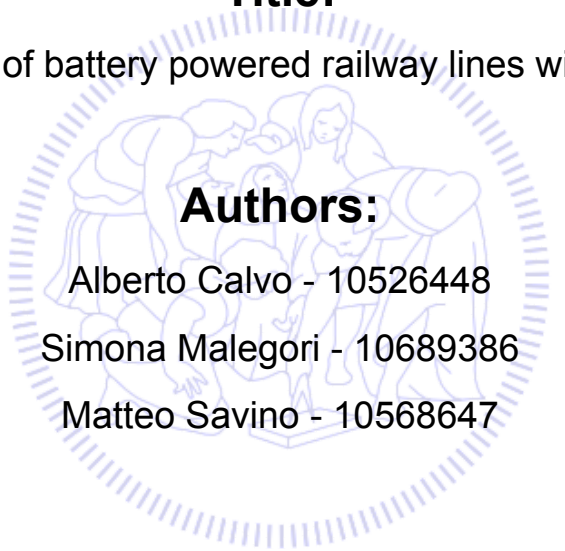
POLITECNICO DI MILANO 1863

Course:

Formal methods for concurrent and real-time systems

Title:

Model checking of battery powered railway lines with UPPAAL tool



Authors:

Alberto Calvo - 10526448

Simona Malegori - 10689386

Matteo Savino - 10568647

Professors and supervisor:

Prof. Livia Lestingi

Prof. Pierluigi San Pietro

Academic Year:

2021-2022

Release date:

11/06/2022

Table of contents

1 Summary

Many states adopted battery-powered trains to reduce greenhouse gas emissions. Obviously, other states are also planning to develop fully electric trains. Thus, in this document, we show in detail the implementation of a battery-powered railway line model. The aim of our work was to obtain a working model suitable for a simulation that respects some fundamental properties like reaching destination on time and not running out of battery before reaching the next station. We used the UPPAAL tool for our purpose and we managed to create a model that respects all the requirements. Moreover, the model can be easily expanded to increase the number of trains and stations.

2 Introduction

The main actors in a railway line system are:

- **Stations**: They are in charge of responding to trains' requests to enter and occupy one of the available tracks so that they can recharge batteries.
- **Lines**: their job is to establish how stations are interconnected.
- **Trains**: They move from station to station along a given line and transport passengers. They also have to make sure that they stop in each station a sufficient amount of time to allow passengers to get off the train.

In the next sections, they will be described thoroughly and the main choices we made will be presented.

3 Design choices and assumptions

In this chapter, we describe the design choices and the assumptions made in modeling the entities of the model. For future reference, the distance is expressed in meters and the time in minutes. We took this unit of measures to avoid using hours because that would have required us to use double numbers, e.g. when the train waits to resubmit a request to a station the minimum time it has to wait would have been an hour, or the same concept is valid for the time necessary for passengers to get off the train.

3.1 Railway Line

We decided not to create an automaton for the lines. Hence, we designed them as a matrix whose rows are arrays of subsequent station identifiers. Each row models a single line. This choice was made to allow trains entering a new station to easily retrieve their current station and their new destination based on the line they belong to.

Each line is also characterized by the distance between each pair of stations and the maximum time a train could spend to travel between them. We assumed that these values are equal in both directions. Thus, we employed two squared matrices which are also

diagonal. They can be accessed by using two station identifiers. If two stations are not directly connected there will be a value equal to minus one.

3.2 Station

A station is characterized by an identifier and communicates with trains through channels. Mainly, it has to listen to trains requests for obtaining the permission to enter and occupy a track. Once a request has been received the station will immediately respond. Each station also has a maximum number of tracks, which is the maximum number of trains that the station can host and a counter for the trains currently recharging inside. If the number of trains occupying a track is less than the total number of tracks then the station will allow other trains to enter by responding positively. Otherwise it will respond negatively.

