1][2][1],5)
    TravF(5,5,graph[2][5][1],10)
    TravF(5,10,graph[5][8][1],11)
    TravFtoB(5,11,graph[8][9][1],12)
    TravB(5,12,graph[9][6][1],7)
    TravB(5,7,graph[6][3][1],2)
    TravBtoF(5,2,graph[3][2][1],5)
    TravF(5,5,graph[2][5][1],10)
    TravFtoB(4,10,graph[5][8][1],9)
    TravB(5,9,graph[8][7][1],8)
    TravB(5,8,graph[7][4][1],3)
    ReturnVehicle(5,3,graph[4][1][1])
def path2_vehicle6():
    GenerateVehicle(6,1,graph[0][1][1])
    TravF(6,1,graph[1][2][1],5)
    TravF(6,5,graph[2][5][1],10)
    TravF(6,10,graph[5][8][1],11)
    TravFtoB(6,11,graph[8][9][1],12)
    TravB(6,12,graph[9][6][1],7)
    TravB(6,7,graph[6][3][1],2)
```

```
    TravBtoF(6,2,graph[3][2][1],5)
    TravF(6,5,graph[2][5][1],10)
    TravFtoB(6,10,graph[5][8][1],9)
    TravB(6,9,graph[8][7][1],8)
    TravB(6,8,graph[7][4][1],3)
    ReturnVehicle(6,3,graph[4][1][1])
def path3_vehicle7():
  GenerateVehicle(7,1,graph[0][1][1])
  TravF(7,1,graph[1][2][1],2)
  TravF(7,2,graph[2][3][1],7)
  TravF(7,7,graph[3][6][1],12)
  TravFtoB(7,12,graph[6][9][1],11)
  TravB(7,11,graph[9][8][1],9)
  TravB(7,9,graph[8][7][1],8)
  TravB(7,8,graph[7][4][1],3)
  ReturnVehicle(7,3,graph[4][1][1])
def path3_vehicle8():
  GenerateVehicle(8,1,graph[0][1][1])
  TravF(8,1,graph[1][2][1],2)
  TravF(8,2,graph[2][3][1],7)
  TravF(8,7,graph[3][6][1],12)
  TravFtoB(8,12,graph[6][9][1],11)
  TravB(8,11,graph[9][8][1],9)
  TravB(8,9,graph[8][7][1],8)
  TravB(8,8,graph[7][4][1],3)
  ReturnVehicle(8,3,graph[4][1][1])
def path3_vehicle9():
  GenerateVehicle(9,1,graph[0][1][1])
  TravF(9,1,graph[1][2][1],2)
  TravF(9,2,graph[2][3][1],7)
  TravF(9,7,graph[3][6][1],12)
  TravFtoB(9,12,graph[6][9][1],11)
  TravB(9,11,graph[9][8][1],9)
  TravB(9,9,graph[8][7][1],8)
  TravB(9,8,graph[7][4][1],3)
  ReturnVehicle(9,3,graph[4][1][1])
def path4_vehicle10():
  GenerateVehicle(10,3,graph[0][1][1])
  TravF(10,3,graph[1][4][1],4)
  TravF(10,4,graph[4][5][1],6)
  TravFtoB(10,6,graph[5][6][1],12)
  TravB(10,12,graph[6][9][1],11)
  TravB(10,11,graph[9][8][1],9)
  TravB(10,9,graph[8][7][1],8)
  TravBtoF(10,8,graph[7][4][1],4)
```

```
  TravF(10,4,graph[4][5][1],6)
  TravFtoB(10,6,graph[5][6][1],7)
  TravB(10,7,graph[6][3][1],2)
  TravB(10,2,graph[3][2][1],1)
  ReturnVehicle(10,1,graph[2][1][1])
def path4_vehicle11():
  GenerateVehicle(11,3,graph[0][1][1])
  TravF(11,3,graph[1][4][1],4)
  TravF(11,4,graph[4][5][1],6)
  TravFtoB(11,6,graph[5][6][1],12)
  TravB(11,12,graph[6][9][1],11)
  TravB(11,11,graph[9][8][1],9)
  TravB(11,9,graph[8][7][1],8)
  TravBtoF(11,8,graph[7][4][1],4)
  TravF(11,4,graph[4][5][1],6)
  TravFtoB(11,6,graph[5][6][1],7)
  TravB(11,7,graph[6][3][1],2)
  TravB(11,2,graph[3][2][1],1)
  ReturnVehicle(11,1,graph[2][1][1])
def path4_vehicle12():
  GenerateVehicle(12,3,graph[0][1][1])
  TravF(12,3,graph[1][4][1],4)
  TravF(12,4,graph[4][5][1],6)
  TravFtoB(12,6,graph[5][6][1],12)
  TravB(12,12,graph[6][9][1],11)
  TravB(12,11,graph[9][8][1],9)
  TravB(12,9,graph[8][7][1],8)
  TravBtoF(12,8,graph[7][4][1],4)
  TravF(12,4,graph[4][5][1],6)
  TravFtoB(12,6,graph[5][6][1],7)
  TravB(12,7,graph[6][3][1],2)
  TravB(12,2,graph[3][2][1],1)
  ReturnVehicle(12,1,graph[2][1][1])
def path5_vehicle13():
  GenerateVehicle(13,3,graph[0][1][1])
  TravF(13,3,graph[1][4][1],8)
  TravF(13,8,graph[4][7][1],9)
  TravFtoB(13,9,graph[7][8][1],10)
  TravB(13,10,graph[8][5][1],5)
  TravBtoF(13,5,graph[5][2][1],2)
  TravF(13,2,graph[2][3][1],7)
  TravF(13,7,graph[3][6][1],12)
  TravFtoB(13,12,graph[6][9][1],11)
  TravB(13,11,graph[9][8][1],9)
  TravB(13,9,graph[8][5][1],8)
```

```
  TravB(13,8,graph[5][2][1],3)
  ReturnVehicle(13,3,graph[2][1][1])
def path5_vehicle14():
  GenerateVehicle(14,3,graph[0][1][1])
  TravF(14,3,graph[1][4][1],8)
  TravF(14,8,graph[4][7][1],9)
  TravFtoB(14,9,graph[7][8][1],10)
  TravB(14,10,graph[8][5][1],5)
  TravBtoF(14,5,graph[5][2][1],2)
  TravF(14,2,graph[2][3][1],7)
  TravF(14,7,graph[3][6][1],12)
  TravFtoB(14,12,graph[6][9][1],11)
  TravB(14,11,graph[9][8][1],9)
  TravB(14,9,graph[8][5][1],8)
  TravB(14,8,graph[5][2][1],3)
  ReturnVehicle(14,3,graph[2][1][1])
def path5_vehicle15():
  GenerateVehicle(15,3,graph[0][1][1])
  TravF(15,3,graph[1][4][1],8)
  TravF(15,8,graph[4][7][1],9)
  TravFtoB(15,9,graph[7][8][1],10)
  TravB(15,10,graph[8][5][1],5)
  TravBtoF(15,5,graph[5][2][1],2)
  TravF(15,2,graph[2][3][1],7)
  TravF(15,7,graph[3][6][1],12)
  TravFtoB(15,12,graph[6][9][1],11)
  TravB(15,11,graph[9][8][1],9)
  TravB(15,9,graph[8][5][1],8)
  TravB(15,8,graph[5][2][1],3)
  ReturnVehicle(15,3,graph[2][1][1])

def path1_vehicle16():
   GenerateVehicle(16,3,graph[0][1][1])
   TravF(16,3,graph[1][4][1],8)
   TravF(16,8,graph[4][7][1],9)
   TravF(16,9,graph[7][8][1],11)
   TravFtoB(16,11,graph[8][9][1],12)
   TravB(16,12,graph[9][6][1],7)
   TravB(16,7,graph[6][3][1],2)
   TravB(16,2,graph[3][2][1],1)
   ReturnVehicle(16,1,graph[2][1][1])
def path1_vehicle17():
   GenerateVehicle(17,3,graph[0][1][1])
   TravF(17,3,graph[1][4][1],8)
   TravF(17,8,graph[4][7][1],9)
```

