UNIVERSITÉ LIBRE DE BRUXELLES

Formal verification of computer systems

Traffic Light Model Checker

DESCLEFS Maxime, GERARD Pierre, HEREMAN Nicolas, MOULART Walter, SCHEMBRI Julian

May 10, 2016

CONTENTS

1 INTRODUCTION

In this section, we will discuss about the goal of the project, the different component used, the assumptions made and model chosen.

1.1 GOAL OF THIS PROJECT

The aim of this project was to design and implement an embedded system. To do so, we decided to create an Arduino controlled traffic light crossroad. In this system, many hypothesis regarding the crossroad properties will be made and will be explain more deeply in the next sections.

We also decided to join the Embedded Software project and the Formal verification of computer systems project.

1.2 Crossroad assumptions

We first had to choose a traffic crossroad to implement. We strongly took inspiration from the Fraiteur crossroad (Boulevard du triomphe and Beaulieulaan) near Delta metro station.

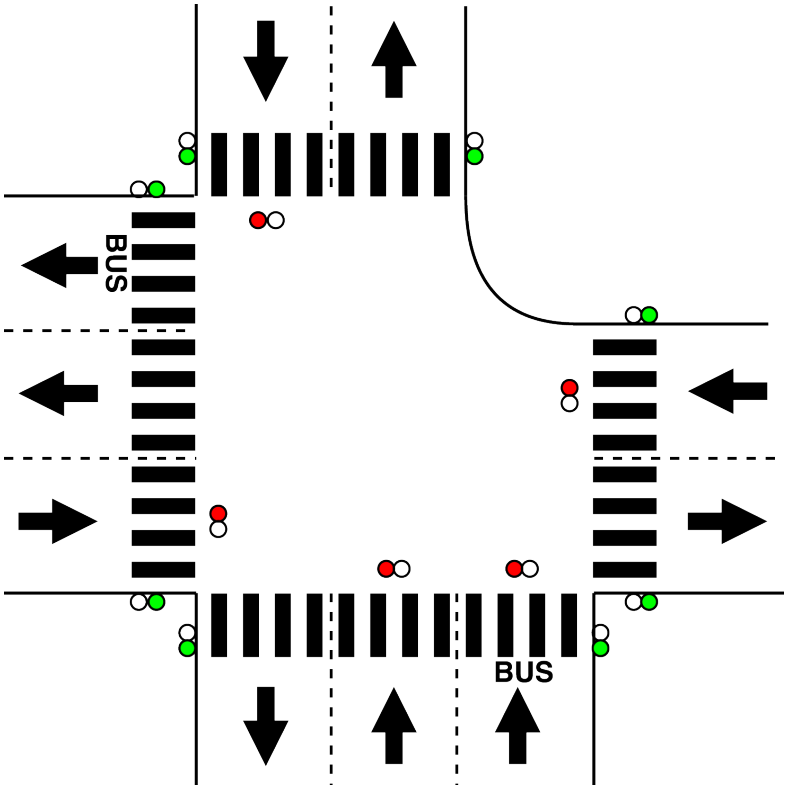

Figure 1.1: The crossroad model chosen

Before defining the different actors and components in our system, some important assumptions about our model have to be taken:

- The different vehicles will respect the Highway Code. This for example means that if a car is turning left and there are car on the opposite direction going straight, the car will stop to let the other pass and will turn after.
- At any time, a pedestrian can push a button to cross the street. There is no priority for pedestrian that would push the button many times in a row.
- The pedestrians will cross the street within the given time. This also means that they follow the laws and rules.

- All the pedestrians light are connected, that means that when pedestrian can cross, all other lights are red.
- As for the pedestrians, the vehicles will also cross the street within a given time.
- A bus can arrive at any time.
- If there is a bus and a pedestrian arriving at the same time, bus call take the upper hand.

1.3 Actors

In this section, the different actors interacting with the system will be reviewed. Those include pedestrians, buses and traffic lights.