Moreover, we included some other relevant parameters such as the count of the trains that are waiting to enter the station. Actually, they are not used directly by the station. Trains check that value to decide whether they can stay inside to reach their max charge or whether they have to exit that station in advance to free some space.

The behavior of stations is described by the automaton presented in Chapter 4.2.

3.3 Train

The train is the main entity in our model. A Train has a starting station, which may be different from train to train. We assume that every train has to move on the line a fixed amount of times (we call them laps) at the end of which it must stop as soon as it is back at its starting point. Trains' batteries have a different maximum capacity and at the beginning they may start with a charge that is not the maximum one. This choice has been made to take into account a real scenario. By design when a train is in a station, it recharges with a predefined global recharging rate.

Each train keeps a record of the current station and of the next station. that define the position of the train on the line and its direction. These are fundamental information so that the train always knows where it is on the line and which is the station it has to request permission to access. That is also important once inside a station for being able to check if it is necessary to leave the station in advance to free space for trains that are waiting outside because the station is currently full. Clearly, only trains with enough charge to complete their trip in the worst scenario are forced to leave beforehand. The algorithm that updates this data can be found in the Appendix.

Each train must reach the next station within a maximum time called maximum delay. In order to do so, it keeps track of the ride time which is the time measured from when it enters a station and when it reaches the next station on the line. Within the maximum delay it is important to consider that a train requires enough time to let passengers get off the train, obtain the charge necessary to reach the next destination and ask multiple times to a congested station the permission to enter. It is important to remember that a train which

received a prohibition to enter has to wait a global predefined time before submitting a new request and moreover, during this time the train consumes its charge. Thus it's important to take all these factors into account.

As previously stated, trains do not travel indefinitely every day but their daily duty involves performing a number of complete laps in the line. A lap is the trip from the starting station to the terminal station and then, back to the initial one. At the end of the "day", a train is assumed to be moved into a train shed in the initial station and thus it is removed from the station count.

The behavior of a train is described by the automaton presented in Chapter 4.3.

4 Components description

As we stated before there are two main components in our model and they are the actors of the possible scenarios: trains and stations. Along with them there are channels that are used by the two actors to communicate.

4.1 Channels

Before moving to the description of the main components it's important to define the channels necessary for the communication between trains and stations. They are fundamental for understanding the behavior of each component.
Actually, it's worth noting that we did not use simple channels but arrays of channels, which are recognizable by the squared parenthesis, so that every station can be contacted by the trains properly. For simplicity, squared parenthesis will be omitted in other sections of the document.

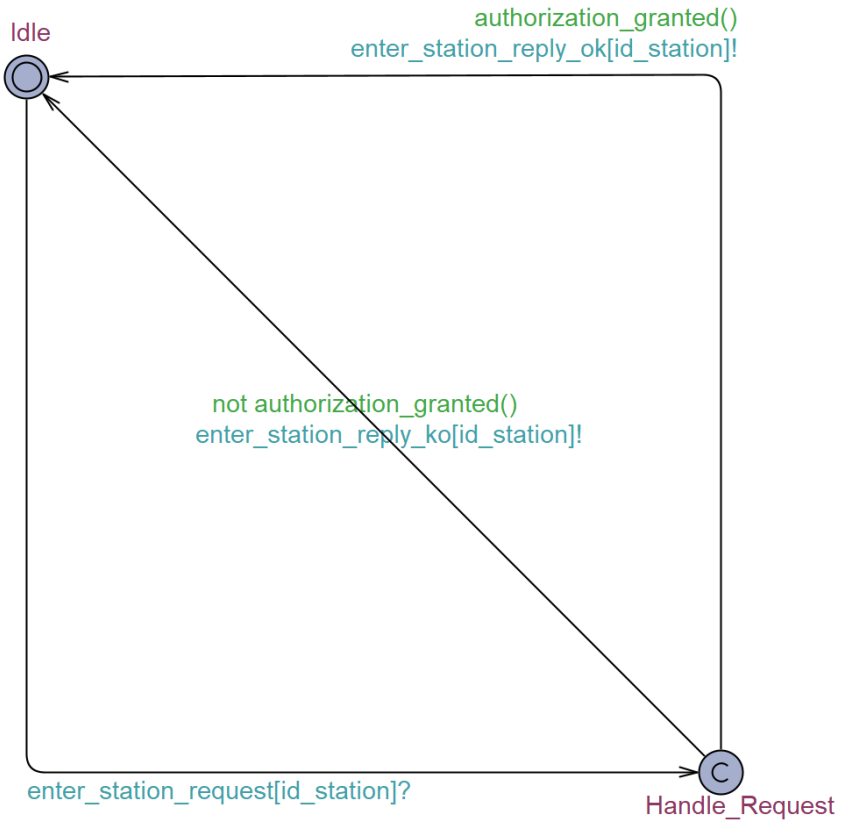
List of channels:

- ***enter_station_request[NUM_STATIONS]***: it is used by the trains to ask for permission to enter and occupy one of the tracks of a station.
- ***[urgent] enter_station_reply_ok[NUM_STATIONS]***: a station uses this for replying positively to the trains that previously sent an *enter_station_request*. This channel is urgent due to the fact that we need the trains to take the edge of the response as soon as possible.
- ***[urgent] enter_station_reply_ko[NUM_STATIONS]***: a station uses this for replying negatively to the trains that previously sent it an *enter_station_request*. This channel is urgent due to the fact that we need the trains to take the edge of the response as soon as possible.

4.2 Station

The station automata is very simple. It starts in the initial state, called *Idle*, and it waits for an *enter_station_request* by a train. Once received the request and moved to the *Handle_Request* state, the station needs to send a response to the train. This state is

committed which means that the response will be sent immediately, which is fundamental to ensure that a train does not lose too much time in the exchange of messages. The response can be either positive with an *enter_station_reply_ok* or negative with an *enter_station_reply_ko*. The choice is made according to the result of the function *authorization_granted()*. Actually, it is a really easy function, it just checks if there is at least a free track in the station.



4.3 Train

The train automata is quite complex and it handles most of the operations into the railway model. An image of it can be found in the appendix.

The train will be described in details state by state:

- **Charging**: It is the initial state and it means that the train is inside the station and that it is not ready to start its journey to the next station. In this location it will recharge until it is capable of moving to the next station. The train has to immediately move out of this state as soon as it has sufficient charge to complete its trip assuming the worst scenario. As mentioned before, every train has to complete a finite number of laps into its line. If the train has reached the station from which it has started and has completed all the laps, then it will wait for the minimal recharge and move in the *End_Of_The_Day* state, freeing a track. Otherwise, in all other cases, it will move on and reach the *Overcharging* state.
- **End_Of_The_Day**: It is a state from which the train automata cannot exit and it has been created to stop state explosion in the verification. The train will enter into this

state only if it has completed all the supposed laps in its line. We decided to remove the train from the free track counter once it reached this state considering it like moving the train into a train shed.

- **Overcharging**: It is a state in which the trains are already prepared to start their trip towards the next station. According to our recharge policy we let a train recharging as much as possible even reaching the maximum charge but there may be exceptions. If the station becomes full, then a train in overcharging will not be able to recharge anymore and it will be forced to move out. The train has to mandatorily move out of this state before that the remaining time is not sufficient to complete the trip.
- **Moving**: A train will stay in this state as long as it is traveling and it will decrease its charge accordingly. Moving away from this state, trains will send an *enter_station_request* to the next station.
- **Waiting**: In this state a train waits for the response of the station. This is committed to make sure that a train takes an edge as soon as it receives the response. There can be two replies: *enter_station_reply_ok* which will grant access to the station at the train and *enter_station_reply_ko* which will bring the train into the *Waiting_Request_Delay* state.
- **Waiting_Request_Delay**: A train which was not authorized to enter a station has to wait a delay before submitting another *enter_request_delay*. That delay is spent in this state before returning into the *Waiting* state at the end of that time.