```
    TravF(17,9,graph[7][8][1],11)
    TravFtoB(17,11,graph[8][9][1],12)
    TravB(17,12,graph[9][6][1],7)
    TravB(17,7,graph[6][3][1],2)
    TravB(17,2,graph[3][2][1],1)
    ReturnVehicle(17,1,graph[2][1][1])
def path1_vehicle18():
    GenerateVehicle(18,3,graph[0][1][1])
    TravF(18,3,graph[1][4][1],8)
    TravF(18,8,graph[4][7][1],9)
    TravF(18,9,graph[7][8][1],11)
    TravFtoB(18,11,graph[8][9][1],12)
    TravB(18,12,graph[9][6][1],7)
    TravB(18,7,graph[6][3][1],2)
    TravB(18,2,graph[3][2][1],1)
    ReturnVehicle(18,1,graph[2][1][1])
def path2_vehicle19():
    GenerateVehicle(19,1,graph[0][1][1])
    TravF(19,1,graph[1][2][1],5)
    TravF(19,5,graph[2][5][1],10)
    TravF(19,10,graph[5][8][1],11)
    TravFtoB(19,11,graph[8][9][1],12)
    TravB(19,12,graph[9][6][1],7)
    TravB(19,7,graph[6][3][1],2)
    TravBtoF(19,2,graph[3][2][1],5)
    TravF(19,5,graph[2][5][1],10)
    TravFtoB(19,10,graph[5][8][1],9)
    TravB(19,9,graph[8][7][1],8)
    TravB(19,8,graph[7][4][1],3)
    ReturnVehicle(19,3,graph[4][1][1])
def path2_vehicle20():
    GenerateVehicle(20,1,graph[0][1][1])
    TravF(20,1,graph[1][2][1],5)
    TravF(20,5,graph[2][5][1],10)
    TravF(20,10,graph[5][8][1],11)
    TravFtoB(20,11,graph[8][9][1],12)
    TravB(20,12,graph[9][6][1],7)
    TravB(20,7,graph[6][3][1],2)
    TravBtoF(20,2,graph[3][2][1],5)
    TravF(20,5,graph[2][5][1],10)
    TravFtoB(20,10,graph[5][8][1],9)
    TravB(20,9,graph[8][7][1],8)
    TravB(20,8,graph[7][4][1],3)
    ReturnVehicle(20,3,graph[4][1][1])
def path2_vehicle21():
```

```
    GenerateVehicle(21,1,graph[0][1][1])
    TravF(21,1,graph[1][2][1],5)
    TravF(21,5,graph[2][5][1],10)
    TravF(21,10,graph[5][8][1],11)
    TravFtoB(21,11,graph[8][9][1],12)
    TravB(21,12,graph[9][6][1],7)
    TravB(21,7,graph[6][3][1],2)
    TravBtoF(21,2,graph[3][2][1],5)
    TravF(21,5,graph[2][5][1],10)
    TravFtoB(21,10,graph[5][8][1],9)
    TravB(21,9,graph[8][7][1],8)
    TravB(21,8,graph[7][4][1],3)
    ReturnVehicle(21,3,graph[4][1][1])
def path3_vehicle22():
  GenerateVehicle(22,1,graph[0][1][1])
  TravF(22,1,graph[1][2][1],2)
  TravF(22,2,graph[2][3][1],7)
  TravF(22,7,graph[3][6][1],12)
  TravFtoB(22,12,graph[6][9][1],11)
  TravB(22,11,graph[9][8][1],9)
  TravB(22,9,graph[8][7][1],8)
  TravB(22,8,graph[7][4][1],3)
  ReturnVehicle(22,3,graph[4][1][1])
def path3_vehicle23():
  GenerateVehicle(23,1,graph[0][1][1])
  TravF(23,1,graph[1][2][1],2)
  TravF(23,2,graph[2][3][1],7)
  TravF(23,7,graph[3][6][1],12)
  TravFtoB(23,12,graph[6][9][1],11)
  TravB(23,11,graph[9][8][1],9)
  TravB(23,9,graph[8][7][1],8)
  TravB(23,8,graph[7][4][1],3)
  ReturnVehicle(23,3,graph[4][1][1])
def path3_vehicle24():
  GenerateVehicle(24,1,graph[0][1][1])
  TravF(24,1,graph[1][2][1],2)
  TravF(24,2,graph[2][3][1],7)
  TravF(24,7,graph[3][6][1],12)
  TravFtoB(24,12,graph[6][9][1],11)
  TravB(24,11,graph[9][8][1],9)
  TravB(24,9,graph[8][7][1],8)
  TravB(24,8,graph[7][4][1],3)
  ReturnVehicle(24,3,graph[4][1][1])
def path4_vehicle25():
  GenerateVehicle(25,3,graph[0][1][1])
```

```
    TravF(25,3,graph[1][4][1],4)
    TravF(25,4,graph[4][5][1],6)
    TravFtoB(25,6,graph[5][6][1],12)
    TravB(25,12,graph[6][9][1],11)
    TravB(25,11,graph[9][8][1],9)
    TravB(25,9,graph[8][7][1],8)
    TravBtoF(25,8,graph[7][4][1],4)
    TravF(25,4,graph[4][5][1],6)
    TravFtoB(25,6,graph[5][6][1],7)
    TravB(25,7,graph[6][3][1],2)
    TravB(25,2,graph[3][2][1],1)
    ReturnVehicle(25,1,graph[2][1][1])
  def path4_vehicle26():
    GenerateVehicle(26,3,graph[0][1][1])
    TravF(26,3,graph[1][4][1],4)
    TravF(26,4,graph[4][5][1],6)
    TravFtoB(26,6,graph[5][6][1],12)
    TravB(26,12,graph[6][9][1],11)
    TravB(26,11,graph[9][8][1],9)
    TravB(26,9,graph[8][7][1],8)
    TravBtoF(26,8,graph[7][4][1],4)
    TravF(26,4,graph[4][5][1],6)
    TravFtoB(26,6,graph[5][6][1],7)
    TravB(26,7,graph[6][3][1],2)
    TravB(26,2,graph[3][2][1],1)
    ReturnVehicle(26,1,graph[2][1][1])
  def path4_vehicle27():
    GenerateVehicle(27,3,graph[0][1][1])
    TravF(27,3,graph[1][4][1],4)
    TravF(27,4,graph[4][5][1],6)
    TravFtoB(27,6,graph[5][6][1],12)
    TravB(27,12,graph[6][9][1],11)
    TravB(27,11,graph[9][8][1],9)
    TravB(27,9,graph[8][7][1],8)
    TravBtoF(27,8,graph[7][4][1],4)
    TravF(27,4,graph[4][5][1],6)
    TravFtoB(27,6,graph[5][6][1],7)
    TravB(27,7,graph[6][3][1],2)
    TravB(27,2,graph[3][2][1],1)
    ReturnVehicle(27,1,graph[2][1][1])
  def path5_vehicle28():
    GenerateVehicle(28,3,graph[0][1][1])
    TravF(28,3,graph[1][4][1],8)
    TravF(28,8,graph[4][7][1],9)
    TravFtoB(28,9,graph[7][8][1],10)
```

