

Lab 2

Objective

An existing state machine exists that models the lights of a traffic stop at a 4 way stop. The lights periodically change from green to yellow to red on each side to allow for the proper flow of traffic. The traffic lights also include arrows to provide protected left lane turning. This state machine is implemented in UMPLE and simulated in a java application.

The objective of this lab is to build on the existing state machine modelling a traffic light stop developed in UMPLE, and add specific behaviour for 3 different modes of traffic; low, medium and high. The state machine developed with UMPLE is to be used to generate Java Code to build on existing code in order to simulate the traffic stop with a java swing GUI application.

Design and Discussion

The state machine developed through UMPLE is as follows:

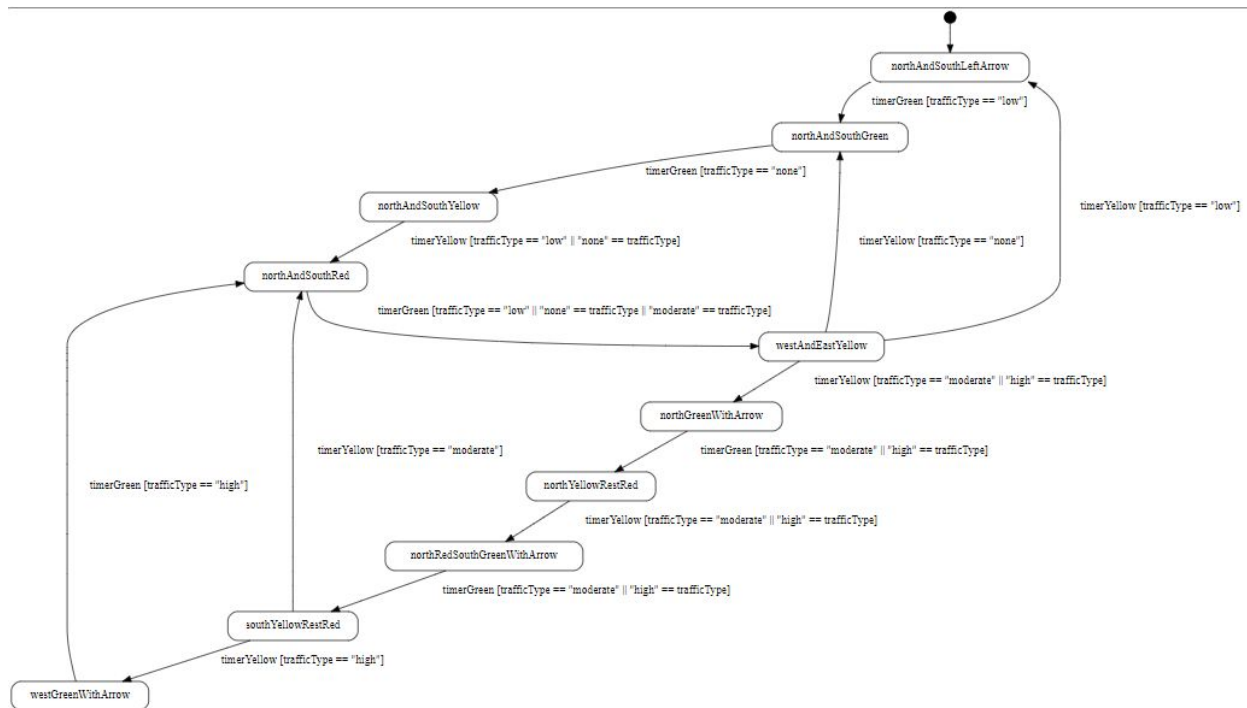


Figure 1: State Machine Modelling Traffic Light Stop Under 3 Different Traffic Modes

The UMPLE code that developed this state machine is shown below:

Ali Aftab, Muhammad

300067438

Mustafa Ali

300060406

```
class TrafficLightNew
{
    status {
        northAndSouthLeftArrow {
            entry { trafficLightManager.northArrow(); }
            entry { trafficLightManager.southArrow(); }
            entry { trafficLightManager.westRed(); }
            entry { trafficLightManager.eastRed(); }
            timerGreen() [trafficType == low] -> northAndSouthGreen;
        }
        northAndSouthYellow {
            entry { trafficLightManager.northYellow(); }
            entry { trafficLightManager.southYellow(); }
            entry { trafficLightManager.westRed(); }
            entry { trafficLightManager.eastRed(); }
            timerYellow() [trafficType == low || trafficType == "none"] -> northAndSouthRed;
        }
        northAndSouthRed {
            entry { trafficLightManager.northRed(); }
            entry { trafficLightManager.southRed(); }
            entry { trafficLightManager.westGreen(); }
            entry { trafficLightManager.eastGreen(); }
            timerGreen() [trafficType == low || trafficType == "none" || trafficType == "moderate"] -> westAndEastYellow;
        }
        westAndEastYellow {
            entry { trafficLightManager.northRed(); }
            entry { trafficLightManager.southRed(); }
            entry { trafficLightManager.westYellow(); }
            entry { trafficLightManager.eastYellow(); }
            timerYellow() [trafficType == moderate || trafficType == "high"] -> northGreenWithArrow;
            timerYellow() [trafficType == low] -> northAndSouthLeftArrow;
            timerYellow() [trafficType == none] -> northAndSouthGreen;
        }
        northAndSouthGreen {
            entry { trafficLightManager.northGreen(); }
            entry { trafficLightManager.southGreen(); }
            entry { trafficLightManager.westRed(); }
            entry { trafficLightManager.eastRed(); }
            timerGreen() [trafficType == none] -> northAndSouthYellow;
        }
        northGreenWithArrow {
            entry { trafficLightManager.northGreenAndArrow(); }
            entry { trafficLightManager.southRed(); }
            entry { trafficLightManager.westRed(); }
            entry { trafficLightManager.eastRed(); }
            timerGreen() [trafficType == moderate || trafficType == "high"] -> northYellowRestRed;
        }
        northYellowRestRed {
            entry { trafficLightManager.northYellow(); }
            entry { trafficLightManager.southRed(); }
            entry { trafficLightManager.westRed(); }
            entry { trafficLightManager.eastRed(); }
            timerYellow() [trafficType == moderate || trafficType == "high"] -> northRedSouthGreenWithArrow;
        }
        northRedSouthGreenWithArrow {
            entry { trafficLightManager.northRed(); }
            entry { trafficLightManager.southGreenAndArrow(); }
            entry { trafficLightManager.westRed(); }
            entry { trafficLightManager.eastRed(); }
            timerGreen() [trafficType == moderate || trafficType == "high"] -> southYellowRestRed;
        }
        southYellowRestRed {
            entry { trafficLightManager.northRed(); }
            entry { trafficLightManager.southYellow(); }
            entry { trafficLightManager.westRed(); }
            entry { trafficLightManager.eastRed(); }
            timerYellow() [trafficType == moderate] -> northAndSouthRed;
            timerYellow() [trafficType == high] -> westGreenWithArrow;
        }
        westGreenWithArrow {
            entry { trafficLightManager.northRed(); }
            entry { trafficLightManager.southRed(); }
            entry { trafficLightManager.westGreenAndArrow(); }
            entry { trafficLightManager.eastRed(); }
            timerGreen() [trafficType == high] -> northAndSouthRed;
        }
    }
}
```