1.3.1 Pedestrians

As shown on Figure 1.2, there are four different crosswalks in our model. To make a call for the pedestrians to cross, a push button has been implemented on the controller. This means that when one pedestrian has pushed one of the button of the crossroad, the call has been made for all the pedestrians from all the sides of it. When the call is accepted by the system, the pedestrians can walk freely in the crossroad because all the lights for the cars and the buses will be red. This also means that if no pedestrians are pushing the button, meaning that no pedestrians are present, their traffic lights will always be red so the cars will always be able to drive through the crossroad. To summarize, when the pedestrians can cross, no other vehicles can drive and have thus to wait for the red light to be green.

1.3.2 Buses

The buses in our model can only go from the South path of the crossroad to the West part of it. Those buses will be created thanks to an outside event, like a pushbutton or a proximity detector. When a bus is coming, it will have a priority on every other actors. This means that when a bus has arrived in the crossroad, because it has to go through all the other lanes, the traffic lights for everyone have to be red.

1.3.3 Traffic Lights

In our embedded systems, traffic lights can be green or red.We purposely remove the orange light from a common belgian crossroad model because it allows us to avoid having too much cables and leds. We consider that when a traffic light is green, cars are going through the crossroad and respect the Highway Code. Cars are thus, in our model, not really an actor since they are not created, we just assume them to pass through the crossroad when their traffic lights are green.

There are some assumptions we made about the traffic lights that are important, because we wanted fairness in our system. For example, even if the pedestrians can always push the button to have the ability to walk through the crosswalk, they have to wait that at least all other traffic lights for the cars have been green once between two green light. This means that, if no

buses come and pedestrians are always pushing the button, the crosswalk with allow the cars of each side to go trough once before the pedestrians can cross again. Thanks to that, each elements in the model can't have only red light.

Another assumption is, like in common belgian crossroad, that when the South light is green, the North light is also green. On the contrary, when the East light is green, the West light is green. This means that South-North and East-West direction will have synchronized lights.

1.3.4 Connection with Formal Verification

We decided to combine those two projects together because they follow the same idea and we believe the two classes to be complementary and thus doing a merged project will allows us to study all side of the project, from a model to an implementation trough verification.

For the embedded project, we had to modelize an environment, and generate a controller based on that model to avoid unwanted situations through winning strategies, computed with timed games.

For the formal verification course, a more formal approach was taken. We also had to modelize the system, but also to verify that the model was compliant to some specific properties.

1.4 Arduino

To test our critical system on a real life environment, we decided to implement it with an arduino, leds and button.

We believe it to be a nice exercice because first, for most of us, it was the first time we build something using a micro-controller. Then as we witness it, micro-controller are subject to external unwanted event, like a inducted current in a cable or a button with a badly made pull-up resistor that create an event that should not have taken place. That forces us to create a strong system that can handle all of that.

2 Prerequisite

In this section, important theoretical concepts will be explained because they must be clearly understood to have a successful project. We will thus here discuss about timed automata, temporal logic and their properties and game theory to reach a winning condition.

2.1 Timed Automata

A timed automaton is essentially a finite automaton (that is a graph containing a finite number of nodes and labeled edges) extended with real-valued variables. Those automata run on an infinite words as input and they accept timed languages. The variables that exist in the model can represent logical clocks of the system that will be increased synchronously with the same rate. It is possible to add constraints called **guards** on the edges to create restriction on our automaton. A transition could thus occur when the value of the clock satisfy the guard's condition on the edge.

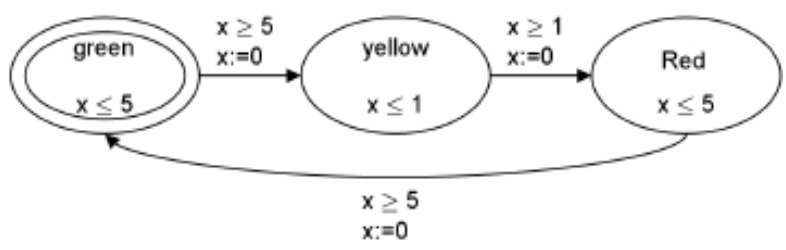

Figure 2.1: A timed automaton example

The example of Figure 2.1 represent a basic light example where the light goes from the green state to the yellow state if the clock is higher or equal than 5. It then reset the clock to 0 and stays in the yellow state for a time lower or equal to one. When the clock has an higher value, it switches to the red state and so on.