```
    TravB(28,10,graph[8][5][1],5)
    TravBtoF(28,5,graph[5][2][1],2)
    TravF(28,2,graph[2][3][1],7)
    TravF(28,7,graph[3][6][1],12)
    TravFtoB(28,12,graph[6][9][1],11)
    TravB(28,11,graph[9][8][1],9)
    TravB(28,9,graph[8][5][1],8)
    TravB(28,8,graph[5][2][1],3)
    ReturnVehicle(28,3,graph[2][1][1])
  def path5_vehicle29():
    GenerateVehicle(29,3,graph[0][1][1])
    TravF(29,3,graph[1][4][1],8)
    TravF(29,8,graph[4][7][1],9)
    TravFtoB(29,9,graph[7][8][1],10)
    TravB(29,10,graph[8][5][1],5)
    TravBtoF(29,5,graph[5][2][1],2)
    TravF(29,2,graph[2][3][1],7)
    TravF(29,7,graph[3][6][1],12)
    TravFtoB(29,12,graph[6][9][1],11)
    TravB(29,11,graph[9][8][1],9)
    TravB(29,9,graph[8][5][1],8)
    TravB(29,8,graph[5][2][1],3)
    ReturnVehicle(29,3,graph[2][1][1])
  def path5_vehicle30():
    GenerateVehicle(30,3,graph[0][1][1])
    TravF(30,3,graph[1][4][1],8)
    TravF(30,8,graph[4][7][1],9)
    TravFtoB(30,9,graph[7][8][1],10)
    TravB(30,10,graph[8][5][1],5)
    TravBtoF(30,5,graph[5][2][1],2)
    TravF(30,2,graph[2][3][1],7)
    TravF(30,7,graph[3][6][1],12)
    TravFtoB(30,12,graph[6][9][1],11)
    TravB(30,11,graph[9][8][1],9)
    TravB(30,9,graph[8][5][1],8)
    TravB(30,8,graph[5][2][1],3)
    ReturnVehicle(30,3,graph[2][1][1])



traff = mp.Process(target=TrafficUpdate)
p1=mp.Process(target=path1_vehicle1)
p2=mp.Process(target=path1_vehicle2)
```

```
p3=mp.Process(target=path1_vehicle3)
p4=mp.Process(target=path2_vehicle4)
p5=mp.Process(target=path2_vehicle5)
p6=mp.Process(target=path2_vehicle6)
p7=mp.Process(target=path3_vehicle7)
p8=mp.Process(target=path3_vehicle8)
p9=mp.Process(target=path3_vehicle9)
p10=mp.Process(target=path4_vehicle10)
p11=mp.Process(target=path4_vehicle11)
p12=mp.Process(target=path4_vehicle12)
p13=mp.Process(target=path5_vehicle13)
p14=mp.Process(target=path5_vehicle14)
p15=mp.Process(target=path5_vehicle15)
p16=mp.Process(target=path1_vehicle16)
p17=mp.Process(target=path1_vehicle17)
p18=mp.Process(target=path1_vehicle18)
p19=mp.Process(target=path2_vehicle19)
p20=mp.Process(target=path2_vehicle20)
p21=mp.Process(target=path2_vehicle21)
p22=mp.Process(target=path3_vehicle22)
p23=mp.Process(target=path3_vehicle23)
p24=mp.Process(target=path3_vehicle24)
p25=mp.Process(target=path4_vehicle25)
p26=mp.Process(target=path4_vehicle26)
p27=mp.Process(target=path4_vehicle27)
p28=mp.Process(target=path5_vehicle28)
p29=mp.Process(target=path5_vehicle29)
p30=mp.Process(target=path5_vehicle30)

traff.start()
p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()
p7.start()
p8.start()
p9.start()
p10.start()
p11.start()
p12.start()
p13.start()
p14.start()
p15.start()
```

```
p16.start()
p17.start()
p18.start()
p19.start()
p20.start()
p21.start()
p22.start()
p23.start()
p24.start()
p25.start()
p26.start()
p27.start()
p28.start()
p29.start()
p30.start()

traff.join()
p1.join()
p2.join()
p3.join()
p4.join()
p5.join()
p6.join()
p7.join()
p8.join()
p9.join()
p10.join()
p11.join()
p12.join()
p13.join()
p14.join()
p15.join()
p16.join()
p17.join()
p18.join()
p19.join()
p20.join()
p21.join()
p22.join()
p23.join()
p24.join()
p25.join()
p26.join()
p27.join()
p28.join()
```

```
p29.join()
p30.join()


finish=time.perf_counter()

