Embedded Control Laboratory

Report of lab Exercise 2

WS 2023/24

Group 10

1762859	**Gopal Vaid**
1767672	**Samyak Indurkar**
1781238	**Darren Melroy Menezes**

Date: 18.12.2023, Version: 01

Lab 2: Modeling State Machines in UPPAAL

1. Introduction

Discrete-time modeling has proven to be a robust method for designing systems, especially when continuous modeling becomes complex or impractical. State machines, or automata, are pivotal in this approach, offering abstract representations of systems transitioning between various states. In the realm of discrete-time modeling, tools like UPPAAL play a crucial role by providing a comprehensive platform for simulating and verifying the behavior of state machines. This report explores the practical application of state machines by modeling a traffic light system at a city cross-section. Through the use of UPPAAL, we aim to demonstrate the relevance and effectiveness of discrete-time modeling in everyday scenarios.

2. State Machines

State machines, historically rooted in computation and logic theories, have evolved from theoretical constructs to practical tools. They represent systems by capturing various states and the rules governing transitions between these states. Key components of a state machine include:-

1. States:
- States represent distinct conditions or situations in a system.
- Each state captures a specific configuration or behavior of the system.
- For example, in a traffic light system, states could include "Red," "Yellow," and "Green."

2. Transitions:
- Transitions define the pathways or conditions that enable a system to move from one state to another.
- They are associated with specific criteria that must be satisfied for the shift to occur.
- In the context of a traffic light, a transition could occur from "Green" to "Red" based on a timer.

3. Guards:
- Guards are boolean expressions associated with transitions.
- They determine whether a transition is permissible.
- For example, a guard in a digital watch model might check if the current time equals the alarm time before transitioning to the "Alarm Ringing" state.

4. Set Actions:
- Set actions are operations executed once a transition is taken.
- They represent changes or activities that occur when moving from one state to another.
- In a digital watch, a set action upon entering the "Alarm Ringing" state might be to initiate the alarm sound.

5. Variables:

- Variables are parameters that can change over time and influence the system's behavior.
- They play a pivotal role in representing dynamic aspects of a system.
- For instance, a clock variable in a state machine can measure the passage of time and affect transitions based on elapsed time.

In summary, states represent conditions, transitions define how a system moves between states, guards control the conditions for transitions, set actions define what happens during transitions, and variables capture dynamic aspects of the system. These concepts collectively form the basis for modeling complex systems using state machines.

UPPAAL, a powerful tool for modeling, simulating, and verifying real-time systems, utilizes timed automata theory for precise analysis.

3. UPPAAL: Modeling and Verifying Real-Time Systems

UPPAAL, a collaborative effort between Uppsala University and Aalborg University, stands as a sophisticated tool environment dedicated to the modeling and verification of real-time systems. Rooted in the robust theoretical framework of timed automata, UPPAAL goes beyond traditional modeling tools by incorporating advanced features, including the capacity to handle stochastic hybrid systems.

Key Components:

Timed Automata Theory: UPPAAL's core foundation lies in timed automata theory, a powerful formalism for modeling systems with temporal and concurrent aspects. This theory enables the precise representation of real-time behavior in a structured manner.

Integrated Environment: UPPAAL provides a comprehensive environment equipped with essential components. The editor serves as the primary workspace for designing models, allowing users to define states, transitions, and associated properties. The simulator facilitates the step-by-step execution of models, providing a dynamic view of system behavior. The verifier is instrumental in model checking, allowing users to specify and evaluate properties to ensure the robustness of their models.

Tasks Overview:
In the subsequent sections, we will explore two specific tasks to illustrate the practical application of UPPAAL. Task 1 involves the intricate modeling of car and pedestrian traffic lights at a city crossroad, emphasizing the synchronization and safety aspects. Task 2 delves into the modeling of a two-elevator system servicing requests from six floors, showcasing the complexity of real-time system design.

.

4. Task 1: Modeling Car and Pedestrian Traffic Lights

In this task, we aim to model a traffic light system for both cars and pedestrians at a city crossroad using UPPAAL as shown in figure 1.