2.2 Temporal Logic and Properties

The aim of temporal logic is to become a formalism of specification and verification of some properties in reactive systems. In our case, two different kind of logic will be of interest: **Linear Temporal Logic (LTL)** and **Computation Tree Logic (CTL)**. Those two kind of logic are only concerned with infinite paths. They for example allow us to make statements, such as "Tomorrow, something will eventually be true", something that was not possible with first order logic. Thanks to this, we can define specific properties our system has to comply to.
There are of course differences between the two. In the first kind of logic, the LTL one, we will look at the traces of executions, meaning that we will look at the states we reach in a linear manner. If we look back at Figure 2.1, the traces would be:
$Green \rightarrow Yellow \rightarrow Red \rightarrow Green \rightarrow Yellow \rightarrow Red \rightarrow Green \rightarrow ....$
In the other kind of logic, the CTL one, we will use a branching time semantic instead of a linear semantic. Here, the different execution (or computation) trees will be considered for all branching possibilities.
An example of propriety we can verify thanks to CTL and that is not possible to verify in LTL is "do all executions always have the possibility to eventually reach a certain state". Indeed, intuitively it would require branching to verify it.
The following temporal properties can be expressed using either CTL or LTL:

- Safety: The system never reaches an unwanted state,
- Liveness: Ensure that our system will reach a certain state infinitely often, meaning that no deadlocks can occur,
- Persistence: Ensure that a property eventually holds forever,
- Fairness: Everyone is at some point satisfied.

2.3 Game Theory and Timed Games

Game theory can be defined as the formal study of decision-making where several players must make choices that potentially affect the interests of the other players[1]. As when playing a game, we would want to find a winning situation. In our case, there will be two different type of players. The first one will be the controller, something that we can control and define, while the second one, his opponent thus, will be called the environment.

The traffic lights will be the first player (the controller since we can control it) while the buses and the pedestrians will be the second player. Since the buses and pedestrians are playing when an external event occurs (a button that is pushed), they will be the environment. The main goal of the controller will be to find a winning strategy in this game, being a sequence of moves that will inevitably lead to the wanted outcome. In our case, we would want the lights to respect the properties that we have defined, even when pedestrians want to cross the street or buses want to go through the crossroad. If at some point all the traffic lights for the cars are green, it would mean that the system is broken and that the controller has lost the game. This is a situation we would not want to happen.

We here talk of **Times Games** because time is an important part of the game and the rules, bringing a real-time dimension. An example of a timed game can be a chess game, where each player has to make a move within a given time. In our case, the controller that is the traffic light has to keep the buses, the cars and the pedestrians safe while taking into account the time constraints on the traffic lights.

3 Tools

In this section, we will describe the tools we use and have used to create our system.

3.1 Arduino Controller

We had the idea not to only test our code in an application, but also to see live what is happening in our system. From that idea, we decided to create thanks to an Arduino Due controller our traffic lights system. In this controller, we placed LEDs on the boards to imitate the green, orange and red lights of the system. We also added a button for the pedestrians to imitate a real call as we would want to go through a crossroad and another push-button to generate the arrival of a bus in the system. There are also what we call information LEDs that can be lighten up if we arrive in a state that is problematic. This for example means that if all the lights for the cars are green at the same time, there would be a problem in our controller and crashes will thus happen. In this sense, we took some proprieties we modeled for the Verification project and we incorporated them into our embedded system.
To do so, we wrote the code needed for the good functioning of our system in C.

[1] `http://www.cdam.lse.ac.uk/Reports/Files/cdam-2001-09.pdf`

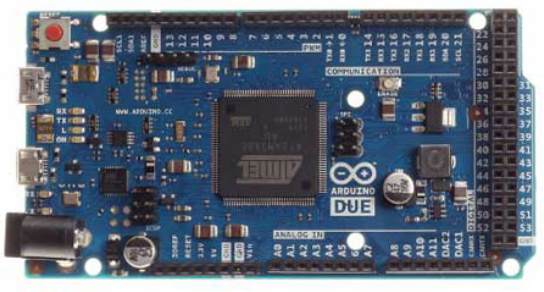

Figure 3.1: Arduino Due used for the embedded system

3.2 UPPAAL

Uppaal is a tool box for validation (via graphical simulation) and verification (via automatic model-checking) of real-time systems. The idea is to model a system using timed automata, as defined in Section 2.1, simulate it and then verify properties that we can define on it. Those automata will be synchronized through channels. Because Uppaal is based on timed automata, clocks will be an important thing in this software. Indeed, the time will be handle thanks to those clocks. We will see that it will be possible to test the value of a clock or even to reset it. Also, clocks will progress in the whole system at the same time.

