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.

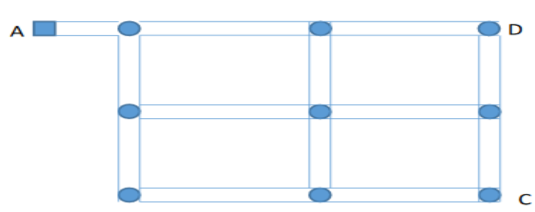

Figure 1. Road System

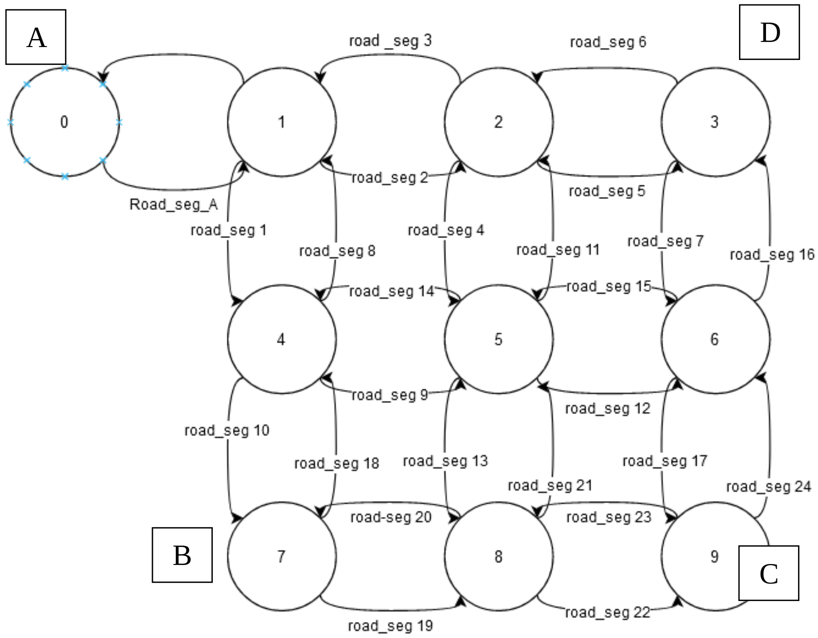

Figure 2: Directed Graph of the Road System

```
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

```python
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

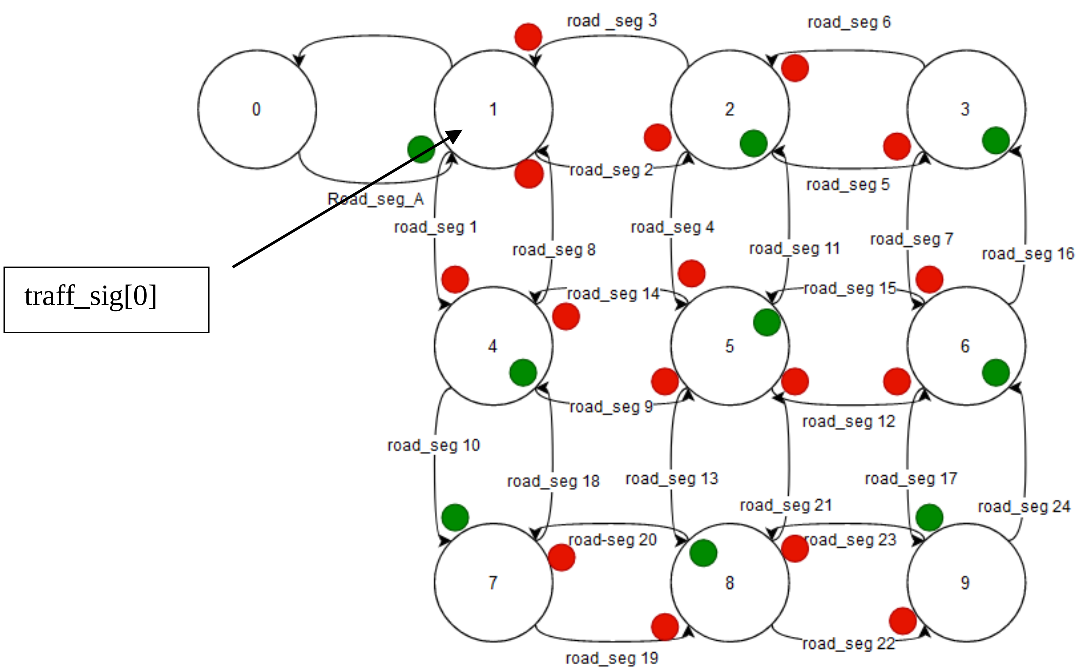

Figure 6: Road System with Intersection based on Figure 4

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

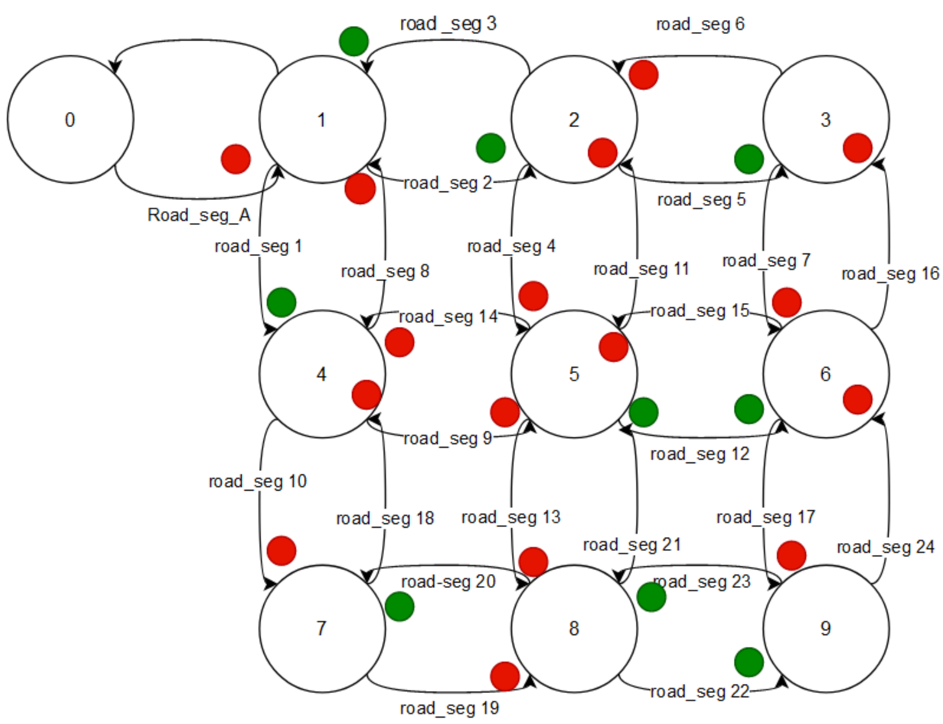

Figure 9: Traffic Signal Sequence After Rotating Once

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, 0, 1, 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

```python
"""
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
```

```python
# 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])
```