print(f'Finished in {round(finish-start,2)} second(s)')
```

ECEN689-602/CSCE689-603 Introduction to Formal Verification Fall 2021

Project Report Phase C

Submitted to the Faculty

Of

Texas A&M University

By

Team 6 I-Group Integration

Brent Arnold Basiano UIN: 130004869

Sri Hari Pada Kodi UIN: 932003040

Under the Guidance of

Professor Jiang Hu

Department of Electrical and Computer Engineering

November 2021

TABLE OF CONTENTS

LIST OF FIGURES

ECEN 689 - Introduction to Formal Verification

Phase C

Team 6 - I-Group

ABSTRACT:

Phase C is about verifying the properties of road grid with traffic system that we have designed in the Phase A. Here we have to verify the properties using the NuSMv model checker tool which the V-Group found out over the Phase B. Here the I-group is responsible for verifying the vehicle properties. There are in total 5 properties out of which 4 are pre-defined and 1 property is of our choice.

The properties that we, the I-group have to verify are as follows:

1. No vehicle takes any U-turn.
2. No vehicle enters an intersection at red light.
3. Every vehicle must visit all of points B, C and D.
4. There is no collision between any two vehicles.
5. Another property at your own choice.

BACKGROUND:

The project is to design a road system and verify its properties. All cars start in a common point in the road called point A which moves to other points labeled B, C, and D. The car must go to all the points in any order but must return to point A. For this project, two groups in a team deal with the road (I-Group) and vehicles (V-Group) separately. The project is also broken down to three phases. This report focuses on Phases C.

For phase C, the I-Group focused on verification of the vehicle properties using NuSMv model checking tool. We have applied the concept of modeling a concurrent system for the road grid with traffic system and obtained a transition system to verify the properties such as safety, mutual exclusion, liveliness et cetera that are defined above as vehicle properties. We have considered V-group system in the Phase A and thus verifying the vehicle properties.

PROCEDURES :

The system is a 3*3 road grid with traffic signal, with 4-two way intersections, 4-three way intersections, and 1-four way intersection. On a whole, a vehicle starts from the point labeled A and travels through the points B, C, and D and then returns to A in any order. Here we are considering the paths defined by V-group in the phase A to verify the vehicle group properties. There are total 30 slots between each segment, here segment means the path between 2 nodes (nodes are represented by circles in the figure 1).

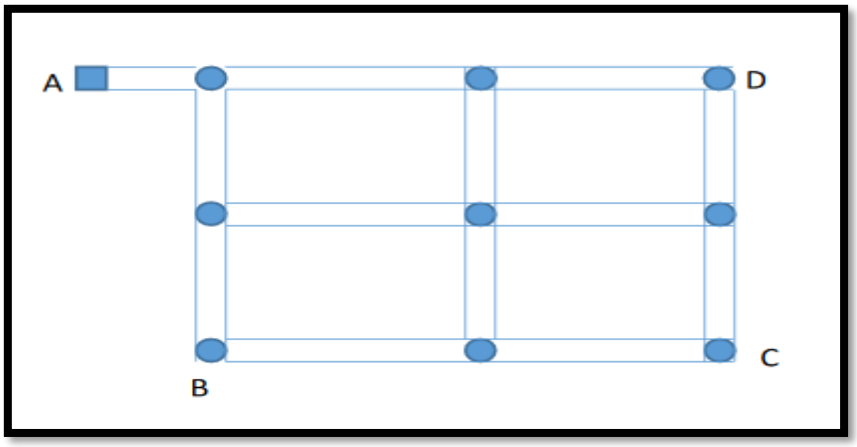

Figure 1. Road System

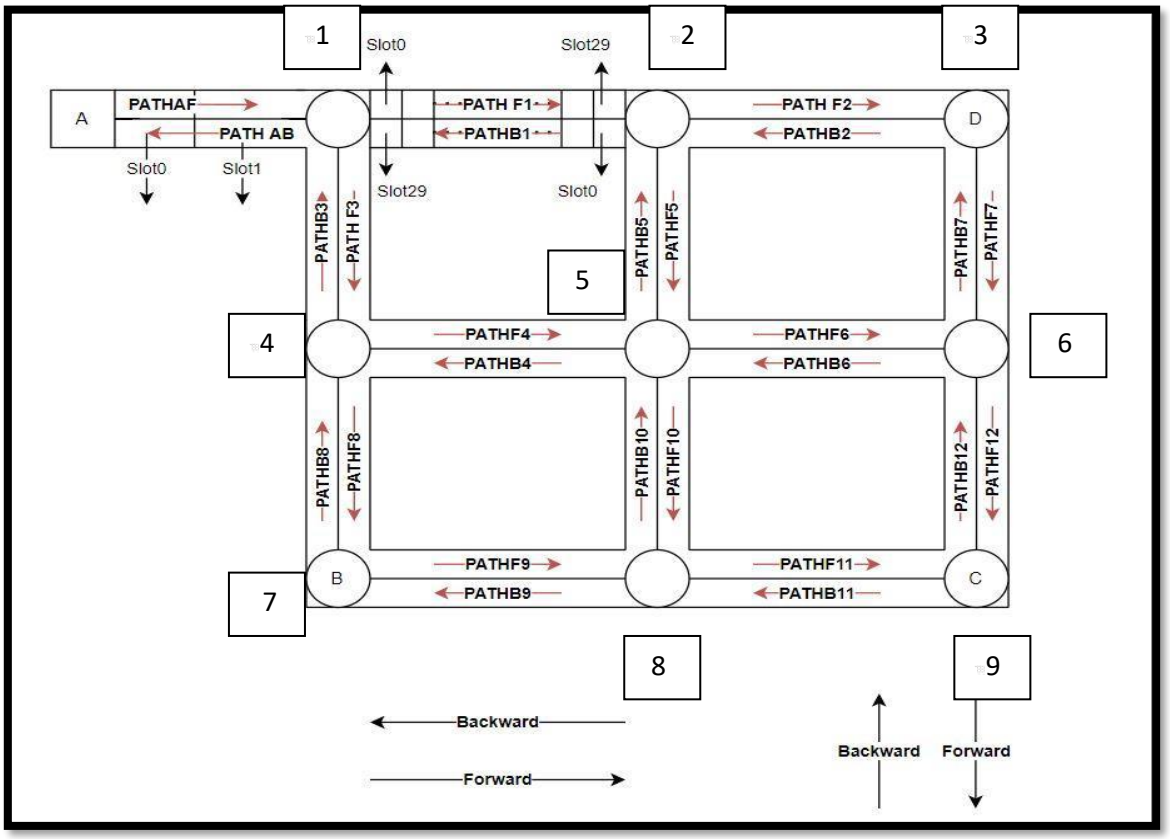

Figure 2: Directed Graph of the Road System

The vehicle movement in the grid showed in Figure 1 is implemented by the concept of directed graph with constraints such as PATH F1 (Forward path), PATH B1 (Backward path), slots (0 to 29) et cetera (refer to figure 2).

PROPERTY VERIFICATION:

1. NO VEHCILE TAKES ANY U-TURN:

A vehicle at any instant of time and position should not take a u-turn in the 3*3 road grid with traffic signal system.

Contextual explanation: Consider a vehicle is travelling in forward (ex: PATH F1) direction at any instant of time, either the vehicle should be moving in the same direction or it should stay in slot (i.e., should not move). It should not shift its direction and go in the reverse path (ex: PATH B1). This scenario can be better understood by a simple animated figure shown below.

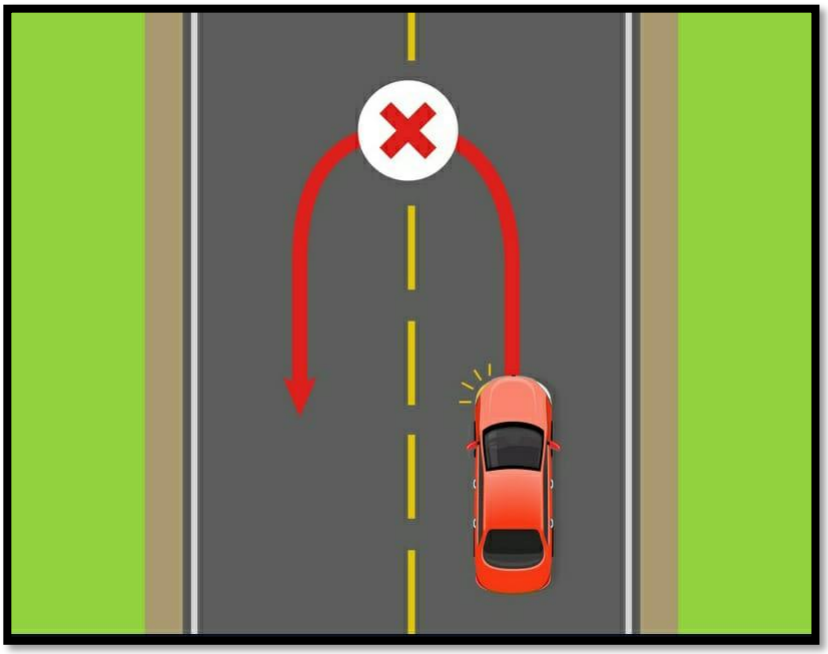

Figure 3: Animation showing there should be no u-turn

- Here the property of vehicle taking a u-turn is considered as something bad that should never happen for this system, hence this signifies the "**Safety**" property for the vehicle.

A simple state system explaining the property of **u-turn** not happening, that we are supposed to verify is as follows:

- Here we are considering the slots as transitions for better understanding.

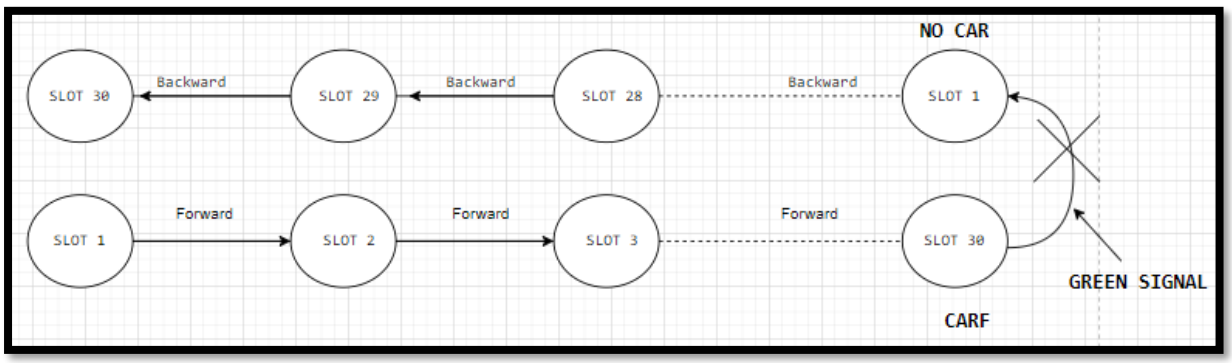

Figure 4: state system representation for property u-turn

Verification of state system explanation (theoretical):

Consider the segment between point 1 and 2 (from the Figure 2), here the vehicle is moving from slot 0 to slot 29. There are 2 possibilities for the u-turn to happen

a. One among them is, at the 29th slot (represented by SLOT 30 in Figure 4) if the signal is green and there is a vehicle at the first slot (**represented by SLOT 1 in Figure 4**) of backward path. There should be no u-turn, otherwise the collision takes place in the next state. This scenario of collision is covered in the 4th property.
b. The second scenario is, when the car in forward path is at the location of 29th slot (represented by SLOT 3 in Figure 4), the signal is green and there is no car in the first slot (represented by SLOT 4 in Figure 4) of the backward path. There should be no u-turn.

Verification of state system explanation (practical) using NUSMV model checking tool:

Using NUSMV we modeled the 3*3 road grid with traffic signal. For checking the **no vehicle turns u-turn** property here we are considering the instance from the above theoretical explanation i.e. segment between point 1 and 2 (from the Figure 2).

Using the **Linear Temporal Logic** concept we verified for the **Global** specification using the below condition. In the figure 5, **check_ltlspec** refers to checking Linear Temporal Logic, G refers to Global condition i.e., satisfies for each and every transition of the system. **F1** – Forward path, **B1** – Backward path, **L1** – traffic signal, **carF** – car in forward path, **no_car** – no car present in the slot.

