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 railway systems in which can circulate battery-powered trains; those can recharge only while inside a station. The analysis of the system has been done starting from some parameters and watching how the system reacts perturbing them. For the topology of the railway network we took inspiration from the Verona-Brescia-Milano line and we decided that it was more interesting to maintain the topology constant among all the tests we did.

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/waiting;
  - **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;
- 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 restarts 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. Each train has it's own internal state and state machine.

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.

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

- rejected: make the train go in a cycle that will request to enter every **R** minutes;
- accepted: will make the train stop in the next station and start the recharge;

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

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

1. The train recharges just the time necessary to reach the other station with the maximum delay possible;
2. The train recharges till it reaches the maximum charge possible

maybe bring this considerations at verification time or conclusions

A consideration is needed on the thought of these recharging policies. 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 "bad" system. In this case the two safety asserts ($SOC > 0$ and $timeTravelling < timeWithMaxDelay$) will fail at the same time.

while quoting the safety properties make a reference to them

Even if the first policy seems less safe than the second for practical uses (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). On the other hand, the second policy seems safer but we can see that the delays to enter in the stations are longer and the problems persist: the safety assets will fail much more often and in less restrictive situations than the other policy, this is so because we accumulate delay and we spend more time recharging the train, so the incoming trains will wait longer.

make a better explanation of the second policy

Once the train has recharged, it will start travelling to the next station communicating to the last station that the train left.

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:

- accept the request if the number of trains is less than the capacity;
- reject the request if the station is full.

When receives the signal that a train left, the station will update 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.

The correlation among stations and links is made "naturally" by the *railway* array of links. 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 $railway_i$.

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.

Add specification that we used the line Verona-Brescia-Milano for inspiration in the parameters

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. In parcticular we checked for each implementation that the **SOC** won't never exceed **SOCmax**;

  Describe all the assertions done

- 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;

  Describe all the assertions done

- Fairness analisys: Checking that all the trains will eventually enter in a station/depart from a station

  Describe all the assertions done

Every property is verified for each train, station or for the whole system, depending on the property itself. For checks on trains 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}:

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

- Check that every time a train reaches one of the two final stations, the train will

continue in the right direction:

$$\forall\Box(\forall t \in trains, (t.lastStation = 0 \implies t.goingForward) \wedge$$
$$(t.lastStation = N\_STATIONS - 1 \implies \neg t.goingForward))$$

- Check that every station will always have less or equal trains than the station capacity:
$$\forall\Box(\forall s \in stations, s.N \leq s.capacity)$$

- Check that every station won't ever have a negative number of trains:
$$\forall\Box(\forall s \in stations, s.N \geq 0)$$

- 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 $state$):
$$\forall t \in trains, t.Init \implies (t.SOC0 \geq t.Cdis \cdot t.estimatedTimeTravelling())$$

- check that every train have enough potential power to travel for the maximum distance present in the railway network:
$$\forall t \in trains, t.SOC_{max} \geq (\frac{maxDistance}{t.V} \cdot t.Cdis \cdot 60)$$

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):
$$\forall\Box(\forall t \in trains, t.SOC \geq 0)$$

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

maximum time available:

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

- Check the deadlock freedom of the system

2.3. Liveness analisys

We want to know if eventually there will always be progress in the system, so there is no livelock:

complete the liveness asserts

- 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):

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

2.4. Parameter perturbation

After defining all the asserts we are interested in, we are 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.

For simplicity we are considering 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 and parameters of the railway network with all his stations.

Also, another restriction we used during the study of the system is that the number of trains and their characteristics are all the same. This permits UPPAAL optimize the verification of the properties so that it won't make forever.

So, the parameters we will take into account 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, 240];

We also won't check all the checks 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 initial state that won't make them complete a run in the railway even in the optimum case (no other train is present in the railway).

Remember to talk about verification time issues using different perturbations: incrementing trains, not nice perturbations of velocity or rates of discharge/recharge

3 | Conclusions and future developments

A final chapter containing the main conclusions of your research/study and possible future developments of your work have to be inserted in this chapter.

Bibliography