POLITECNICO
MILANO 1863

Computer Science and Engineering

# Model Checking of Battery-Powered Railway Lines

## Formal Method for Concurrent and Real Time Systems Project

Academic year 2021 - 2022

24 June 2022
Version 1.0

*Authors*:
Lorenzo IOVINE
Nicola LANDINI
Francesco LEONE

*Professors*:
Pierluigi SAN PIETRO
Livia LESTINGI

| | |
|---|---|
| **Deliverable:** | Formal Method for Concurrent and Real Time Systems project |
| **Title:** | Model Checking of Battery-Powered Railway Lines |
| **Authors:** | Lorenzo Iovine, Nicola Landini, Francesco Leone |
| **Version:** | 1.0 |
| **Date:** | 24-June-2022 |
| **Copyright:** | Copyright © 2022, Lorenzo Iovine, Nicola Landini, Francesco Leone – All rights reserved |

# Contents

# List of Figures

# 1   Design

## 1.1   Purpose

The fight to reduce greenhouse gas emissions is bringing together researchers and manufacturers from all over the world. In particular, rechargeable batteries as a source of power in place of fossil fuels are already widespread in cars and making their way into the rail transport sector. Battery-powered trains are already operative in several countries like Japan, Austria, and Britain. Italy is also planning on producing and deploying fully-electric trains starting mid-2022, thanks to a deal with Hitachi Rail.

Like any electric vehicle, trains can cover a limited distance running only on battery power before needing to recharge. In this project, we will model a railway line in which electric trains can either recharge in a station. Nevertheless, trains must still reach the following station on time; in case of excessive delay, the company is obliged to issue monetary compensation to the passengers.

Precisely, given a set of simplifying assumptions, we will model the main actors of the system as a network of **Timed Automata (TA)** whose behavior depends on specific key parameters.

## 1.2   High Level Model Description

We created two different configurations for the railway model. Both of them include 4 trains and 3 stations.
The first one represents the main configuration of the system and verifies all the properties. The railway model is set as follows:

distance: 40km
max delay: 50min

distance: 48km
max delay: 60min

The second configuration doesn't verify the mandatory properties of the project and it is set as follows:

distance: 40km
max delay: 25min

distance: 48km
max delay: 30min

## 1.3   Initializations

The stations have the following initial configurations:

- **Station 0:** 2 tracks, 1 available

- **Station 1:** 3 tracks, 2 available

- **Station 2:** 2 tracks, 0 available

The trains that we designed have constant speed set to 120km/h. They are initialized as follows:

- **Train 0 - charge 100:** starts from station 0 with station 2 as destination

- **Train 1 - charge 100:** starts from station 1 with station 2 as destination

- **Train 2 - charge 100:** starts from station 2 with station 0 as destination

- **Train 3 - charge 100:** starts from station 2 with station 0 as destination

## 1.4   Design assumptions

In order to efficiently describe the model, we decided to set a railway in which every station has less tracks than the total number of train train, in order to obtain much more often situations in which a train has to wait to have the permission to enter his next station.
Furthermore we decided not to design the railway with a dedicated template. That's because the railway's most important features are implicitly designed and verified, without creating additional variables.
In the end, another assumptions that we made to simplify the description is that we assumed the clock unit as a minute.

# 2   UPPAAL Model

## 2.1   Global Declarations

### 2.1.1   System Parameters

| Parameter | Description |
|-----------|-------------|
| nStations | Number of stations in the system |
| nTrains | Number of trains in the system |
| chargingSpeed | A multiplier for the battery charging speed |
| journeyDischargeMultiplier | A multiplier for the battery discharge during the journey |
| waitingDischargeMultiplier | A multiplier for the battery discharge during the waiting |
| MIN_TIME_PER_STATION | Lower bound to allow passengers to get on and off the train |
| trainSpeed | Trains' speed |
| stationNumOfTracks | Array with the number of tracks of each station |
| railLine_distance | Matrix containing the distance between stations |
| railLine_delay | Matrix containing the maximum delay between stations |

### 2.1.2   System Variables

| Variable | Description |
|----------|-------------|
| trainStation | Array with the current station of each train |
| trainDestination | Array with the destination of each train |
| nextStop | Array with the next station of each train |
| stationFreeTracks | Array with the current number of free tracks of each station |
| chargeOfTrain | Array with the current charge of each train battery |