5 Properties verification

There are some properties which are mandatory for a railway line with battery powered trains:

- It never happens that a train reaches the destination after the deadline.
- It never happens that a train runs out of power while traveling or waiting to enter a station.
- The number of trains in a station never exceeds the available tracks.

Though, the verification of these properties does not depend merely on the model but also on the value we give to some parameters. In fact, as an example, we cannot verify the second property if the train doesn't have a battery large enough to sustain the trip to the next station.

Properties for verification in TCTL notation:

1. **∀ □ (train(0).ride_time <= MAX_DELAYS[train(0).current_station][train(0).next_station])**
2. **∀ □ (train(0).charge > 0)**
3. **∀ □ (TRAINS_INSIDE[station(0).id_station] <= MAX_TRACKS[station(0).id_station])**

We expect the third property to be always verified by model construction. A station should never allow in more trains than its capacity.
Some images with the results of the properties verification can be found in the Appendix.

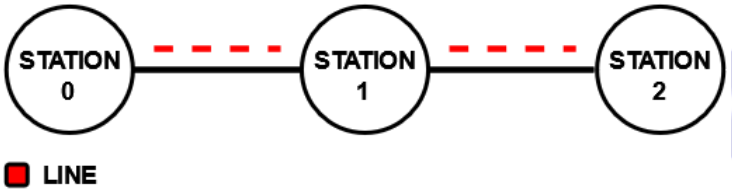
6 Configurations

As we previously mentioned, changing the parameters value can greatly influence the model behavior. All the configurations that we believe are interesting have been included in the model code (file FM_project.xml). It is possible to change configuration just by removing comments from the one we want to use and commenting the one that we don't want anymore. There are images with the different configurations of the parameters we used in the appendix.

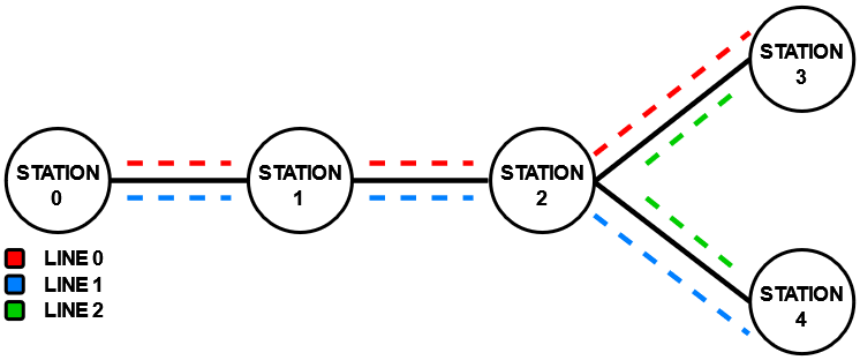
Brief description of them:

- **Simple settings KO**
  Considering the configuration (see Chapter 9.2.2 of the Appendix), we expect that at a certain point Train1 won't be able to reach the target station either by running out of charge or by being late. This, together with a tight constraint on the delay and a station which is congested most of the time (Station1), causes the model to fail properties one and two. As expected, only property three is verified.

- **Simple settings OK**
  Considering a correctly tuned configuration (see Chapter 9.2.1 of the Appendix), the properties are successfully verified despite the bottleneck in Station1 which has just one track in a system with three trains. Compared to the KO settings, trains have been provided with a higher speed, larger maximum charge and proper maximum delays between stations.



- **Complex settings KO and Complex settings OK**
  We also modeled a more general and complex system with 5 stations, 5 trains and 3 different lines (every train follows a single predefined line). As before, the train settings together with the maximum delay between stations influence the correct verification of the properties. Both configurations show some bottlenecks in the stations, in particular Station2 is the main bottleneck since it has only two tracks and all five trains are obliged to pass through this station.