Figure 2: UMPLE Code to Develop Modified State Machine

The code is very simple in that it follows a standard convention. Each possible state is defined with some descriptive name and upon entry into the state, certain effects are carried out. These are not too relevant to the state diagram but rather more relevant to the actual java implementation in that trafficLightManager is the class that is what interfaces with the GUI of the

Mustafa Ali
300060406

app and as such, certain methods need to be called depending on the current state. Moreover, the way the system works is two main events are periodically called in an alternative fashion. A timerGreen event occurs when there is some light that is green and has used up its time being green. After this event, the light turns yellow. A timerYellow event indicates the end of a yellow light session and usually another light turns green while the light that was yellow itself turns red. With this in mind, it is evident in the code that after a certain event is triggered while in a specific state, a state transition needs to occur. The state to transition to is dependent on guard conditions that differentiate the three different traffic modes. There are key points at which the different traffic modes branch off from one another in terms of the next state to go to. These guard conditions are crucial in order to ensure the proper sequence of states for a certain traffic level. An observation that was made is that in certain cases, two green events would happen in a row since a light would be going from arrow to green or vice versa meaning that the length of time that specific light would stay the same would be longer than normal. This makes sense, however, since an arrow is considered green anyway.

The above UMPLE code was translated to JAVA and the code generated is attached in this lab submission. It is clear the code is quite similar to the original provided implementation with some key differences. The main differences are found inside the timerGreen and timerYellow methods, the addition of the trafficType variable and the implementation of the moderateTraffic, lowTraffic, and highTraffic methods. The general design is that the traffic methods will set an initial state for the traffic type and set the trafficType variable accordingly. Whenever the timerGreen and timerYellow events are now called, the code inside them, via decision based logic with switch-case and if statements, will determine the appropriate state to transition to from the current one. To determine what state to transition to, the code looks at not only the current state but also takes into account the newly added trafficType variable as different traffic types might have different transitions from one common state to another. Examples of code snippets are seen below.

```
334 • @Override
335 public boolean lowTraffic() {
336     setStatus(Status.northAndSouthLeftArrow);
337     trafficType="low";
338     return true;
339 }
```

Figure 3: lowTraffic Method Implementation to Set Initial State and Traffic Type

Ali Aftab, Muhammad
300067438

Mustafa Ali
300060406

```
case westAndEastYellow:
    if (trafficType.equals("moderate") || trafficType.equals("high"))
    {
        setStatus(Status.northGreenWithArrow);
        wasEventProcessed = true;
        break;
    }
    if (trafficType.equals("low"))
    {
        setStatus(Status.northAndSouthLeftArrow);
        wasEventProcessed = true;
        break;
    }
    if (trafficType.equals("none"))
    {
        setStatus(Status.northAndSouthGreen);
        wasEventProcessed = true;
        break;
    }
    break;
```

Figure 4: timerGreen Method Implementation Slice

Figure 4 above shows what occurs when the current state is westAndEastYellow and a timerYellow event occurs. The transition to the next state is dependent on the traffic type and is clear that the different traffic types change status differently. The way the new traffic light class was implemented involved using the strings low, moderate, high and none to indicate the different traffic types. None refers to the original traffic design before the new traffic type behaviour was added.

The only challenges initially faced was determining how to incorporate all different flows of traffic into one state diagram but then it slowly became evident that the simplest solution was guard conditions. By utilizing guard conditions, one could branch the state transitions to properly suit the flow of events for each different traffic type behaviour. Of course the generated java code did not include this additional variable used in the guard conditions but the solution was to simply add it as a private variable inside the new traffic light class.

Conclusion

The objective of the lab was to implement new behaviour into a system modelling a traffic light to provide better simulation of low, medium and high traffic. This goal was achieved simply by adding new states into the state machine and using guard conditions that relied on a trafficType variable in order to control the sequence of state transitions depending on the type of traffic. The translation to java code was made simpler by the fact that the changes compared to the original were very minimal. Overall, the new state machine and the java implementation both perfectly model the behaviour of the traffic light under low, moderate and high traffic conditions.