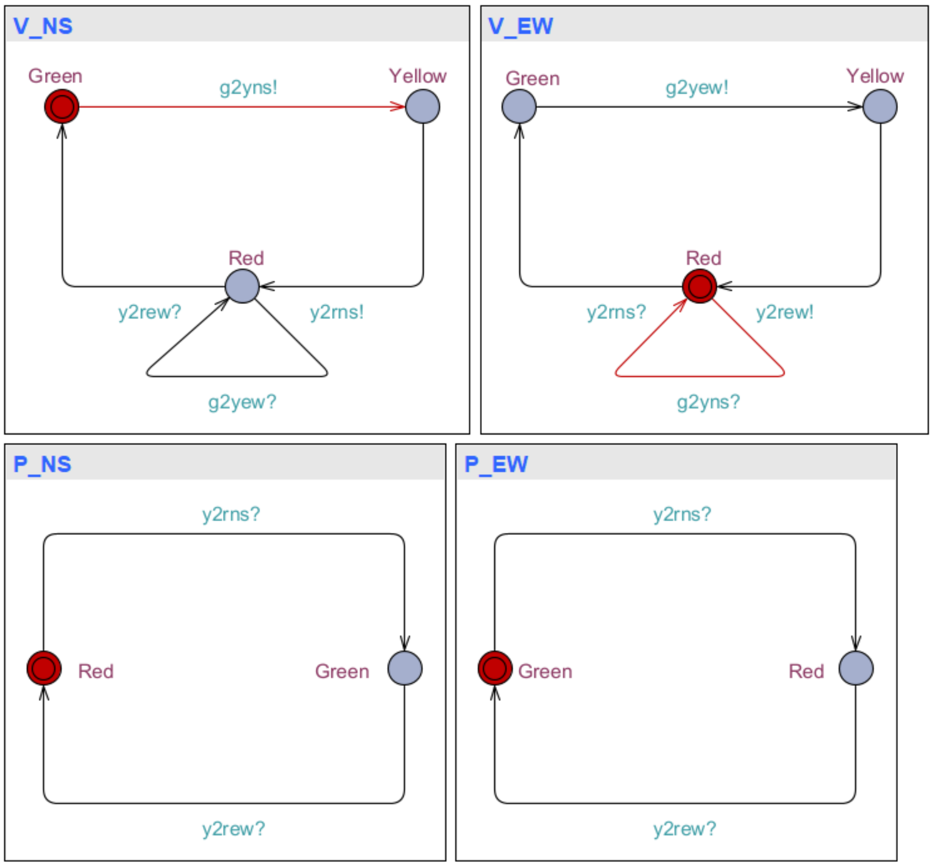
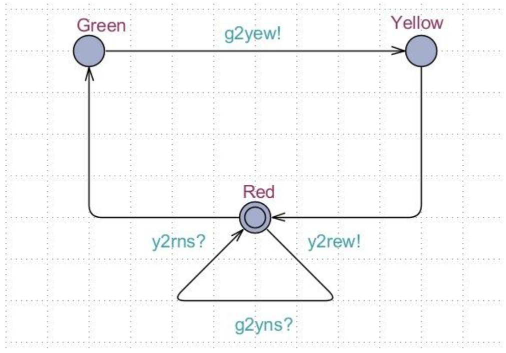


Figure 1: State View of Car and Pedestrian Traffic Lights

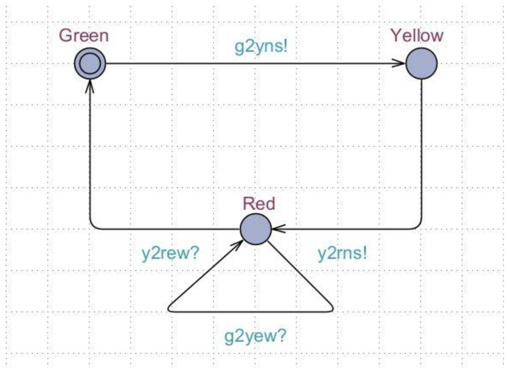
The system comprises four components: Vehicle North-South (V_NS), Vehicle East-West (V_EW), Pedestrian North-South (P_NS), and Pedestrian East-West (P_EW). Each component has its traffic light states and synchronized behaviors to ensure safe traffic flow. The model is defined using the UPPAAL language, incorporating synchronization channels and transitions.