In the simple configurations, we decided to make every train perform eight laps on their lines. In the worst scenarios that should be more than fifteen hours which is a realistic assumption. On the other hand, we had to reduce the number of laps for the complex settings due to the fact that the verification time would have been too high. In any case, the time is around five hours, that is enough to reach the critical points of the model: some stations become congested and trains have to wait outside before obtaining a free track.

7 Conclusion

We developed a complete and working model. It has been built in the most general way offering the possibility to easily extend the number of lines, stations and trains. We created two different configurations: a simple one and a complex one. The simple configuration features three trains and three stations with a single line while the complex one involves five trains and five stations which are part of three different but interconnected lines.

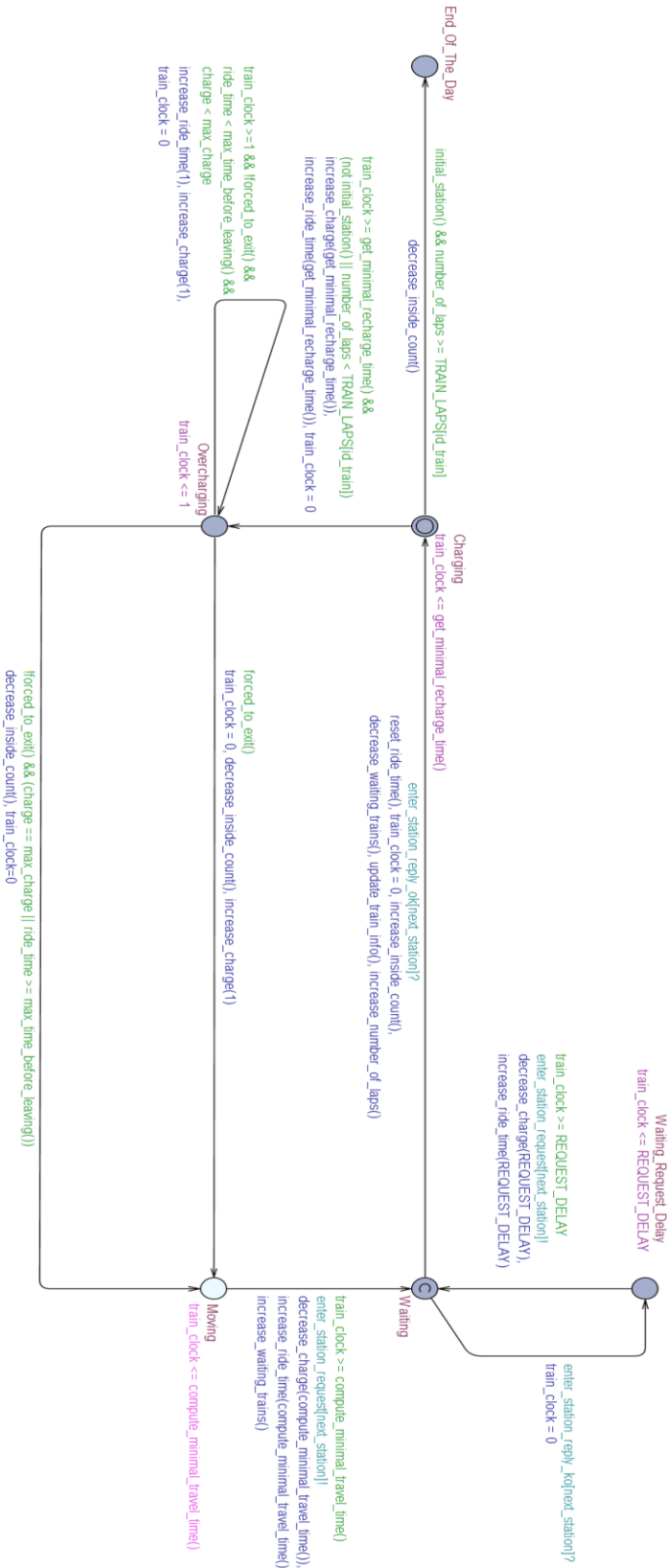
We discovered that the properties we wanted to be asserted were successfully verified with the settings we prepared. Moreover, we tried to tweak the model in order to find some configurations where the properties were failing. For example, an easy way to break a property consists in lowering the time that a train has to reach the next station so that it is harder or even impossible to reach it in time. Clearly, by slowly working on settings, it is possible to iteratively find the configuration that's most efficient for the railway line.