```
NuSMV > go
NuSMV > check_ltlspec
-- specification  G (((F1 = carF & L1 = green) & B1 = no_car) -> !(B1 = carF))  is true
```

Figure 5: Output in NuSMV showing LTL spec verification for "G" is true for no u-turn

NuSMV model checking tool output explanation:

Here the output signifies that, for each and every transition when a car is moving in F1 direction and has green light i.e. L1 at the junction, and with empty slot at B1 direction. The next state of the car will never be B1 i.e. the car never takes a u-turn (i.e.! (B1 = carF)).

Symbols meanings:

a) **&** means **and**
b) **->** means **next state**
c) **!** means **negate** or **not**
d) **G** means **Global**

- Therefore the requirement of **no vehicle takes any u-turn** is satisfied globally (G) using the linear temporal logic (check_ltlspec) for the 3*3 road grid with traffic signal.

2. NO VEHICLE ENTERS AN INTERSECTION AT RED LIGHT.

A vehicle at any intersection in the 3*3 road grid with traffic signal system should not violate the red light.

Contextual explanation: Consider there are 4 vehicles at the 29^{th} slot of each segment at an intersection from all the directions. In this scenario only one vehicle should be allowed to move, rest all vehicles should not move i.e. at most 1 signal should be green at any intersection. This scenario is valid only when the vehicle which is having a red light in its path should not move.

Below is an animated figure of a 4-way intersection describing the situation of above scenario.

Figure 6: A 4-way intersection for visual understanding

- This situation is exhibiting "**safety**" property for vehicle which at any point of time in the 3*3 road grid with traffic signal does not break the red signal.

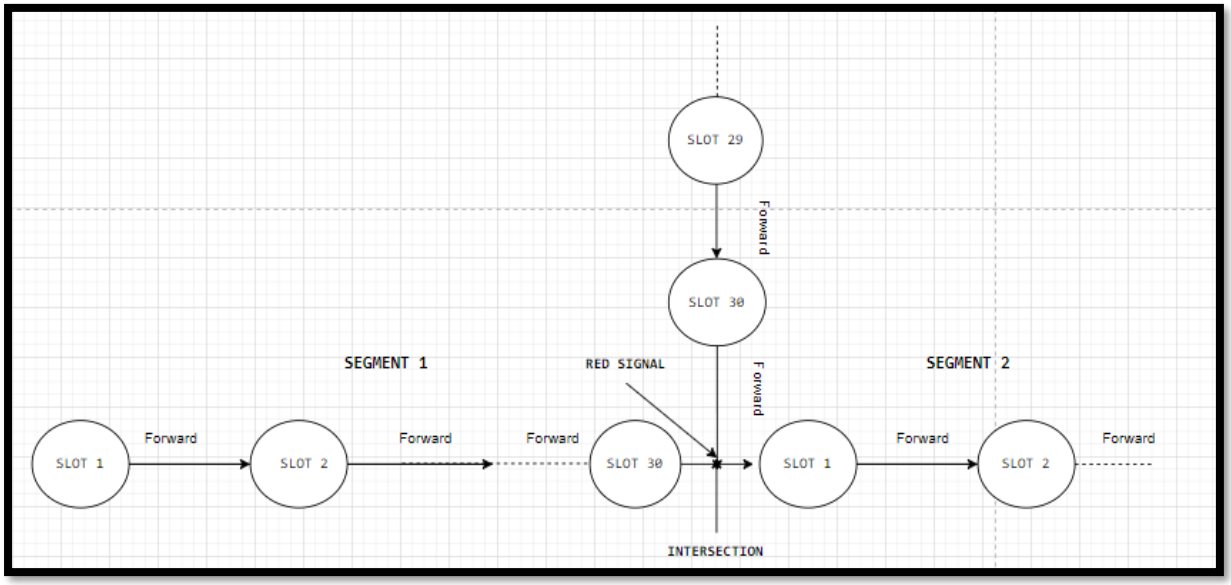

Figure 7: state system representation for property no vehicle at intersection while red light

Verification of state system explanation (theoretical):

Here let us consider a vehicle is at a 2-way intersection, the condition which is followed to satisfy our property is 'at most only 1 signal at an intersection is green'. Therefore when a car has red signal in its direction, it should stop even when there is no vehicle in the next slot or state. Refer to the above figure (Figure 7) for better understanding.

Verification of state system explanation (practical) using NUSMV model checking tool:

Using NUSMV we modeled the 3*3 road grid with traffic signal. For checking the **no vehicle entering an intersection at red signal property**, here we are considering the instance from the

above theoretical explanation i.e. vehicle traveling from SEGMENT 1 to SEGMENT 2 at a 2-way intersection (from the Figure 7). To verify the result we are also showing the signal property at the 2-way intersection for better understanding.

Using the **Linear Temporal Logic** concept we verified for the **Global** specification using the below condition. In the figure 8 & 9, **check_ltlspec** refers to checking Linear Temporal Logic, G refers to Global condition i.e., satisfies for each and every transition of the system.

Here the below output of NuSMV model checking tool shows that both the signals cannot be red at the same time. This means the property of "at most 1 signal green is obeyed here".

