POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

MODEL CHECKING OF BATTERY-POWERED RAILWAY LINES

Project for Formal Methods for Concurrent and Real-Time Systems

Authors: **Alice Cariboni (10617174)**
Emilio Corigliano (10627041)

Students ID: 10617174, 10627041
Professor: Prof. Pierluigi San Pietro
Assistant: Ph.D. Livia Lestingi
Academic Year: 2021-22

Abstract

The project consists in modeling and studying a railway system in which circulates battery-powered trains that can recharge only while inside a station. The analysis of the system has been done starting from some parameters and watching how the safety of the system reacts perturbing them.

Contents

Introduction

The context of the project is the analysis of a battery-powered railway network where trains (that can have different characteristics) travel thanks to the energy kept in some batteries. The principal characteristics of the system (with the nomenclature used in the next sections and in the code) are described below:

- Trains:
  - **SOCmax**: the capacity of the batteries of the train; the **SOC** (State Of Charge) of each train can't exceed this parameter in any moment.
  - **V**: the speed of the train (constant when the train moves);
  - **Cdis**: The coefficient of discharge per minute of travelling to a station or waiting outside a station;
  - **Crec**: The coefficient of recharge per minute while in station;
- Stations:
  - **capacity**: The maximum number of trains that the station can contain; **N** is the counter of the trains that are in the station at the moment, this can't exceed in any moment the capacity of the station;
- System:
  - **R**: Period of time after which a train can resend the request to enter in the following station;
  - **minTimeInStation**: minimum period of time that a train must stay in a station in order to let the passengers get on and off the train;
  - **chargingStrategy**: the policy that will be used to recharge the trains;

Each train starts from a station and, from there, it will stop to each following station. Once at the terminus, the train will restart travelling in the opposite direction. The stations, instead, accept as many trains as they can, not exceeding their capacity. If they can't accept other trains, they will reject the request to enter.

1 | Modeling

Our system is composed of three main subjects: trains, stations and railway network. The first two have been modelled as Timed Automaton while the railway network is modelled as a combination of the stations and an array of structures that contain information about the "Links" of two stations. All the times in the system are considered in min, all the distances in Km, all the speeds in $\frac{Km}{h}$ and all the rates of charging/discharging are in $\frac{min}{min}$.

1.1. Trains

The trains are modeled as Timed Automatons because they have a dynamic behaviour with respect to their internal state. Also, they have to communicate with the stations; so, being a completely different entity, they have to communicate via channels. Each train has it's own internal state and state machine.

1.1.1. Initialization

They start from the *Init* urgent state, so that at the beginning of the simulation the trains initialize their internal state.

From there they go directly to the state *travelling*. This has been chosen as the real "entrypoint" of the state machine because of consistency: doing this, we can initialize all the stations to have an initial number of trains in the station equal to 0 because all the trains begun travelling at time 0.

1.1.2. Interaction with stations

When the time to reach the next station is passed, we send an "entry" request to that station and wait for its response:

- rejected: make the train go in a cycle that will request to enter every **R** minutes till it is accepted;

- accepted: will make the train stop in the next station and start the recharge;

The state *waitToEnter*, where the train waits the response of the station, is urgent so that we will be precise on the period for resubmitting the request.

Once the train enters in the station and has recharged for at least the minimum time to pass in the station, it will start travelling to the next station communicating to the last station that the train left.

1.1.3. Recharging policies

The time spent in the charging phase depends on the policy of the railway. We thought of two different policies:

1. The train recharges just the time necessary to reach the other station with the maximum delay possible: $SOC = C_{dis} \cdot (\frac{distance}{V} \cdot 60 + maxDealy)$; So we get the minutes that takes the train to reach the next station summed to the maximum delay accepted on that link and we multiply this time by the rate of discharge to gain the minimum SOC necessary to start travelling. In the real model this quantity is always a bit more than the minimum in order to compensate for possible approximations down.
2. The train recharges till it reaches the maximum charge possible ($SOC = SOC_{max}$).

1.2. Stations

The stations are modeled as Timed Automatons because they have to interact with the Trains thorough channels. Stations will usually stay in the "Idle" state, but when the station receives a request to enter, the station will immediately respond (thanks to an urgent state):

- accepts the request if the number of trains is less than the capacity; So, a train will arrive and the counter of trains in the station increments;
- rejects the request if the station is full.

When receives the signal that a train left, the station will decrement the number of trains that are currently recharging in the station.