1.1 Vehicle North-South (V_NS)

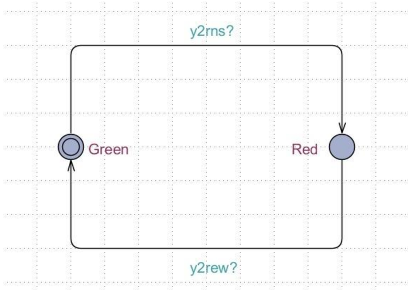
- States: Red, Yellow, Green
- Initial State: Green
- Transitions:
  - Red to Red (guarded by g2yew?)
  - Red to Green (guarded by y2rew?)
  - Yellow to Red (synchronized with y2rns!)
  - Green to Yellow (synchronized with g2yns!)



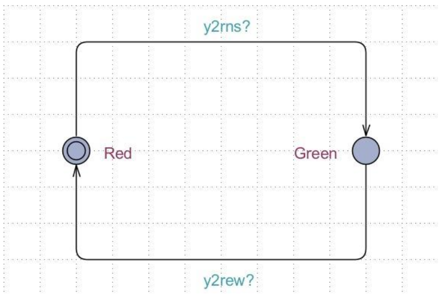
1.2 Vehicle East-West (V_EW)
- States: Red, Yellow, Green
- Initial State: Red
- Transitions:
  - Red to Red (guarded by g2yns?)
  - Yellow to Red (synchronized with y2rew!)
  - Green to Yellow (guarded by g2yew?)
  - Red to Green (synchronized with y2rns!)



1.3 Pedestrian North-South (P_NS)
- States: Red, Green
- Initial State: Red
- Transitions:
  - Red to Green (synchronized with y2rew?)
  - Green to Red (synchronized with y2rns!)



1.4 Pedestrian East-West (P_EW)
- States: Red, Green
- Initial State: Green
- Transitions:
  - Green to Red (synchronized with y2rew?)
  - Red to Green (synchronized with y2rns!)



Acronym

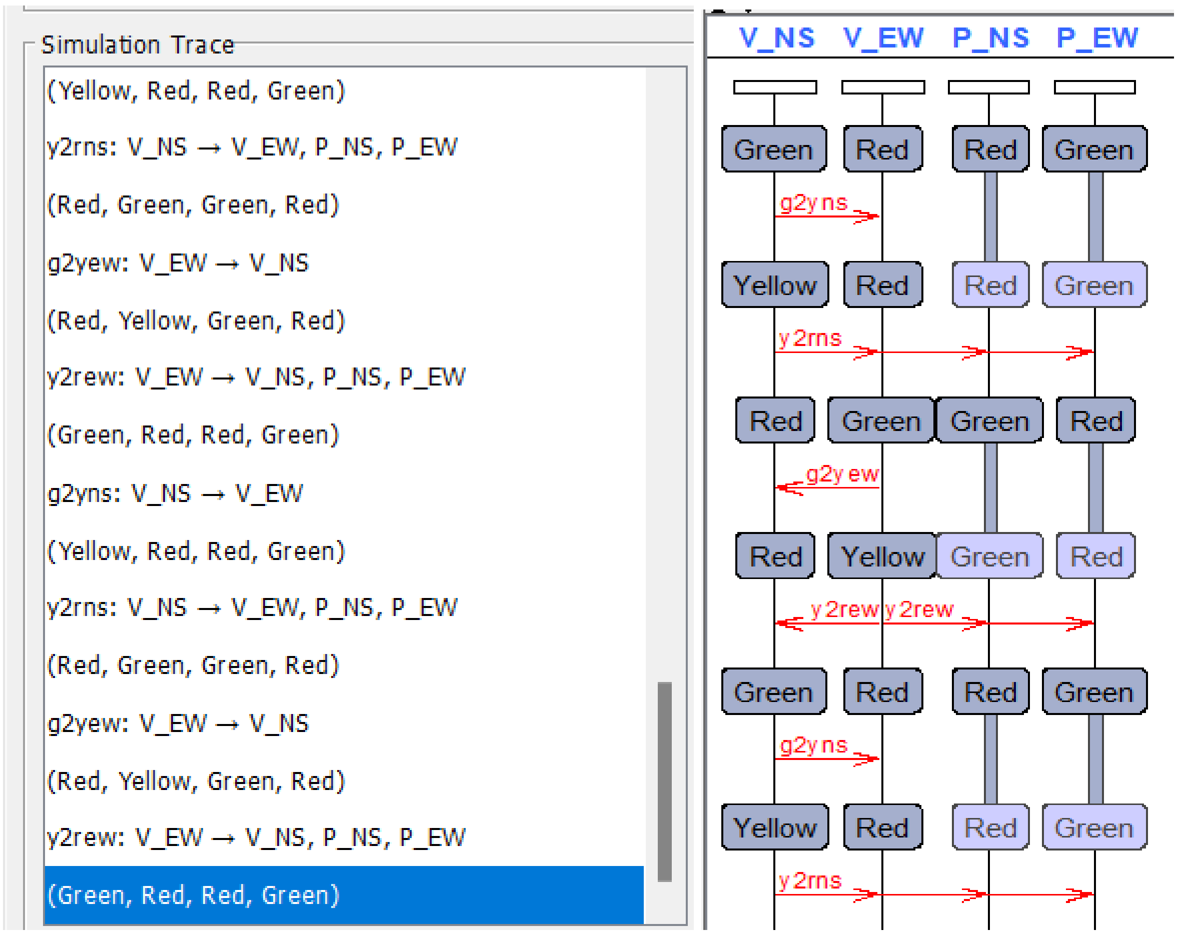
- V_NS = North-South road
- V_EW = East-West road
- P_NS = Pedestrian crossing on North-South road
- P_EW = Pedestrian crossing on East-West road
-
- g2yns = Green to Yellow transition on North-South road
- y2rns = Yellow to Red transition on North-South road
- g2yew = Green to Yellow transition on East-West road
- y2rew = Yellow to Red transition on East-West road