```
NuSMV > check_ltlspec -p "G (!(L1 = green) | !(L2 = green))
ignoring unbalanced quote ...
-- specification  G (!(L1 = green) | !(L2 = green))  is true
NuSMV >
```

Figure 8: Output in NuSMV showing LTL spec verification for "G" is true for signals at 2-way intersection

Now since the signal property stands true, we are verifying the property of **No vehicle enters an intersection at red light.**

```
NuSMV > go
NuSMV > check_ltlspec
-- specification  G (((F1 = carF & L1 = red) & (F2 = carF | F2 = no car)) -> !(F1 = no car))  is true
```

Figure 9: Output in NuSMV showing LTL spec verification for "G" is true for vehicle enters an intersection at red light

NuSMV model checking tool output explanation:

Here the output signifies that, for each and every transition when a vehicle **carF** is at the **29th** slot of the **first segment F1** and the **signal L1** is **red** and at the **second segment**, whether the slot 1 **F2** is having a **car** or **no car**, the vehicle **carF** will not go the slot 1 or next state (i.e. F1 = car means car is present at F1).

Symbols meanings:

a) **&** means **and**
b) **->** means **next state**
c) **!** means **negate** or **not**
d) **G** means **Global**

carF -> car moving in forward path

no car -> no car is present at a slot

F1 -> forward path for segment 1

F2 -> forward path for segment 2

- Therefore the requirement of **No vehicle enters an intersection at red light** is satisfied globally (G) using the linear temporal logic (check_ltlspec) for the 3*3 road grid with traffic signal.

3. **EVERY VEHICLE MUST VISIT ALL OF POINTS B, C and D.**

Any vehicle entering into the 3*3 road grid with traffic signal system should travel covering the points B, C and D.

Contextual explanation: In the phase A of the project, the vehicle group defined 5 paths for a vehicle to travel covering all the points B, C and D. For simplicity we are verifying the system for those properties using the NuSMv model checking tool. Here consider a vehicle is at point 4 (refer to the numbering in the Figure 10) it should cover point B (point 7 in Figure 10), point 9, point C (point 9 in Figure 10), point 6, point D (point 3 in the Figure 10) to verify the property.

Here the vehicle should cover 5 traffic signals and 5 segments at least to cover the points B, C and D and come out. Here we are verifying the system for 2 cases i.e. B->C->D and D->B->C.

- Here the vehicle is travelling across all the 3 points B, C and D. Similarly for D, C and B. Hence the vehicle is satisfying the "**Liveness**" property for the 3*3 road grid with traffic signal system.

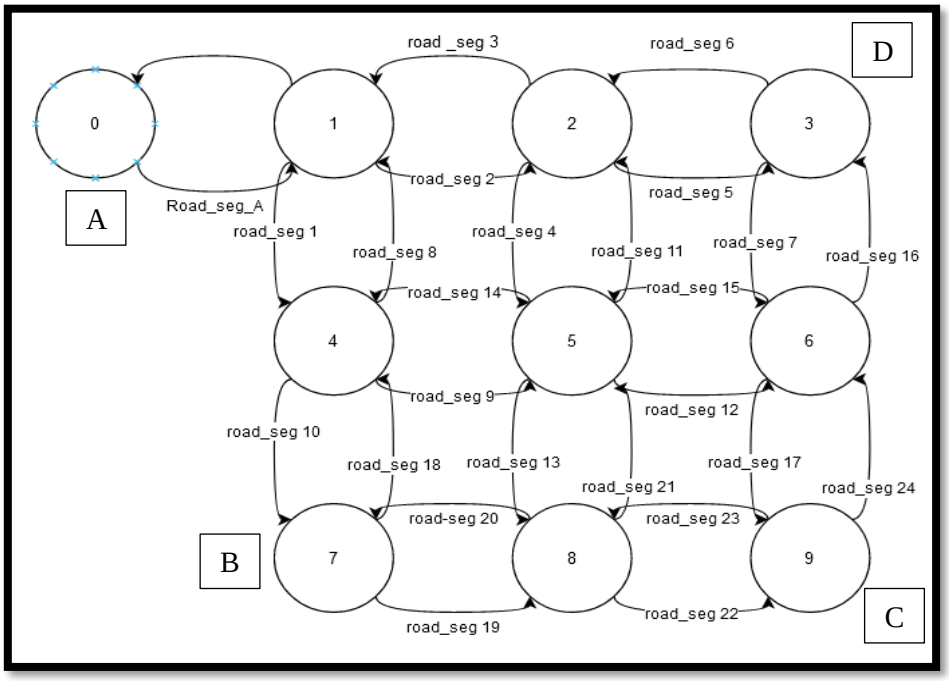

Figure 10: Directed graph of road system with points B, C and D

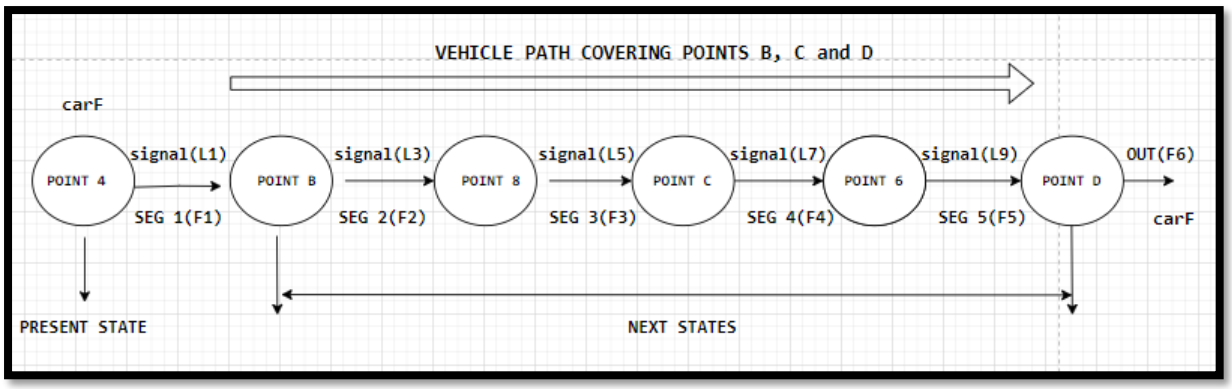

Figure 11: state system for vehicle covering the points B, C and D

Verification of state system explanation (theoretical):

Here let us consider a vehicle is at point 4 in Figure 11 (for clear understanding follow the Figure 10: Directed graph). So, the vehicle should cover point B, point 8, point C, point 6, point D (next states). Condition followed by the vehicle here is the signals in the path should be green at the intersections. The vehicle here goes through 5 segments to cover the points B, C and D (segments are listed in Figure 11). The vehicle follows the same process for the path covering D, C and B points respectively.

Verification of state system explanation (practical) using NUSMV model checking tool:

Using NUSMV we modeled the 3*3 road grid with traffic signal. For checking the **every vehicle must visit all of points B, C and D** property, here we are considering the instance from the above theoretical explanation i.e. vehicle traveling from SEGMENT 1 to SEGMENT 5 and coming out covering B, C and D points.

Using the **Linear Temporal Logic** concept we verified for the **Global** specification using the below condition. In the figure 12, **check_ltlspec** refers to checking Linear Temporal Logic, G refers to Global condition i.e., satisfies for each and every transition of the system.