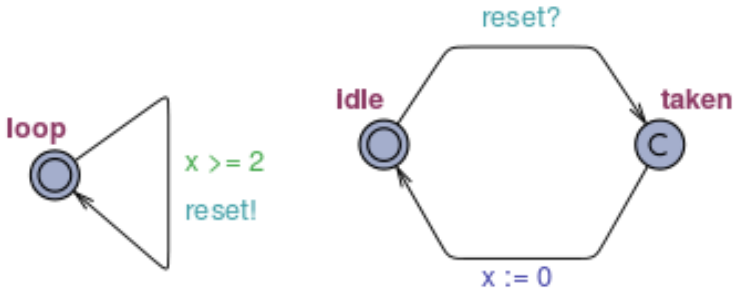

Figure 3.2: Basic example made with UPPAAL

In Figure 3.2, two different elements can be seen. On the left part, there is a loop that will, when the clock x is higher or equal to 2, trigger a reset transition. On the right part, we have a system that wait for a reset. When the trigger has been done by the left component, it is send to the right component thanks to the channels between them. If the reset has been done, we transit into the taken state before going back to the Idle state and resetting the x clock to 0. This program then repeat itself.

3.3 UPPAAL TiGa

Tiga is an extension of Uppaal, allowing timed games. The main difference with Uppaal is the possibility of having transitions taken non-deterministically by the environment. Indeed, while in UPPAAL everything is defined by the controller, in TiGa, the environment can also play. Since it's a game between the controller and the environment, we would want to find a winning strategy. In our case, as said in Section 2.3, the controller will be the traffic lights while the environment will be the buses and the pedestrians.

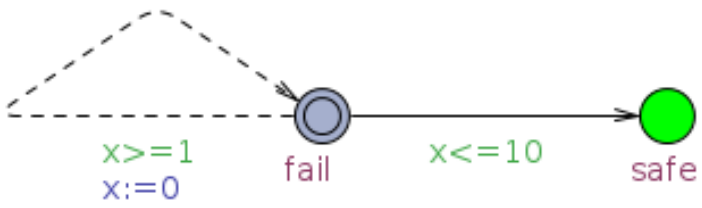

Figure 3.3: Basic example made with UPPAAL TiGa

In this example, the environment is represented by the cut lines. We start from the fail state and two different states can be reached. Either we go back to the left state because the environment decided to take his transition and to reset the clock x to 0, either to go to the safe state if the controller decided so.

4 Model Checking

4.1 Introduction

To build our final model, we decided to use an incremental way of construction, meaning that we started from a Minimum Viable Product that we develop more a little bit more at each step. We did verify each one and that way we are sure we have a correct base and can identify issue more easily.

At the end of the day, we did it in 4 different steps. We will now, without entering in the details for the first steps, explain what can be done there. The final step will be deeply explained since it is the most complex one.

4.1.1 Step 1: Basic Traffic Lights system

In this first basic step, we created the model in Uppaal of a simple controller sending a pulse every 3 seconds to two cars traffic lights. There is no pedestrians and no buses in this step. Every 3 seconds, the South and North traffic lights switch from green to red while the East and

West traffic lights switch, on the opposite, from red to green.

Thanks to this basic example, we understood Uppaal basics and notably how it is possible to send messages from one component to an other component thanks to the channel message passing system.

In this system, it is possible to check whether our first traffic light is green and the other is red at certain time and vice-versa. Thus we check if we never reach a state where our system is in an impossible and dangerous state where both traffic lights would be green.