1.3. System

The communication among processes takes place thanks to arrays of channels of length equal to the number of stations ($N\_STATIONS$). Each station is associated to an index (in the system declaration) so that they will respond only to requests made for that station.

A link is a structure that contains the information (distance and maximum delay accepted) of a bidirectional piece of railway that goes from station s_i to station s_{i+1}. This array has length $N\_STATIONS - 1$ and is indexed by the index of the station with lower index in the link (so to get the link of two stations s_i and s_{i+1} we will access $railway_i$). The correlation among stations and links is made "naturally" by the indexes of link and stations.

Trains will save the index of the *lastStation* they visited, so the link they are travelling on is $railway_{lastStation}$ if they are going forward, $railway_{lastStation-1}$ if they are going backwards.

2 | System analysis

We divided the analysis of the system in different sections:

- Correctenss analisys: simply deals with some crosschecks that the system respects the basic specifications and doesn't exceed bounds;
- Safety analisys: We studied how the system reacts with the variation of the instance parameters in order to have an idea of the safety boundaries of the system;
- Fairness analisys: Checking that all the trains will eventually enter in a station/depart from a station; so, we are checking if the trains will always make some progress eventually.

Every property is verified for each train, station or for the whole system, depending on the property itself. For checks on trains and stations we just grouped all the asserts and combined them with ands, so we won't distinguish among different trains but we will just see if the whole system satisfies or not the properties.

2.1. Correctenss analisys

We begin with some properties that have to be verified in order to validate the correctness of the system. Some check the model correctness while others check if the set of parameter chosen are valid:

- Check that the trains won't ever exceed SOC_{max}:

$$\forall\Box(\forall t \in trains, t.SOC \leq t.SOC_{max}) \tag{2.1}$$

- Check that every time a train reaches one of the two final stations, the train will continue in the right direction:

$$\begin{aligned}\forall\Box(\forall t \in trains, (t.lastStation = 0 \implies t.goingForward) \wedge \\ (t.lastStation = N\_STATIONS - 1 \implies \neg t.goingForward))\end{aligned} \tag{2.2}$$

- Check that every station will always have less or equal trains than the station capacity:

$$\forall\Box(\forall s \in stations, s.N \leq s.capacity) \tag{2.3}$$

- Check that every station won't ever have a negative number of trains:

$$\forall\Box(\forall s \in stations, s.N \geq 0) \tag{2.4}$$

- check that every train have enough initial power to reach the next station; this depends only on the initial instantiation of the trains (we use the same notation of UPPAAL where $t.state$ is true only when the active state is $state$):

$$\forall t \in trains, t.Init \implies (t.SOC0 \geq t.Cdis \cdot t.estimatedTimeTravelling()) \tag{2.5}$$

- check that every train have enough potential power to travel for the maximum distance present in the railway network:

$$\forall t \in trains, t.SOC_{max} \geq (\frac{maxDistance}{t.V} \cdot t.Cdis \cdot 60) \tag{2.6}$$

2.2. Safety analisys

We now want to check for some properties about the system in order to see if it satisfies our energy and time constraints:

- Check that the trains will always have enough charge to reach the next station (i.e. the trains will always have a non-negative SOC value):

$$\forall\Box(\forall t \in trains, t.SOC \geq 0) \tag{2.7}$$

- Check that the trains will always reach the stations in time, not exceeding the maximum time available:

$$\forall\Box(\forall t \in trains, t.waitToEnter \implies (t.timeTravelling \leq t.maximumTimeAvailable())) \tag{2.8}$$

- Check the deadlock freedom of the system

Two different cases in which the first two assertions will pass and in another case will fail are when we change the charging policy when we instantiate 5 different trains with 60/100 of SOC0/SOCmax, velocity of 120, rate of discharge of 3 and rate of recharge of 6, three trains departing from station 0 and the other two from station 4, all going forward.

2.3. Liveness analisys

We may want to check if eventually there will be progress in the system:

- We verify that from the state *charging* we will eventually reach the state *travelling* and also that from state *travelling* we will reach the state *charging*. This should be enough to demonstrate that the trains won't ever livelock (they will always make some progress):

$$\forall t \in trains, \forall \Box (t.charging \implies \forall \diamond t.travelling \wedge t.travelling \implies \forall \diamond t.charging) \tag{2.9}$$

2.4. Parameter perturbation

After defining all the asserts we are interested in studying the system changing some parameters of interest. We are in the context of the Verona-Brescia-Milano line of Trenord, that inspired (with lots of variations anyway) the parameters about the distance of the stations and the quantity of tracks.