8 References

- UPPAAL
- Material from the "Formal methods for concurrent and real-time systems" course.

9 Appendix

9.1 Train automaton



9.2 Configurations

9.2.1 Simple settings OK:

```
//[***SIMPLE SETTINGS OK***]
//Train settings
const int NUM_TRAINS = 3;
const int RECHARGING_MULTIPLIER = 4;                //recharging rate
const int SPEED[NUM_TRAINS] = {1660, 1162, 1494};   //unit = m/min --> 100km/h, 70km/h, 90km/h
const int MAX_CHARGE[NUM_TRAINS] = {32, 34, 38};    //unit = min
const int INITIAL_CHARGE[NUM_TRAINS] ={25, 27, 30}; //must be less than MAX_CHARGE
const int INITIAL_STATION[NUM_TRAINS] = {0, 2, 0};
const int T_LINES[NUM_TRAINS] = {0, 0, 0};          //line that the train follows
const int NEXT_STATION[NUM_TRAINS] = {1, 1, 1};     //must be inside the line
const int TRAIN_LAPS[NUM_TRAINS] = {8, 8, 8};

//Station settings
const int NUM_STATIONS = 3;
const int SPARE_TIME = 5;
const int MAX_TRACKS[NUM_STATIONS] = {2,1,2};
const int PASSENGERS_TIME = 3;
const int REQUEST_DELAY = 2;
int TRAINS_INSIDE[NUM_STATIONS] = {2,0,1};
int WAITING_TRAINS [NUM_STATIONS] = {0,0,0};

//Railways settins
const int NUM_LINES = 1;
const int NUM_MAX_STATIONS_LINE = 3;
const int LINES[NUM_LINES][NUM_MAX_STATIONS_LINE] = {{0,1,2}};
const int DISTANCES [NUM_STATIONS][NUM_STATIONS] = {{0,18000,-1},
                                                    {18000,0,12000},
                                                    {-1,12000,0}};     //unit = m
const int MAX_DELAYS [NUM_STATIONS][NUM_STATIONS] = {{0,35,-1},
                                                     {35,0,30},
                                                     {-1,30,0}};       //unit = min
//[*** END SIMPLE SETTINGS OK***]
```

9.2.2 Simple settings KO:

```
//[***SIMPLE SETTINGS KO***]
//Train settings
const int NUM_TRAINS = 3;
const int RECHARGING_MULTIPLIER = 4;                //recharging rate
const int SPEED[NUM_TRAINS] = {1660, 830, 1494};    //unit = m/min --> 100km/h, 50km/h, 90km/h
const int MAX_CHARGE[NUM_TRAINS] = {20, 18, 19};    //unit = min
const int INITIAL_CHARGE[NUM_TRAINS] ={15, 12, 12}; //must be less than MAX_CHARGE
const int INITIAL_STATION[NUM_TRAINS] = {0, 2, 0};
const int T_LINES[NUM_TRAINS] = {0, 0, 0};          //line that the train follows
const int NEXT_STATION[NUM_TRAINS] = {1, 1, 1};     //must be inside the line
const int TRAIN_LAPS[NUM_TRAINS] = {8,8,8};

//Station settings
const int NUM_STATIONS = 3;
const int SPARE_TIME = 5;
const int MAX_TRACKS[NUM_STATIONS] = {2,1,2};
const int PASSENGERS_TIME = 4;
const int REQUEST_DELAY = 3;
int TRAINS_INSIDE[NUM_STATIONS] = {2,0,1};
int WAITING_TRAINS [NUM_STATIONS] = {0,0,0};

//Railways settins
const int NUM_LINES = 1;
const int NUM_MAX_STATIONS_LINE = 3;
const int LINES[NUM_LINES][NUM_MAX_STATIONS_LINE] = {{0,1,2}};
const int DISTANCES [NUM_STATIONS][NUM_STATIONS] = {{0,18000,-1},
                                                    {18000,0,12000},
                                                    {-1,12000,0}}; //unit = m
const int MAX_DELAYS [NUM_STATIONS][NUM_STATIONS] = {{0,38,-1},
                                                     {38,0,20},
                                                     {-1,20,0}};   //unit = m
//[*** END SIMPLE SETTINGS KO***]
```