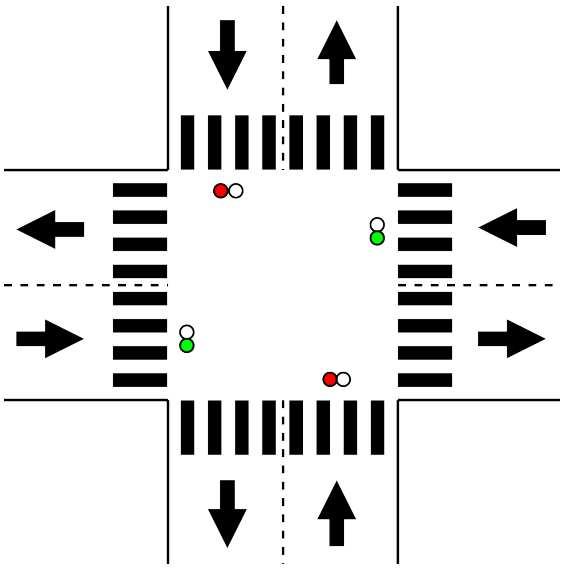

Figure 4.1: Arduino Due used for the embedded system

4.1.2 Step 2: Pedestrians

In this step, pedestrians were now added and many different things had to be changed to allow accommodate this new change.

When a pedestrians call was send by the environment, the car's traffic light that was green had to become red and the other car traffic light had to stay red. When those two traffic lights are red, the pedestrians traffic light could be green. Because we also wanted fairness in our system, two assumptions were here created.

1. When the pedestrians traffic light is green, we have to remember which car traffic light was previously green. This will be useful for the next assumption.

2. We wanted fairness between the actors, even if pedestrians have the priority. The problem at the beginning was that starvation could happen if, when the pedestrians lights are green, we always switched to a specific car's traffic light. This would mean that

the other car's traffic light would remain red if pedestrians are always calling. We thus added what we called a "delayed" call where, even if the pedestrians are always pressing the button, the two cars lights have to be green at least once before the pedestrians could have their lights green again. This is why the first assumption is useful, we have to remember which car's light was previously green so the cars in the other side of the crossroad don't always have to wait 2 times to be green again.
As an example, if the East-West car's traffic light is green and a pedestrian pressed the button, after the pedestrians lights switch from green to red, the North-South car's traffic light is switched to green so they do not have to wait another time.

In this step, more properties were checked. We for example check if, after a transition, the value of a component always reaches the enquired value. We also check, as for the first one, that certain states can never be reached in our model (all traffic lights are green for example).

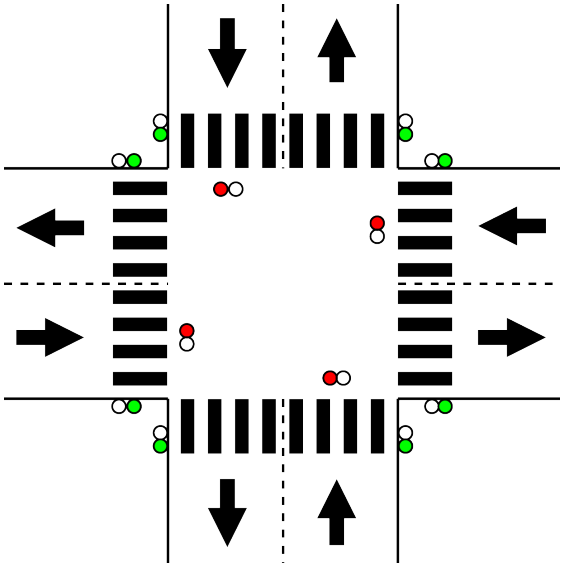

Figure 4.2: Arduino Due used for the embedded system

4.1.3 Step 3: Buses

In this step, we added the buses generated by the environment. The idea here is that the buses have priority on the rest of the traffic.
When a bus is generated, the next traffic light that has to be green is the one of the bus. When the bus is crossing, all other lights are red because the bus will go through all the crossroad.
Even if the pedestrians have called to have the green lights before the bus, if a bus is generated between the call and the pedestrians green light, we will first give the green light for the bus, then the pedestrians, then the cars. This mean that the priority hierarchy is *Buses-Pedestrians-Cars*.