2.4.1. System configuration

For simplicity we considered only some stations in order to have a big system but not too big that will take forever to verify. We thought that, in order to study how parameters influence the satisfiability of our time and energy constraints, is better to maintain constant the topology, stations and parameters of the railway network.

Another restriction we used during the study of the system is that the number of trains is pretty low: we will consider only 4 trains. Also, their characteristics are the same (except for the starting station). This permits UPPAAL to optimize the verification of the properties so that it won't take forever to finish a simple verification.

The parameters that we chose not to change are:

- **Stations**: The stations and the railway are left unchanged;

- **Trains**: We will consider only 4 trains with all the same characteristics, that are initial charge of 60 and maximum charge of 100, starting stations (two starting from the first station and two starting from the fifth);
- **resubmission time**: considered only a resubmission time of 1 because it make us see better the changings in the response of the system
- **minimum time in station**: considered only 1 minute of minimum time in station;

2.4.2. Parameters of interest

The parameters we will take into account during the analysis of the system are:

- **chargingStrategy**: we will consider both strategies to see what strategy works the best in our context;
- **Crec**: the rate of recharge of the batteries in the train, we will vary it among [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
- **Cdis**: the rate of discharge of the batteries in the train, we will vary it among [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
- **V**: the velocity of the train, we will vary it among [120, 150, 180, 210, 240];

2.4.3. Verification process

In order to automatize the verification of a large number of models we developed a python script (available in our github repository). This permits us to automate the changing of the model and the verification part (did from the verifyta software, that verifies assertions for models written in UPPAAL).

We also won't check all the properties we wrote in the previous sections because of time constraints: some assertions, once verified for some models, are no longer interesting (such as correctness of the calculation of SOC, the increment of the number of trains in the stations and the liveness properties). Others, such as the checks on the initial state of the model, are useful to filter models with an invalid initial state: these will be verified in the script to speed up the process. So, due to time issues, the only properties we check with verifyta are the Safety ones ((2.7) and (2.8)).

2.5. Results

After our analysis we can make some conclusions on the variations of the parameters we considered.

2.5.1. Recharging policies

The first thing we want to discuss is about the recharging policies we chose. The idea behind the first one is that once the system fails to make the trains arrive in the next station with the maximum delay, we already have a fail in the requirements of the system. In this case the two safety asserts ($SOC > 0$ and $timeTravelling < timeWithMaxDelay$) will fail at the same time.

In the first strategy we can see that sometimes only one fails. When only the "has enough SOC" assert passes, the cause could be due to the slight approximation in excess during the calculation of SOC in the first strategy.

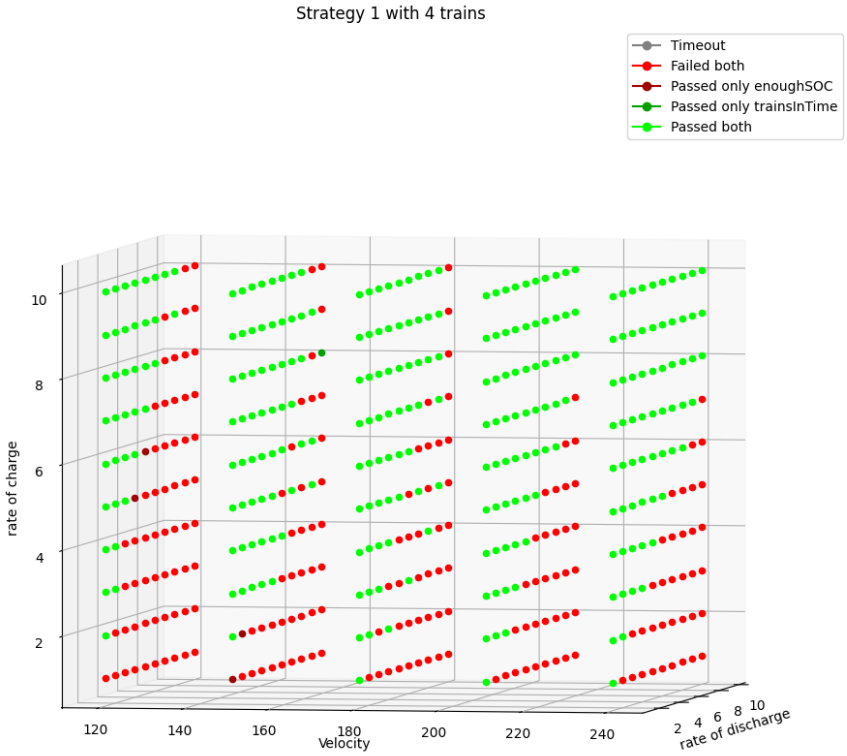
Without the tests, the first policy could seem less safe for practical uses than the second one (because if we make a bit more delay than the maximum, not only we have to pay the customers but we will have trains along the railway with no power, not able to reach the next station to recharge).