1.5 System Declaration

- System: V_NS, V_EW, P_NS, P_EW

1.6 Simulation

Using the UPPAAL simulator, the behavior of the traffic light system can be observed step by step. During simulation, the real-time visualization in the right pane reflects the changes in the states and transitions which are shown in image below. Colors of locations change based on their active/inactive status, and transitions animate as they're taken, offering a dynamic view of the system's behavior.



```
Simulation Trace
(Yellow, Red, Red, Green)
y2rns: V_NS → V_EW, P_NS, P_EW
(Red, Green, Green, Red)
g2yew: V_EW → V_NS
(Red, Yellow, Green, Red)
y2rew: V_EW → V_NS, P_NS, P_EW
(Green, Red, Red, Green)
g2yns: V_NS → V_EW
(Yellow, Red, Red, Green)
y2rns: V_NS → V_EW, P_NS, P_EW
(Red, Green, Green, Red)
g2yew: V_EW → V_NS
(Red, Yellow, Green, Red)
y2rew: V_EW → V_NS, P_NS, P_EW
(Green, Red, Red, Green)
```

Fig 2: Transition view

1.6 Verification Queries

1. A[] not deadlock:

This query ensures that there is never a state of deadlock in the system, emphasizing continuous and safe operation.

2. A[] not (V_NS.Green && V_EW.Green):
This query guarantees that car traffic lights for North-South and East-West directions are never simultaneously green, preventing potential conflicts and ensuring traffic safety.

3. A[] (V_EW.Green imply P_EW.Red) && (V_NS.Green imply P_NS.Red):
This query enforces the synchronization between car and pedestrian lights. It ensures that when car lights are green, the corresponding pedestrian lights indicate red, promoting safe crossing.