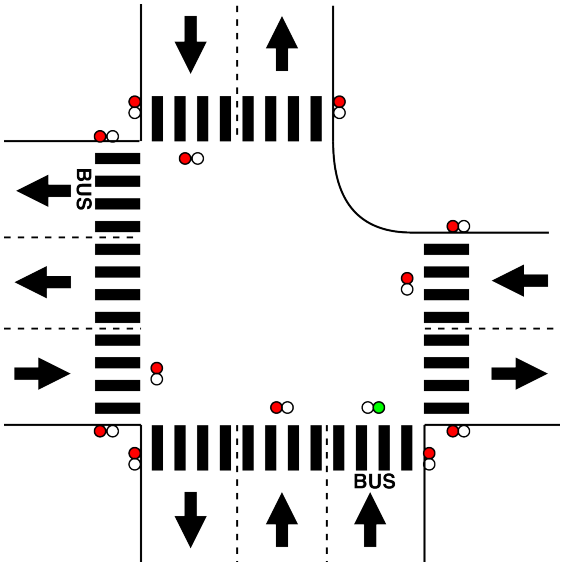

Figure 4.3: Arduino Due used for the embedded system

4.1.4 Step 4: Yellow Light

For this step, we kept the same model done in step 3 but we added a yellow light. This means that, as in the real life, our cars traffic lights will switch from green to yellow to red.
This is thus our final model developed in Uppaal TiGa.

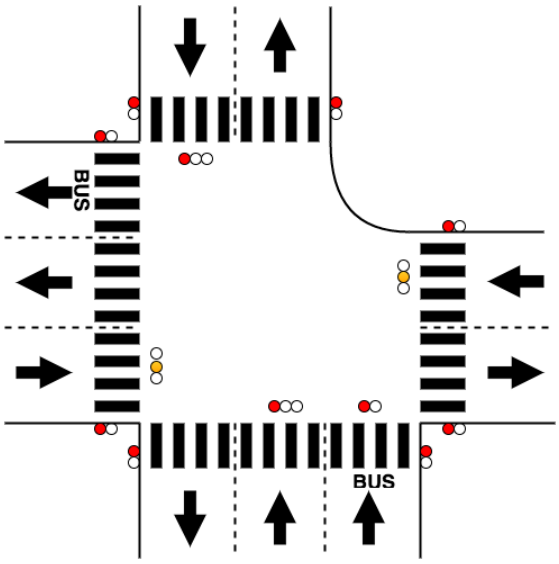

Figure 4.4: Arduino Due used for the embedded system

4.2 Timed Automatons

In order to formally model the system, we use several timed automatons from Uppaal. While some of those automatons are actors in the environment, the others are part of the controller. The theory tells us that, when the environment is playing, we, humans, cannot decide the action that will be taken. For example, buses and pedestrians arrivals are unpredictable and uncontrollable, so they belong to the environment. The actors which implement the solution for avoiding crashes belong to the controller. They interact with the environment in order to avoid crashes and thus they aim to find a winning strategy.
We will now give the implementations explanations of the environment and controller actors. In the first one, we will take a look at the pedestrians and buses generators while in the second one, we will look at our traffic light system that tries to find a winning situation (no crashes).

4.2.1 Environment

Crosswalk
We will here talk about the mechanisms of our crosswalk generator done in our step 4 from Section 4.1.4.
There are several elements that can be seen on Figure 4.1.