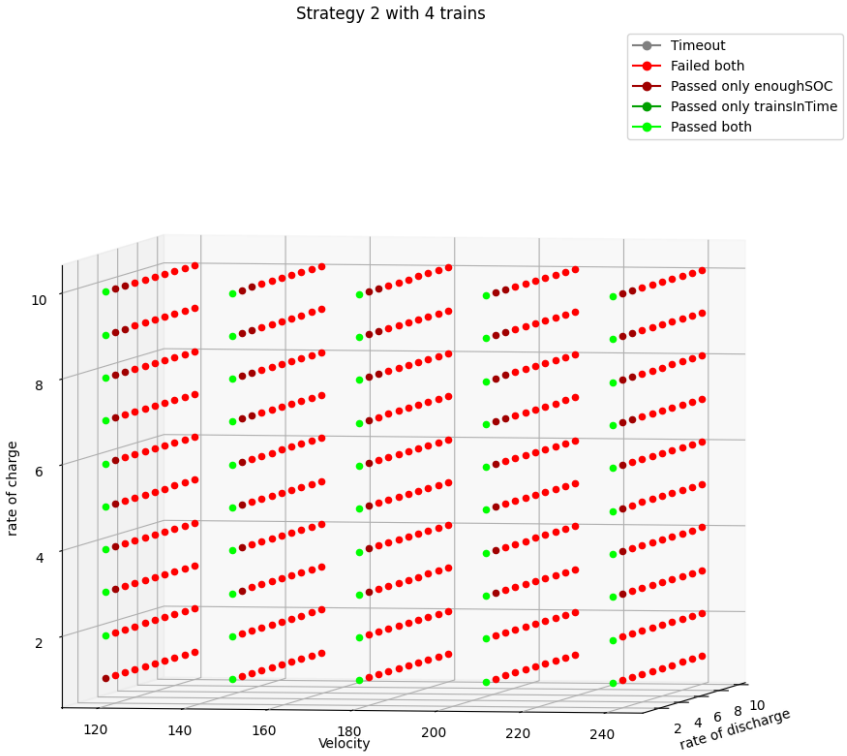
Thanks to the tests, we can see that this is not the case: with the second strategy the time to charge the trains increases, so the delays to enter in the stations increase too and the problems persist. So, with the second strategy the safety asserts will fail much more often and in less restrictive situations than the other policy.

2.5.2. Other parameters

Regarding the other parameters analyzed, they mostly follow the intuitive direction:

- Increasing the rate of charge will make the system perform in a much better way, lowering the charge time and so also limiting the waiting outside the stations;
- Having a high power efficiency (so, a low rate of discharge) is important so that we won't need to charge the battery a lot. This also reduces the necessary capacity of the battery;
- Going at higher speeds make the distance between stations less influential than the waiting outside of the station: optimizing this is important if there are long distances but less important if we have a busy system with lots of trains travelling.


Strategy 1 with 4 trains
Timeout
Failed both
Passed only enoughSOC
Passed only trainsInTime
Passed both
rate of charge
2
4
6
8
10
120
140
160
180
200
220
240
Velocity
2 4 6 8 10
rate of discharge


Strategy 2 with 4 trains
Timeout
Failed both
Passed only enoughSOC
Passed only trainsInTime
Passed both
rate of charge
2
4
6
8
10
120
140
160
180
200
220
240
Velocity
2 4 6 8 10
rate of discharge

3 | Conclusions and future developments

With the model and the tools we developed there is a great margin of study on these type of systems: we could study for example how to size appropriately the maximum delay for each link or how the system performs solving the bottlenecks or we can even test the impact of other parameters we left constant (for example the minimum time to spend in a station or the period of request of entry in the stations). Also, improving the queries and developing a distribution of the load in the verification process (given that the verification of the properties is embarrassingly parallel) we could speed up the verification part by a lot.