University of Minho

Cyber Physical Computation

Modelling and analysis of a cyber-physical traffic lights system

Diogo Ramos
Pg53771

Gabriel Costa
Pg53828

April, 2024

Contents

1 First Part

1.1 Introducing the Problem

On the first part of the assignment we were given the task of modelling a system of traffic lights ate a T-junction:

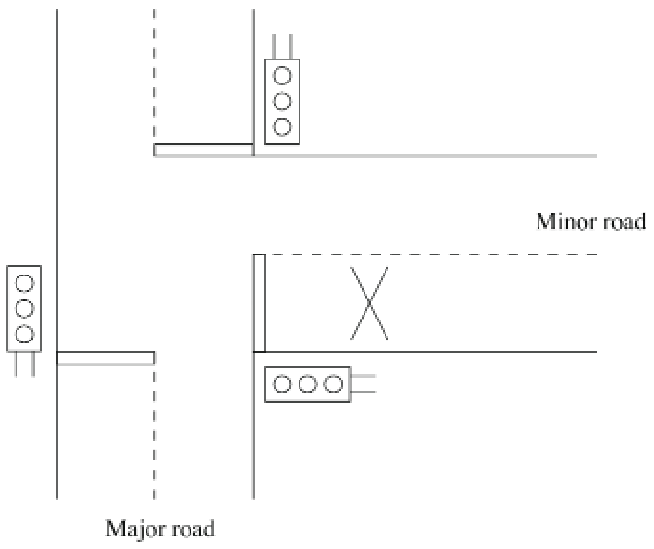


Figure 1: Structure of the T-Junction

In this system the light is always green on the major road, except when a car is detected in the minor road. In the latter case the lights have to commute, and then revert to the initial state.
Given this, the following restrictions were given:

1. The lights on the major road will be always set on green, and red on the minor road unless a vehicle is detected by the sensor.
2. In the latter case, the lights will switch in the standard manner and allow traffic to leave the minor road. After a suitable time interval (30s), the lights will revert to their default position so that traffic can flow on the major road again.
3. Finally, as soon as a vehicle is detected by the sensor the latter is disabled until the minor-road lights are on red again

Along with this restrictions, the following temporal constraints were introduced:

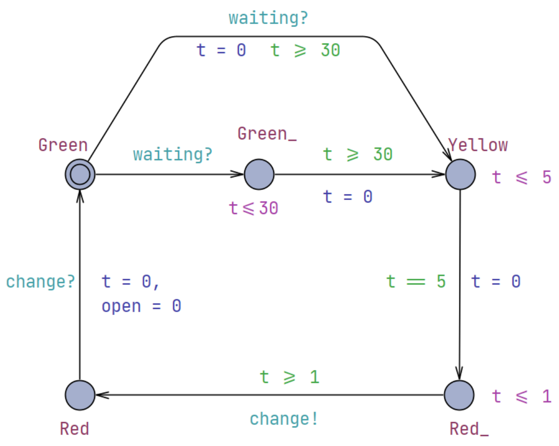
1. Interim lights stay on for 5s.
2. There exists 1s delay between switching one light off and the other on.
3. The major-road light must stay on green for at least 30s in each polling cycle, but must respond to the sensor immediately after that.

1.2 Design choices and models

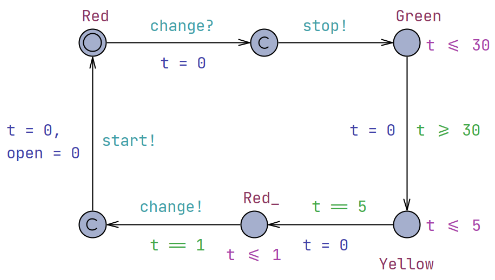
In order to model the intersection described, we decided to divide the system into for components: a major road, a minor road, a sensor and a car, to better tackle the problem. The division of the road into a major and a minor road was done in order to reduce the complexity of the road, allowing us to sync up the lights in both roads in a simpler way.