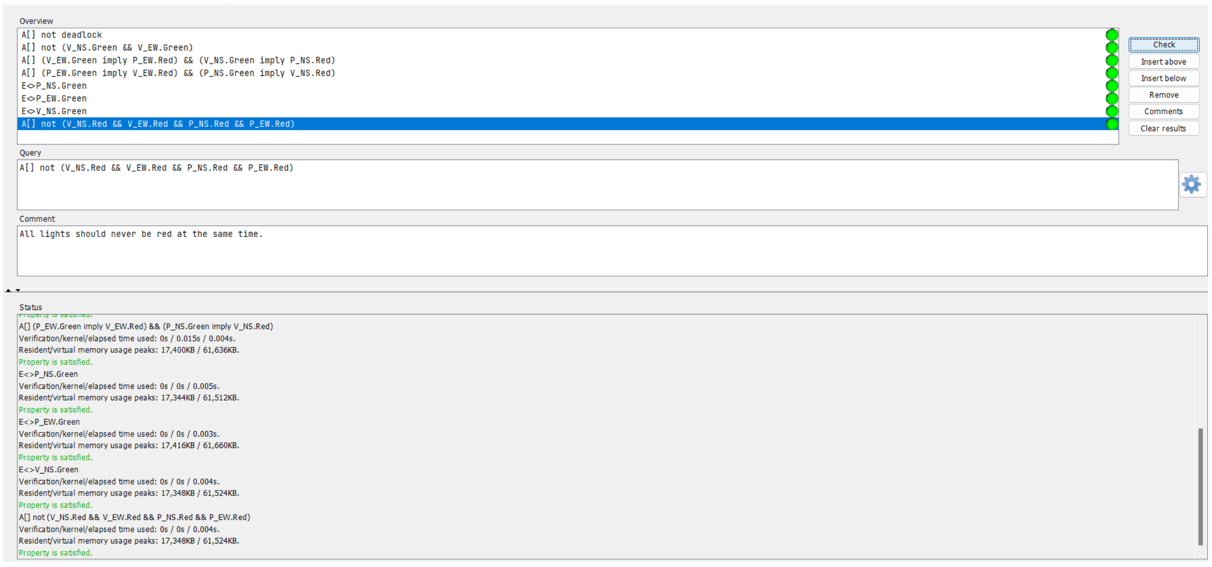
4. A[] (P_EW.Green imply V_EW.Red) && (P_NS.Green imply V_NS.Red):
Similar to the previous query, this ensures that when pedestrian lights are green, the corresponding car lights indicate red, preventing conflicts at pedestrian crossings.

5. E<>P_NS.Green:
This query asserts that at some point in the future, the pedestrian lights for North-South direction will turn green, guaranteeing pedestrians the opportunity to cross.

6. E<>P_EW.Green:
Similar to the previous query, this ensures that at some point in the future, the pedestrian lights for East-West direction will turn green, allowing pedestrians to cross safely.

7. E<>V_NS.Green:
This query ensures that at some point in the future, the car lights for North-South direction will turn green, facilitating the flow of traffic in that direction.

8. A[] not (V_NS.Red && V_EW.Red && P_NS.Red && P_EW.Red):
This query guarantees that all lights (car and pedestrian) are never simultaneously red, ensuring a continuous flow of either car or pedestrian traffic.



```
A[] not deadlock
A[] not (V_NS.Green && V_EW.Green)
A[] (V_EW.Green imply P_EW.Red) && (V_NS.Green imply P_NS.Red)
A[] (P_EW.Green imply V_EW.Red) && (P_NS.Green imply V_NS.Red)
E<>P_NS.Green
E<>P_EW.Green
E<>V_NS.Green
A[] not (V_NS.Red && V_EW.Red && P_NS.Red && P_EW.Red)
```

```
A[] not (V_NS.Red && V_EW.Red && P_NS.Red && P_EW.Red)
```

```
All lights should never be red at the same time.
```

```
A[] (P_EW.Green imply V_EW.Red) && (P_NS.Green imply V_NS.Red)
Verification/kernel/elapsed time used: 0s / 0.015s / 0.004s.
Resident/virtual memory usage peaks: 17,400KB / 61,636KB.
Property is satisfied.
E<>P_NS.Green
Verification/kernel/elapsed time used: 0s / 0s / 0.005s.
Resident/virtual memory usage peaks: 17,344KB / 61,512KB.
Property is satisfied.
E<>P_EW.Green
Verification/kernel/elapsed time used: 0s / 0s / 0.003s.
Resident/virtual memory usage peaks: 17,416KB / 61,660KB.
Property is satisfied.
E<>V_NS.Green
Verification/kernel/elapsed time used: 0s / 0s / 0.004s.
Resident/virtual memory usage peaks: 17,348KB / 61,524KB.
Property is satisfied.
A[] not (V_NS.Red && V_EW.Red && P_NS.Red && P_EW.Red)
Verification/kernel/elapsed time used: 0s / 0s / 0.004s.
Resident/virtual memory usage peaks: 17,348KB / 61,524KB.
Property is satisfied.
```

Figure 3: Verification Query

These verification queries collectively ensure the proper functioning of the traffic light system, preventing deadlocks, maintaining synchronization, and promoting safe traffic flow for both cars and pedestrians.

The UPPAAL queries validate the correctness of the model and ensure safety conditions are met, such as avoiding deadlocks and maintaining proper synchronization between car and pedestrian lights.

The subsequent sections will delve into the detailed explanation of each component, transitions, and the rationale behind the verification queries.