```
NuSMV > go
NuSMV > check_ltlspec
-- specification  G ((((((L1 = green & L3 = green) & L5 = green) & L7 = green) & L9 = green) & (((((F1 = carF -> F1 = no_car) & (F2 = carF -> F2 = no_car)) & (F3 = carF
 -> F3 = no_car)) & (F4 = carF -> F4 = no_car)) & (F5 = carF -> F5 = no_car))) ->  F F6 = carF)  is true
-- specification  G ((((((L2 = green & L4 = green) & L6 = green) & L8 = green) & L10 = green) & (((((B6 = carB -> B6 = no_car) & (B5 = carB -> B5 = no_car)) & (B4 = car
B -> B4 = no_car)) & (B3 = carB -> B3 = no_car)) & (B2 = carB -> B2 = no_car))) ->  F B1 = carB)  is true
NuSMV >
```

Figure 12: Output in NuSMV showing LTL spec verification for "G" is true for vehicle covering paths in order BCD and DCB

NuSMV model checking tool output explanation:

Case 1 (B->C->D) explanation from NuSMV model checker output: Here the output signifies that, for each and every transition when vehicle is at point 4 from figure 10, then for next state **F1 = carF** (car is at point B from figure 10) and **L1 = green** as signal for corresponding 2-way intersection. Next state is the next segment **F2 = carF** (car is at point 8 from figure 10) and **L3 = green** as signal for corresponding 3-way intersection, next state is the next segment **F3 = carF** (car is at point C from figure 10) and **L5 = green** as signal for corresponding 2-way intersection, next state is the next segment **F4 = carF** (car is at point 6 from figure 10) and **L7 = green** as signal for corresponding 3-way intersection, next state is the next segment **F5 = carF** (car is at point D from figure 10) and **L9 = green** as signal for corresponding 2-way intersection. Thus covering all the carF will be at **F6** (F **F6 = carF** means Final state of system at which vehicle is present).

Case 2 (D->C->B) explanation from NuSMV model checker output: Here the output signifies that, for each and every transition when vehicle is at point 2 from figure 10, then for next state **B6 = carB** (car is at point D from figure 10) and **L2 = green** as signal for corresponding 2-way intersection. Next state is the next segment **B5 = carB** (car is at point 6 from figure 10) and **L4 = green** as signal for corresponding 3-way intersection, next state is the next segment **B4 = carB** (car is at point C from figure 10) and **L6 = green** as signal for corresponding 2-way intersection, next state is the next segment **B3 = carB** (car is at point 8 from figure 10) and **L8 = green** as signal for corresponding 3-way intersection, next state is the next segment **B2 = carB** (car is at point B from figure 10) and **L10 = green** as signal for corresponding 2-way intersection. Thus covering all the carF will be at **B1** (F **B1 = carB** means Final state of system at which vehicle is present).

Symbols meanings:

a) **&** means **and**
b) **->** means **next state**
c) **!** means **negate** or **not**
d) **G** means **Global**

carF -> car moving in forward path; carB -> car moving in backward path; no car -> no car is present at a slot; L1, L2,…L10 -> signals; F1……F6 & B1……B6 -> paths forward & backward.

- Therefore the requirement of **every vehicle must visit all of points B, C and D** is satisfied globally (G) using the linear temporal logic (check_ltlspec) for the 3*3 road grid with traffic signal.

4. **THERE IS NO COLLISION BETWEEN ANY 2 VEHICLES.**

At any point of time in 3*3 road grid with traffic signal system, there should never be collision between any 2 vehicles.

Contextual explanation:

In the phase A of the project, the vehicle group defined 30 vehicles for the entire 3*3 road grid with traffic signal system. We are considering all the 30 vehicles to verify the property of "**no collision between any 2 vehicles**". For that, we are here considering a segment in the road system and inserting 30 vehicles into that path. Here we are verifying the property for the segment present between point 1 and 2 (refer to Figure 10) for proper visualization. Below is an animated figure of a vehicle collision describing the situation of above scenario.

Figure 13: Collision of 2 vehicles

- Here the property of vehicles having any collision is considered as something bad that should never happen for this system, hence this signifies the "**Safety**" property for the vehicle.

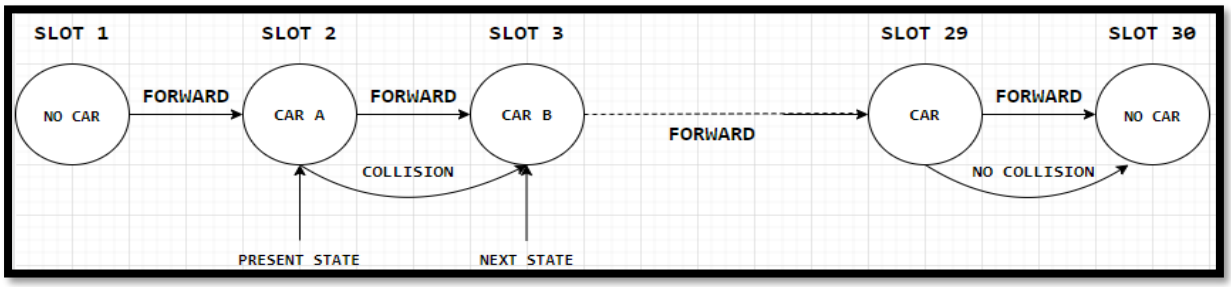

Figure 14: state system for collision of 2 vehicles

Verification of state system explanation (theoretical):

Here we are considering a road segment between points 1 and 2 (refer to the figure 10), there are total 30 slots with vehicles assigned randomly in those slots (refer to figure 14). Now consider a slot where there is a vehicle as a present state, the condition that the system should follow is, check if there is vehicle in the next slot or not, if there is no vehicle then the present state vehicle will move to the next slot or state, else if there is vehicle in the next slot (carB) then the present state vehicle (carA) should not go to next state.

Verification of state system explanation (practical) using NUSMV model checking tool:

Using NUSMV we modeled the 3*3 road grid with traffic signal. For checking the collision between any two vehicles, here we are assigning the slots with Boolean values i.e. if car is present means slot value is 1 (carA or carB) else the value of vehicle is 0.

Using the **Linear Temporal Logic** concept we verified for the **Global** specification using the below condition. In the **figure 15, check_ltlspec** refers to checking Linear Temporal Logic, **G** refers to Global condition i.e., satisfies for each and every transition of the system.