9.2.3 Complex settings OK:

```
//[***COMPLEX SETTINGS OK***]
//Train settings
const int NUM_TRAINS = 5;
const int RECHARGING_MULTIPLIER = 4;                          //recharging rate
const int SPEED[NUM_TRAINS] = {1660, 1162, 1494, 1992, 1328}; //unit = m/min --> 100km/h, 70km/h, 90km/h, 120km/h, 80km/h
const int MAX_CHARGE[NUM_TRAINS] = {32, 38, 34, 30, 36};      //unit=min charge
const int INITIAL_CHARGE[NUM_TRAINS] ={26, 32, 28, 26, 30};   //must be less than MAX_CHARGE
const int INITIAL_STATION[NUM_TRAINS] = {0,3,0,4,4};
const int T_LINES[NUM_TRAINS] = {1,2,0,2,1};                  //line that the train follows
const int NEXT_STATION[NUM_TRAINS] = {1,2,1,2,2};             //must be inside the line
const int TRAIN_LAPS[NUM_TRAINS] = {2, 2, 2, 2, 2};

//Station settings
const int NUM_STATIONS = 5;
const int SPARE_TIME = 4;
const int MAX_TRACKS[NUM_STATIONS] = {3,2,2,1,2};
const int PASSENGERS_TIME = 3;
const int REQUEST_DELAY = 2;
int TRAINS_INSIDE[NUM_STATIONS] = {2,0,0,1,2};
int WAITING_TRAINS [NUM_STATIONS] = {0,0,0,0,0};

//Railways settins
const int NUM_LINES = 3;
const int NUM_MAX_STATIONS_LINE = 4;
const int LINES [NUM_LINES][NUM_MAX_STATIONS_LINE] = {{0,1,2,3},{0,1,2,4},{3,2,4,-1}};
const int DISTANCES [NUM_STATIONS][NUM_STATIONS] = {{0,18000,-1,-1,-1},
                                                    {18000,0,15000,-1,-1},
                                                    {-1,15000,0,12000,20000},
                                                    {-1,-1,12000,0,-1},
                                                    {-1,-1,20000,-1,0}};
const int MAX_DELAYS [NUM_STATIONS][NUM_STATIONS] = {{0,52,-1,-1,-1},
                                                     {52,0,42,-1,-1},
                                                     {-1,42,0,38,60},
                                                     {-1,-1,38,0,-1},
                                                     {-1,-1,60,-1,0}};
//[*** END COMPLEX SETTINGS***]
```