1.2.1 Major and Minor Roads

Given the previous description, we following major and minor road models were built:



(a) Model of the Major Road



(b) Model of the Minor Road

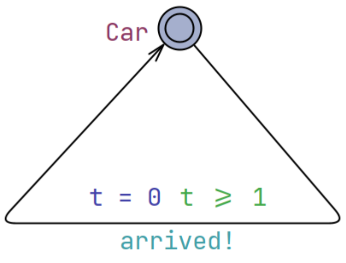
Figure 2: Models of the minor and major roads

The major road has 5 different locations: **Green**, **Green_**, **Yellow**, **Red_** and **Red**, being **Green** an initial state. If a car reaches the minor road i.e. synchronises with *waiting?*, then, if the major road had green signal for more than 30 seconds, it will turn yellow, otherwise it will continue green while it's internal clock (t) is less than 30 seconds, transitioning to yellow afterwards. It then stays yellow for 5 seconds before turning red. Following that, both lights stay red for 1 seconds, before the major road sends a synchronization (*change!*), opening turning the signal in the minor road green. Beside the clock already refereed, the major road also include a clock *open*, to allow us to make some verifications when benchmarking the system.

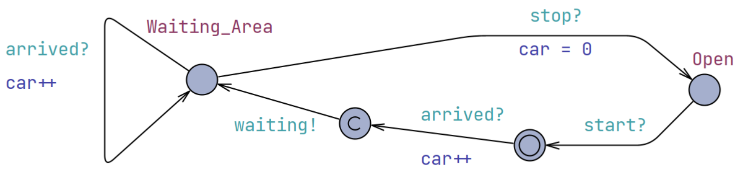
The minor Road has 6 different location: **Green**, **Yellow**, **Red_**, **Red** and 2 committed locations. Similarly to the major road, it has synchronizations to change the signal form red to green and to open the major road. However, this road has **Red** as it initial state, allowing the traffic to accumulate. When it receives the information to open through the sync *change?*, it will turn green, going before to a committed location with a sync *stop!* that informs that there are no more cars accumulating (because the signal is not green). However, since there isn't always traffic on the minor road, the signal only stays green for 30 seconds, following the same procedure to turn red as the major road, being the only difference one committed stated between **Red** and **Red_**, to allow traffic to accumulate, through the use of the sync *start!*. Once again, the clock *open* was used to verify and benchmark the system.

1.2.2 Sensor and Car

In order to simulate the arrival of to the minor road and the activation of a sensor, the following models were built:



(a) Model of the Car



(b) Model of the Sensor

Figure 3: Models of the sensor and car

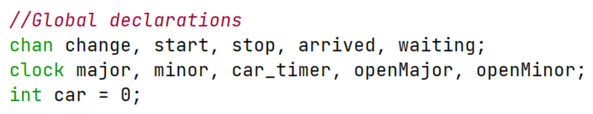
The car model's sole purpose is to inject cars onto the minor road with at least a one-second interval between them. This model has a single state and transition.

A clock (t) ensures at least one second between cars. When a car arrives, it synchronizes with the *arrived!* channel, notifying the sensor, which increments the car count.

The sensor has four states: **Initial State**, **Waiting_Area**, **Open**, and a Committed State. When a car arrives, it synchronizes with the *arrived!* channel. The sensor then increments the car count, transitions to the **Waiting_Area** state, and sends a *waiting!* synchronization to initiate the signal change process. While waiting for the green light, additional cars can arrive, increasing the total number queued. Since this was a simple model and the minor road was clear of heavy traffic, was assumed that all the cars were able to leave the road when the signal turned red. This is reflected by resetting the "car" variable. The information about the signal change is received through the *stop?* channel, as described in the minor road model.

1.3 Properties and benchmarking

In order to create the models previously described, the following global declarations were made:



```
//Global declarations
chan change, start, stop, arrived, waiting;
clock major, minor, car_timer, openMajor, openMinor;
int car = 0;
```

Figure 4: Declaration of variables used in the system

In the previous models, all the clocks were passed as arguments, and so, *major*, *minor* and *car_timer* corresponded to the clock *"t"*, used in the Major Road, Minor Road and Car, respectively. Similarly *openMajor* and *openMinor* correspond to the clock *"open"* used in the major and minor roads, respectively.