```
NuSMV > go
NuSMV > check_ltlspec
-- specification  G carA != carB  is true
-- specification  G carA = carB  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    carA = 15
    carB = -1
  -> State: 1.2 <-
    carA = 16
    carB = 0
  -> State: 1.3 <-
    carA = 17
    carB = 1
  -> State: 1.4 <-
    carA = 18
    carB = 2
```

Figure 15: Output in NuSMV showing LTL spec verification for "G" for both cases of collision and no collision

NuSMV model checking tool output explanation:

Here the carA and carB are present slot vehicle and vehicle in the next slot respectively. For the property to satisfy the value of carA and carB should never be same since both carA and

carB are assigned with the same Boolean value of 1. And value 1 refers to the slot with a vehicle, so if both are having vehicles at their slots respectively then there should be no movement in the vehicle i.e., carA and carB should not be equal i.e. **carA != carB** (**refer to figure 15**) is **True** for the 3*3 road grid with traffic signal system. When both are equal i.e. **carA = carB** there occurs collision which is showed in the counter example (**refer to figure 15**) as a **False** condition.

Symbols meanings:

a) **&** means **and**
b) = checking for the condition whether they are equal
c) **!** means **negate** or **not**
d) **G** means **Global**
e) **carA is vehicle in present slot (or state)**
f) **carB is vehicle in next slot (or state)**

- Therefore the requirement of "**no collision between any 2 vehicles**" is satisfied globally (G) using the linear temporal logic (check_ltlspec) for the 3*3 road grid with traffic signal.

5. **ANOTHER PROPERTY AT YOUR OWN CHOICE (AT MOST ONE VEHICLE INSERTION AT POINT A).**

At any point of time in 3*3 road grid with traffic signal system, there can enter **at most one vehicle at a time from the point A.** N number of vehicles can enter the road system but in a sequential manner and not parallel.

Contextual explanation:

The last property which we are going to verify is, in a 3*3 road grid with traffic signal system at most 1 vehicle can enter at a time. The distance between points 0 and 1 has only 2 paths one for vehicle moving from 0-1 (forward path) and other for the vehicle moving 1-0 (backward path). Below is the animated image showing the cars in a queue for both entering and leaving grid. Here the property that we are focused is only, **there can enter at most one vehicle at a time from the point A.** Below is a animated image of the vehicles going to and fro into the point A for proper visualization.

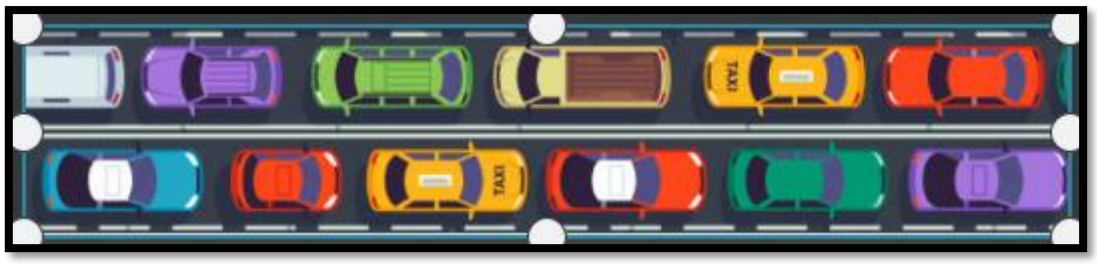

Figure 16: Vehicles entering (forward path) and leaving (backward path) at point A (refer to Figure 10 directed graph)

- Here the property of vehicles entering first slot of road segment 0 at point A (refer to the figure 10) is at most one i.e. at any instant of time no two vehicles can stay in a single slot 1 if that happens it is exhibiting bad behavior, hence this signifies the **"Safety"** property for the vehicle.

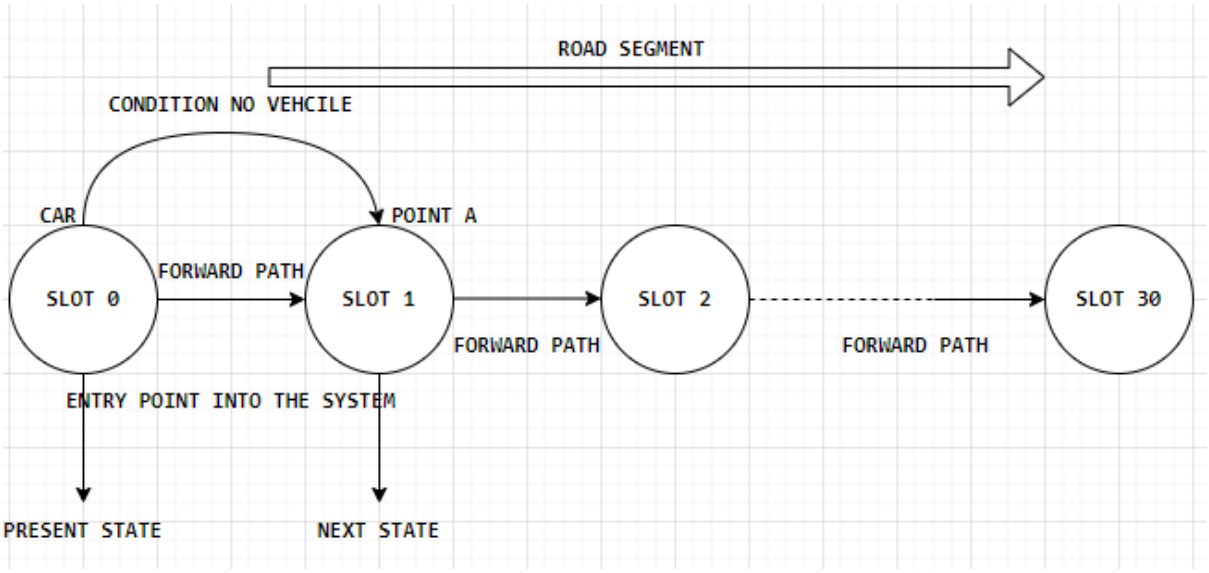

Figure 17: state system for vehicle insertion into the point A

Verification of state system explanation (theoretical):

Here we are considering a road segment between points 0 and 1 (refer to the figure 10), there are total 30 slots within the segment from point A to point 1. Here a vehicle at slot 0 i.e. slot before entering the 3*3 road grid with traffic signal, it should check if there is vehicle in the next segment. If there is no vehicle then the car can enter into slot 1 of road segment and continue to move in forward path else if vehicle is present at slot 1 then the vehicle at slot 0 before road segment should not move.

Verification of state system explanation (practical) using NUSMV model checking tool:

Using NUSMV we modeled the 3*3 road grid with traffic signal. For checking the vehicle insertion property we are considering

here a total number of 200 cars, the condition for vehicle getting inserted into the road segment is no vehicle should be present at the slot 1 of the road system at point A. We are assigning a counter for number of cars entering and showing the result.

Using the **Linear Temporal Logic** concept we verified for the **Global** specification using the below condition. In the **figure 18, check_ltlspec** refers to checking Linear Temporal Logic, **G** refers to **Global condition** i.e., satisfies for each and every transition of the system.

```
NuSMV > go
NuSMV > check_ltlspec
-- specification  G !(pointA_state >= 2)  is true
```

```
-> State: 1.2 <-
  pointA_state = 1
  car_count = 199
-> State: 1.3 <-
  pointA_state = 0
  car_count = 199
-> State: 1.4 <-
  pointA_state = 1
  car_count = 198
-> State: 1.5 <-
  pointA_state = 0
  car_count = 198
```

Figure 18: Output in NuSMV showing LTL spec verification for "G" for insertion of vehicle at point A of the road system

```
NuSMV > go
NuSMV > check_ltlspec
-- specification  G pointA_state >= 2  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    pointA_state = 0
    car_count = 200
  -> State: 1.2 <-
    pointA_state = 1
    car_count = 199
  -> State: 1.3 <-
    pointA_state = 0
  -> State: 1.4 <-
    pointA_state = 1
    car_count = 198
```

Figure 19: Output in NuSMV showing LTL spec verification for "G" for insertion of vehicle at point A of the road system (counter example)

NuSMV model checking tool output explanation:

Here both the **figures 18** and **19** are showing the output for the **insertion of vehicles** at **pointA** with **example** and a **counter example.** As discussed above here we are considering **200 vehicles** and entering them sequentially at pointA whilst verifying the condition whether a vehicle is present at the slot or next state of the road segment. Here the output (**for figure 18**) says that !**(pointA_state >=2)** is **true** means **number of vehicles at pointA_state is always less than 2 and the car_count decrement is by 1 vehicle for each state.** And for counter example part the output (**for figure 19**) says that **pointA_state >=2** is **false** and all the states and **car_count** is represented respectively.

Symbols meanings:

a) = checking for the condition whether they are equal
b) **!** means **negate** or **not**
c) **G** means **Global**
d) **pointA_state means state of the first slot of segment 1**
e) **car_count** means the counter assigned to **number of vehicles.**

- Therefore the requirement of "**at most one vehicle insertion at point A**" is satisfied globally (G) using the linear temporal logic (check_ltlspec) for the 3*3 road grid with traffic signal.

CONCLUSION:

For phase C, all the properties defined for the I-group is verified and corresponding results are displayed in the above section for the 3*3 road grid with traffic signal. Here are the properties are verified using the NuSMV model checking tool. The conditions that we verified for the above properties satisfies globally (G) using the Linear Temporal Logic (check_ltspec). Here all the vehicle properties defined and verified are for the system designed by I-group and V-group on a whole considering both the phase A and phase B of the project.