### 2.1.3   System Channels

| Channel | Description |
|---------|-------------|
| exitTrain | Channels through which a train inform its current station that it is leaving |
| full | Channels through which a station communicate to an incoming train that it has no available tracks |
| allowedIn | Channels through which a station grant the access to a train that is waiting |
| arrived | Channels through which a train inform its next station that it is arrived |

## 2.2   Templates

### 2.2.1   Station
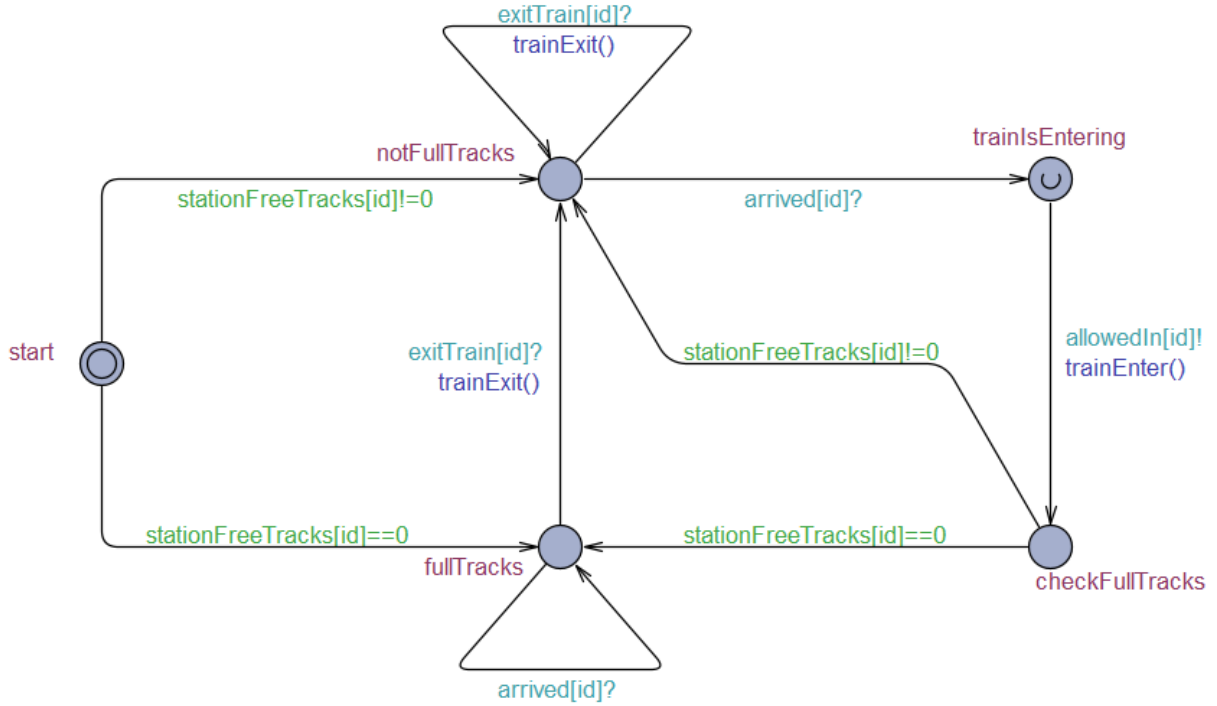


- **start:** this is the initial state of the Station Template. In case the number of available tracks is equal to zero the next state is notFullTracks, otherwise the next state is *fullTracks*.

- **notFullTracks:** when this state receives a message on the *exitTrain* channel, it updates his *stationFreeTracks* variable and the current state doesn't change. Upon receiving a message on the *arrived* channel the current state moves to *checkFreeTracks* state.

- **fullTracks:** when this state receives a message on the *arrived* channel the current state doesn't change. Instead, if this state receives a message on the *exitTrain* channel, it updates his *stationFreeTracks* variable and the current state moves to *notFullTracks* state.

- **trainIsEntering:** this is an urgent state created to describe when a train is joining the station.

- **checkFullTracks:** this state is reached immediately after the update of the *stationFreeTracks* variable and checks if the station has available tracks. In this case the current state becomes *notFullTracks*, otherwise it becomes *fullTracks*.