With all the declarations done we can finally test and benchmark our system, checking for reachability, safety and liveness properties, and for absence of deadlocks, using CTL formulae.

Reachability properties:

- **E<>Major.Green**: This property guarantees that there is at least one execution path where the major road has green light.
- **E<>Minor.Green**: There is at least one execution path were the minor road has green light.
- **E<>(Major.Green or Major.Green_) and major > 30**: There is at least one execution path were the major road light is green for more than 30 seconds.

Safety Properties:

- **A[ ] not (Major.Green and Minor.Green)**: The signals on the major and minor road are never green at the same time.
- **A[ ] Minor.Green imply minor <= 30**: The green light on the minor road can only be green for 30 seconds.
- **A[ ] Minor.Yellow imply openMinor >= 30**: If the minor road as yellow light, then the minor road had green light for at least 30 seconds.[1]
- **A[ ] Major.Yellow imply car > 0**: If the major light truns yellow, then there must be a car waiting in the minor road.
- **E[ ] Major.Green imply car==0**: There is one execution path were if the major road light is always green then there are no cars in the minor road.

Liveness Properties:

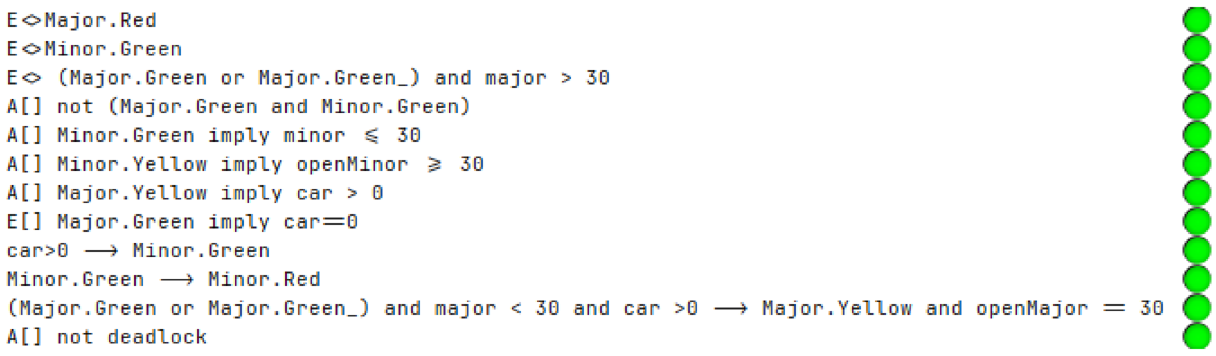
- **car > 0 –> Minor.Green**: If there are cars waiting they will eventually have green light.
- **Minor.Green –> Minor.Red**: If the minor road has green light, it will eventually turn red.
- **(Major.Green or Major.Green_) and major < 30 and car > 0 –> Major.Yellow and openMajor == 30**: If the major road had green signal for less than 30 seconds and there are cars waiting in the minor road, the light will only turn yellow after being green for exactly 30 seconds.

[1]This rule, together with the one above, guarantee that the minor road has green light for exactly 30 seconds.

Deadlock:

- **A[ ] not deadlock**: There is never a deadlock in the system.

We can check if these properties hold by running the Uppaal verifier with the CTL rules. By doing that we validate each one of them, which means that the system works flawlessly.



```
E<>Major.Red                                                                    ●
E<>Minor.Green                                                                  ●
E<> (Major.Green or Major.Green_) and major > 30                                ●
A[] not (Major.Green and Minor.Green)                                           ●
A[] Minor.Green imply minor ≤ 30                                                ●
A[] Minor.Yellow imply openMinor ≥ 30                                           ●
A[] Major.Yellow imply car > 0                                                  ●
E[] Major.Green imply car==0                                                    ●
car>0 --> Minor.Green                                                           ●
Minor.Green --> Minor.Red                                                       ●
(Major.Green or Major.Green_) and major < 30 and car >0 --> Major.Yellow and openMajor == 30 ●
A[] not deadlock                                                                ●
```

Figure 5: Overview and verification of the CTL rules in the modeled system

1.4 Conclusions

To conclude the first model, we have that with a rather simplistic approach we can model quite a robust system for traffic lights and guarantee that the system works by proof rather than by testing in the real world, which is not ideal for cyber-physical systems.