9.2.4 Complex settings KO:

```
//[***COMPLEX SETTINGS KO***]
//Train settings
const int NUM_TRAINS = 5;
const int RECHARGING_MULTIPLIER = 4;
const int SPEED[NUM_TRAINS] = {1660, 996, 1494, 1162, 996};     //unit=m/min --> 100km/h 60km/h 90km/h 70km/h 60km/h
const int MAX_CHARGE[NUM_TRAINS] = {32, 38, 34, 30, 34};        //unit=min charge
const int INITIAL_CHARGE[NUM_TRAINS] ={26,28,28,26,30};         //must be less than MAX_CHARGE
const int INITIAL_STATION[NUM_TRAINS] = {0,3,0,4,4};
const int T_LINES[NUM_TRAINS] = {1,2,0,2,1};                    //line that the train follows
const int NEXT_STATION[NUM_TRAINS] = {1,2,1,2,2};               //must be inside the line
const int TRAIN_LAPS[NUM_TRAINS] = {2, 1, 2, 2, 1};

//Station settings
const int NUM_STATIONS = 5;
const int SPARE_TIME = 4;
const int MAX_TRACKS[NUM_STATIONS] = {3,2,1,1,2};
const int PASSENGERS_TIME = 3;
const int REQUEST_DELAY = 2;
int TRAINS_INSIDE[NUM_STATIONS] = {2,0,0,1,2};
int WAITING_TRAINS [NUM_STATIONS] = {0,0,0,0,0};

//Railways settins
const int NUM_LINES = 3;
const int NUM_MAX_STATIONS_LINE = 4;
const int LINES [NUM_LINES][NUM_MAX_STATIONS_LINE] = {{0,1,2,3},{0,1,2,4},{3,2,4,-1}};
const int DISTANCES [NUM_STATIONS][NUM_STATIONS] = {{0,18000,-1,-1,-1},
                                                    {18000,0,15000,-1,-1},
                                                    {-1,15000,0,15000,20000},
                                                    {-1,-1,15000,0,-1},
                                                    {-1,-1,20000,-1,0}};
const int MAX_DELAYS [NUM_STATIONS][NUM_STATIONS] = {{0,44,-1,-1,-1},
                                                     {44,0,38,-1,-1},
                                                     {-1,38,0,38,40},
                                                     {-1,-1,38,0,-1},
                                                     {-1,-1,40,-1,0}};
//[*** END COMPLEX SETTINGS***]
```

9.3 Update train information algorithm

```
void update_train_info(){
    int next_index = 0;
    for (current_index : int[0, NUM_MAX_STATIONS_LINE-1])
    {
        if(LINES[line][current_index] == current_station){
            current_station = next_station;
            next_index = current_index;
            if(next_index-1 < 0 || LINES[line][next_index-1] == -1){
                next_index = next_index + 1;
            }
            else if(next_index+1 > NUM_MAX_STATIONS_LINE-1 || LINES[line][next_index+1] == -1){
                next_index = next_index - 1;
            }
            else{
                if(LINES[line][next_index+1] == next_station){
                    next_index = next_index+1;
                }
                else if(LINES[line][next_index-1] == next_station){
                    next_index = next_index-1;
                }
            }
            if(current_index < next_index){
                if(next_index+1 > NUM_MAX_STATIONS_LINE-1 || LINES[line][next_index+1]== -1)
                {
                    next_index = next_index - 1;
                }
                else
                {
                    next_index = next_index + 1;
                }
            }
            else if(next_index -1 < 0 || LINES[line][next_index-1]== -1)
            {
                next_index +=1;
            }
            else
            {
                next_index -=1;
            }
            next_station = LINES[line][next_index];
            return;
        }
    }
    return;
}
```

9.4 Properties verified

9.4.1 Simple model OK:

```
A[] forall(i:station_t) TRAINS_INSIDE[Station(i).id_station] <= MAX_TRACKS[Station(i).id_station]
A[] forall(i:train_t) Train(i).ride_time <= MAX_DELAYS[Train(i).current_station][Train(i).next_station]
A[] forall(i:train_t) Train(i).charge > 0
```

9.4.2 Complex model OK:

```
A[] forall(i:station_t) TRAINS_INSIDE[Station(i).id_station] <= MAX_TRACKS[Station(i).id_station]
A[] forall(i:train_t) Train(i).ride_time <= MAX_DELAYS[Train(i).current_station][Train(i).next_station]
A[] forall(i:train_t) Train(i).charge > 0
```