### 2.2.2 Train
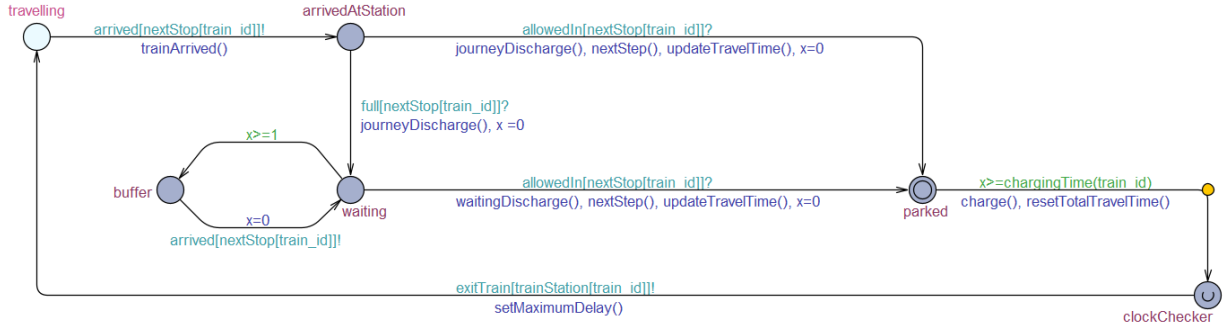


- **parked:** this is the initial state of the Train Template. It represents the situation in which the train is stopped in a station. The current state changes only if the clock is greater or equal to the needed charging time. Before the current state changes in *clockChecker* we compute the *charge* method that increases the train battery linearly with the time spent in the station.

- **clockChecker:** this is an urgent state created to send a message on *exitTrain* channel and to set the maximum delay to reach the next station.

- **travelling:** this state represents the situation in which the train left the previous station and heads to the next one. Before reaching the next station, we send a message on *arrived* channel to inform the station of the arrival and update train's current station.

- **arrivedAtStation:** this state checks if the train has to wait before entering or if it is allowed to join the station. In the first case the current state becomes *waiting* after receiving a message on *full* channel, otherwise the system comes back to the initial state after receiving a message on *allowedIn* channel. Before reaching one of the next state we compute the *journeyDischarge* method that decreases the train battery linearly with the travel time.

- **waiting:** this is the state in which the train waits until the station, throw *allowedIn* channel, allows him to join. Before entering the station we compute the *waitingDischarge* method that decreases the train battery linearly with the waiting time.

- **waiting-buffer:** this is a snippet of Train Template that describes the loop in which the train is while waiting to be allowed to join the next station. In this loop we send a message through *arrived* channel to the station every clock time unit.
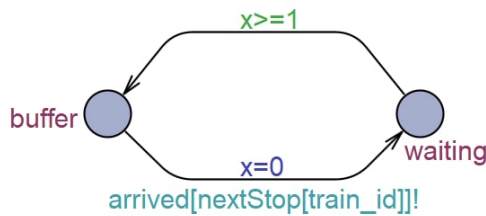
| Train local variables | Description |
|---|---|
| x | This is the clock used to temporalize the system |
| totalTravelTime | An int that represents the overall time to reach the next station |
| parkedTime | An int that represents the time spent in a station |
| travelTime | An int that represents the time spent in the journey |
| waitingTime | An int that represents the time wasted until the station authorizes the access |
| actualMaximumDelay | An int used to save the maximum delay allowed to reach the next station |

# 3 Analysis and Results

## 3.1 Property Verification

### 3.1.1 Mandatory Properties

In this section we will describe the analyzed properties. The first two properties analyzed are the compulsory ones:

- $\forall\square(chargeOfTrain[Train\_id] > 0)$

- $\forall\square(train.totalTravelTime <= train.actualMaximumDelay)$

We have created two configuration of the system: in the first one the two properties are satisfied, in the second one not. The difference between the two configuration is the max delay allowed.

**Configuration1** In this configuration the properties are satisfied

- Max Delay Station 1-2 : 50

- Max Delay Station 2-3 : 60

**Configuration2** In this configuration the properties are not satisfied

- Max Delay Station 1-2 : 25

- Max Delay Station 2-3 : 30

The two properties could be satisfied even in this configuration if we increase the *chargingMultiplier*

### 3.1.2 Additional Properties

We have written two additional properties, the first one to check that the number of free Tracks in a station is always greater or equal to 0, the second one to check that every train eventually reaches its destination.

- $\forall\square(stationFreeTracks[station\_id] >= 0)$

- $\forall\lozenge(trainStation[Train\_id] = trainDestination[Train\_id])$

Both properties are satisfied in both configuration of the system.

## 4   Conclusion