2 Second Part

2.1 A new approach to the problem

On the second part of the assignment we are tasked with modelling a system similar to the first but with sensors in all roads, that this, both lights are now dependent on sensor signals. For this effect we have sensors that output high, low or none values depending on the volume of traffic present.

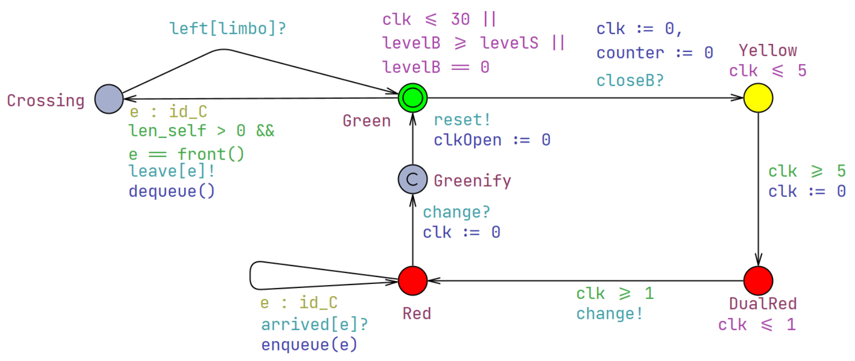
This approach aims to generalize the problem we modelled in part one, since the assumption that one road doesn't have the same level of traffic as the other which might prove to be too restricting for real world applications.

2.2 Design choices and Models

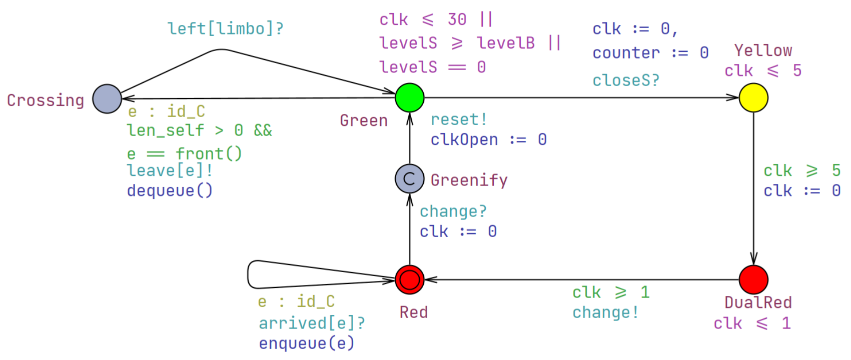
Our intersection got more complex with the added properties, as such our model will have to make some assumptions in order to keep computational load for benchmarking manageable, we will discuss this in greater detail bellow.

We made some changes to the model to accommodate the new sensors, as such we decided to incorporate the sensor in the road, as they should function as a homogeneous system and added a controller to process the data from the sensors. In the end we have four components: the major road, the minor road, a car and a controller.

2.2.1 Road



(a) Model of the Major Road



(b) Model of the Minor Road

Figure 6: Models of the minor and major roads

The major and minor roads now present with 5 working locations, with **Greenify** being only used for benchmarking, this locations are **Green**, **Yellow**, **DualRed**, **Red** and **Crossing**. It's important to note that now both roads are equal up to the initial state which must differ since there can never be two lights on **Green**.

Since both lights behave equally we will only explain the major since it applies to the minor as well.

When the light is **Green** it should stay on provided that, (1) it's internal clock doesn't exceed 30 seconds, this condition only provides a minimum value, (2) the traffic level on the major road is higher than that of the minor road and (3) not every car on the major road has cleared the intersection. This three conditions constitute the invariant in **Green** and are necessary for the properties we want our model to have, which will later be presented and explored.