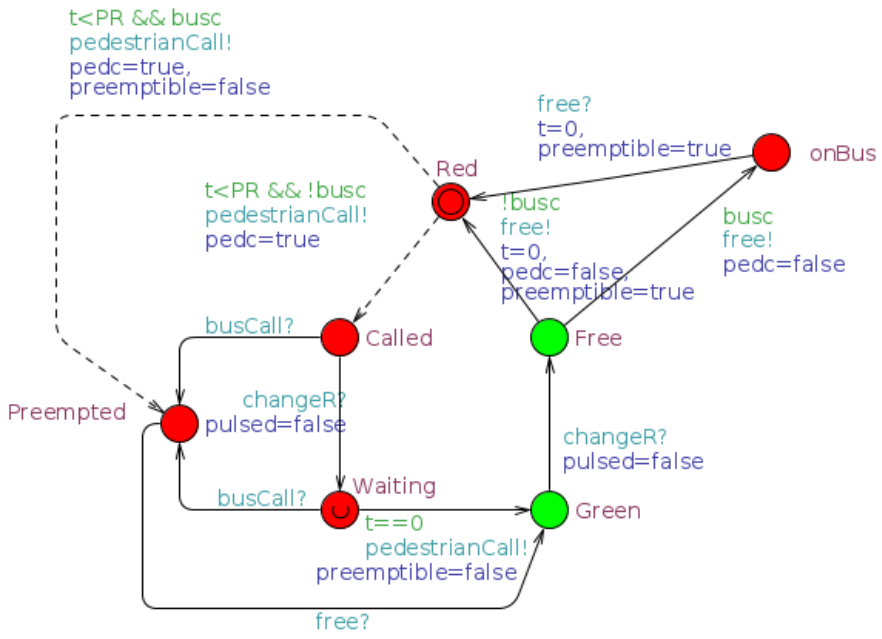

Figure 4.5: Pedestrians generator

It can first be seen that no orange lights are present here. Indeed, as it was a pedestrians traffic lights in Brussels, there is only the red and green light. We thus decided not to add an orange

light for this component.
When the environment play, different states can be reaches. Because we said the bus had total priority, different cases can happen. We will first explain the easiest case that can occur. When there is a call for the light to be green, if no buses are generated, we will reach the waiting state. In this state, we will wait for the cars traffic lights to be red, and when it is the case, the pedestrians light can be green. Of course, it is not that simple, since the buses have priority. Indeed, when the fire is in the red state, a transition going to the preempted state ca happen if there is already a bus. When the bus is gone, then the pedestrians lights can be green.
Another problem can still happen when the call has been made for the pedestrians light to be green. Indeed, if a bus arrives, it must have the priority on the light. This is why we check, until we allowed the green light, if a bus has been generated. If it is the case, we have to wait for the bus to leave the crossroad and then the pedestrians light can be green.
When the green light has to be switched red again, we will first go in a free state where we will check again if another bus has arrived. If an other bus arrives, we let him cross again before releasing the light and then, in any cases (bus or not bus), we go back to the basic red state.

BUS
We will here talk about the mechanisms of our bus generator done in our step 4 from Section 4.1.4.

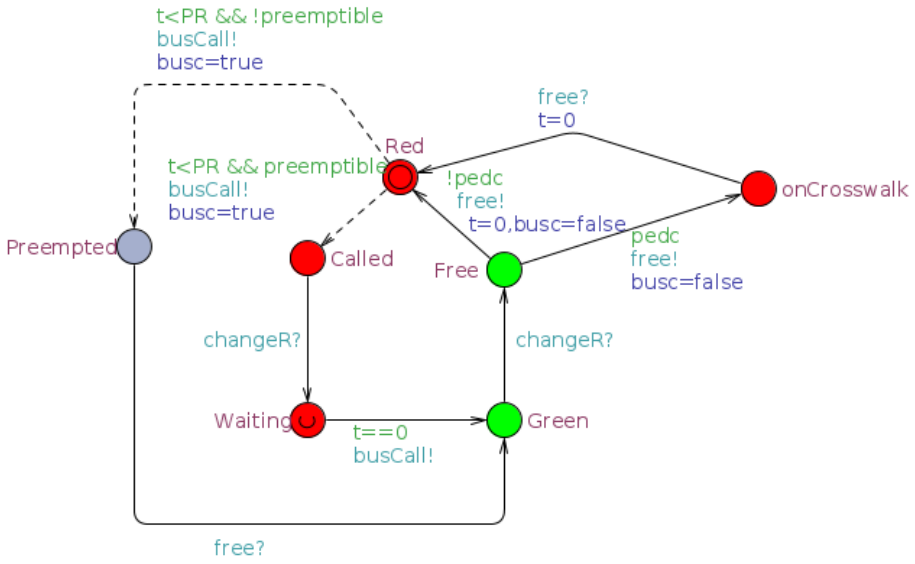

Figure 4.6: Buses generator

This generator follows the same idea as the pedestrians generator. Since there is nothing that has a higher priority than the bus, the model is easier that the one of the pedestrians. We

initially start on a red state. If a bus is called (read generated), we wait for the light that is green on the system to be red. When it is the case, the bus light has to be turn to green. We then also check if pedestrians were generated to allow them to have their light green after.

Those were the two actors controlled by the environment.

4.2.2 CONTROLLER