The transition to **Yellow** will occur when the *controller* synchronizes with the road through the *closeB* channel, which is defined to be an urgent channel as the controller should dictate when to transition without waiting. As is mandated by the system requirements the **Yellow** stays on for exactly 5 seconds. After that, the light transitions to an intermediate state, **DualRed**, which serves to guarantee that both lights stay red for exactly 1 second, similarly to what happened in the first part. Concluding, the light transitions to **Red** and synchronizes with the other road to turn it **Green**, by means of the *change* channel.

We have defined how the lights work but not how the sensor is integrated in the road. For that we have that, when the light is **Red** cars can arrive and when the light is **Green** cars can cross the intersection. This is a simplification, since normally we expect cars to arrive independently of light color and that some may cross the intersection when the light is **Yellow**. Unfortunately adding all this increases the computational workload too much, without breaking the generalization we can obtain with this simpler model. Given this we have then defined a queue for each sensor, which keeps track of the number of cars and the traffic level. The latter is calculated from a combination of waiting cars and a bias from the number of cars that have crossed the intersection through the other road, to prevent that a single car will wait forever to cross, since eventually it will have enough 'weight' to force the light to transition.

2.2.2 Model of the Car

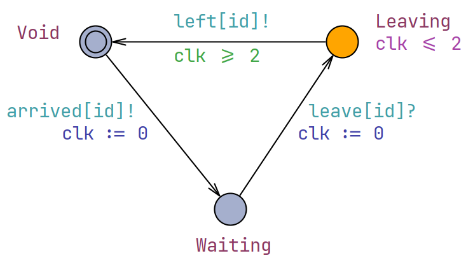


Figure 7: Model of the Car

The model of the car is rather simple as we only need to keep track of cars that are waiting at the lights and cars that are crossing the intersection. Given this our model needs only 3 locations, an initial location, **Void**, which represents the existence of a car which has not arrived at any light, and is in this location irrelevant, **Waiting** which is reached from **Void** by the *arrived* channel that synchronizes with the road to signal that a car has reached and it's waiting at a light. Lastly, it has a location **Leaving**, which makes use of the *leave* and *left* channels to signal a car crossing the intersection. In this location we establish that a car takes 2 seconds to cross the intersection to prevent every car from crossing the intersection instantly.

It's important to note that every car will enter a queue in the road where it is waiting for the light, being this mechanism used to simulate the sensor on the road and determine the traffic level.

2.2.3 Model of the Controller

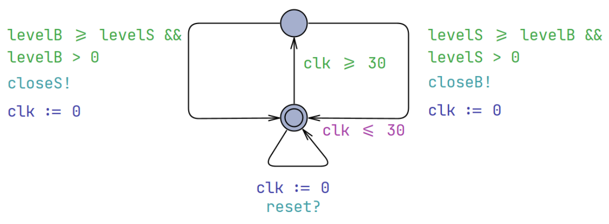


Figure 8: Model of the Controller

The controller is used to ensure that a light must stay **Green** at least 30 seconds before being able to transition to **Yellow** and to dictate when a light must turn to **Red**. To do this the controller checks the traffic levels calculated by the sensor and, if there is an overturn of intensity, the light must transition to allow the traffic from the other road to flow.

2.3 Properties and benchmarking

Since there are a lot of declarations involved in this second part, we will omit them, since the models already generalize the idea behind the system. We can now test and benchmark our system, checking for reachability, safety and liveness properties, and for absence of deadlocks, using CTL formulae.

Reachability properties:

- **E<> major.Red**: There is at least one execution path where the major road has red light.
- **E<> minor.Green**: There is at least one execution path where the minor road has green light.
- **E<> major.Green and clkB > 30**: There is at least one execution path where the major road has green light for more than 30 seconds.
- **E<> major.Green and clkB > 30**: There is at least one execution path where the minor road has green light for more than 30 seconds.
- **E<> levelB == 2 and levelS == 2**: There is at least one execution path where both the major and minor roads have high traffic.
- **E<> major.list[0] == N-1**: There is some state where the last car N-1 car is waiting in the major road.
- **E<> minor.list[0] == N-1**: There is some state where the last car N-1 car is waiting in the minor road.
- **E<> levelB == 2 and levelS == 2 and minor.basicLevel[2] == 1 and minor.Green**: There is a case where, if a car is waiting in a lower priority road (minor road), it will have green light.
- **E<> levelB == 2 and levelS == 2 and major.basicLevel == 1 and major.Green**: There is a case where, if a car is waiting in a lower priority road (major road), it will have green light.

Safety Properties:

- **A[ ] major.Yellow imply clkOpen >= 30**: If the major has yellow light, it must have been open for at least 30 seconds.
- **A[ ] minor.Yellow imply clkOpen >= 30**: If the minor has yellow light, it must have been open for at least 30 seconds.
- **A[ ] not(major.Green and minor.Green)**: The major and minor road can never have green light at the same time.
- **A[ ] major.Yellow imply levelS >= levelB**: The major can only have yellow light once the traffic on the minor road is equal or heavier than the traffic on the major road.
- **A[ ] minor.Yellow imply levelB >= levelS**: The minor can only have yellow light once the traffic on the major road is equal or heavier than the traffic on the minor road.
- **A[ ] forall (i:id_C) forall (j:id_C) Car(i).Leaving && Car(j).Leaving imply i == j**: There is never more than one car leaving the intersection at the same time.
- **E[ ] levelS == 0 and major.Green**: If the minor road never has traffic, then the major road will always have green light.
- **E[ ] levelB == 0 and not(major.Greenify)**: If the major road never has traffic, then the minor road will never leave green light (once it reaches there).
- **E[ ] levelS == 0 and levelB == 0 and major.Green and minor.Red**: If there is no traffic, then the lights never change.

[2]The basicLevel represents the traffic level without the bias introduced

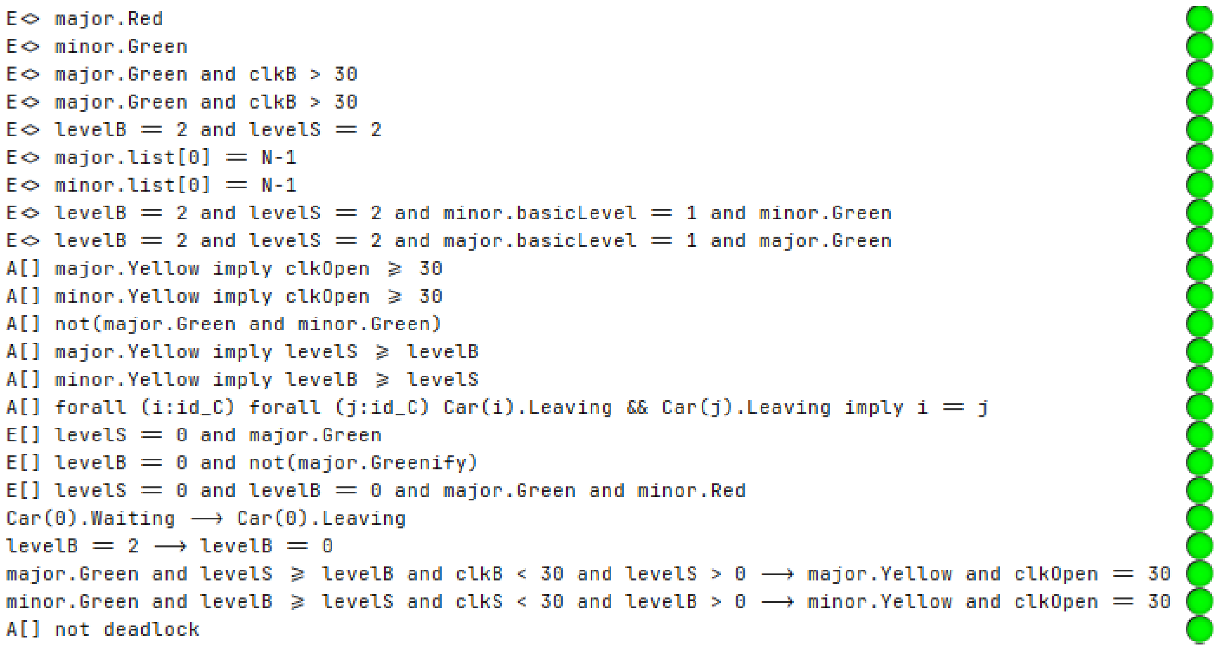
Liveness Properties:

- **Car(0).Waiting –> Car(0).Leaving**: If there are cars waiting they will eventually have green light.
- **levelB == 2 –> levelB == 1**: If there is heavy traffic, it will eventually dilute.
- **major.Green and levelS >= levelB and clkB < 30 and levelS > 0 –> major.Yellow and clkOpen == 30**: If the traffic in the minor road becomes heavier or equal to the traffic in the major and is greater than 0 while the signal has been green in the major road for less than 30 seconds then the light will eventually turn yellow once it as been green for exactly 30 seconds.
- **minor.Green and levelB >= levelS and clkS < 30 and levelB > 0 –> minor.Yellow and clkOpen == 30**: If the traffic in the major road becomes heavier or equal to the traffic in the minor and is greater than 0 while the signal has been green in the minor road for less than 30 seconds then the light will eventually turn yellow once it as been green for exactly 30 seconds.

Deadlock:

- **A[ ] not deadlock**: There is never a deadlock in the system.

We can check if these properties hold by running the Uppaal verifier with the CTL rules. By doing that we validate each one of them, which means that the system works as intended.



```
E<> major.Red                                                                        ●
E<> minor.Green                                                                      ●
E<> major.Green and clkB > 30                                                        ●
E<> major.Green and clkB > 30                                                        ●
E<> levelB == 2 and levelS == 2                                                      ●
E<> major.list[0] == N-1                                                             ●
E<> minor.list[0] == N-1                                                             ●
E<> levelB == 2 and levelS == 2 and minor.basicLevel == 1 and minor.Green            ●
E<> levelB == 2 and levelS == 2 and major.basicLevel == 1 and major.Green            ●
A[] major.Yellow imply clkOpen >= 30                                                 ●
A[] minor.Yellow imply clkOpen >= 30                                                 ●
A[] not(major.Green and minor.Green)                                                 ●
A[] major.Yellow imply levelS >= levelB                                              ●
A[] minor.Yellow imply levelB >= levelS                                              ●
A[] forall (i:id_C) forall (j:id_C) Car(i).Leaving && Car(j).Leaving imply i == j    ●
E[] levelS == 0 and major.Green                                                      ●
E[] levelB == 0 and not(major.Greenify)                                              ●
E[] levelS == 0 and levelB == 0 and major.Green and minor.Red                        ●
Car(0).Waiting --> Car(0).Leaving                                                    ●
levelB == 2 --> levelB == 0                                                          ●
major.Green and levelS >= levelB and clkB < 30 and levelS > 0 --> major.Yellow and clkOpen == 30 ●
minor.Green and levelB >= levelS and clkS < 30 and levelB > 0 --> minor.Yellow and clkOpen == 30 ●
A[] not deadlock                                                                     ●
```

Figure 9: Overview and verification of the CTL rules in the modeled system

2.4 Conclusions

To end we should highlight that, even though the difference between the first and second part not seeming significant, the model we need to simulate the system is a lot more complex. This should not be attributed only to the difference in systems, since we have created a much more resilient intersection in the second section as seen in the **Properties and benchmarking** section. The higher complexity of the model might mean a more robust cyber-physical system but, unfortunately, it increases the computational workload to a point where we must make compromises in real-world fidelity, as mentioned in **Design choices and Models** section. Another compromise we had to make is that, in order for the properties to be verified in a relatively short amount of time, the total number of cars in the system had to be kept low, because of the increase in state space that comes with a higher number of cars in the system.