System Validation 2025
Homework Part 1 - Model Checking

Marieke Huisman Robert Mensing

Available: Friday, September 05, 2025
Deadline: **23:59 CET, Friday, September 26, 2025**

The assignments should be made in pairs.
This assignment consists of

- Writing and analyzing temporal logic specifications.
- Understanding and adapting NuSMV models.
- Understanding and reproducing CTL and LTL model checking algorithms.

All solutions should be uploaded to Canvas. And handed in as a single ZIP file with:

- A report describing your solution to part 1 in **PDF** format as `report-part-1.pdf`. The report should fully describe your solutions. *Please document your reasoning for all exercises, including exercises that should be solved in code.*
- A report describing your solution to part 2 in **PDF** format as `report-part-2.pdf`.
- The input and output files as required in each of the exercises.

Note that if you use a generative AI like ChatGPT for your answers, you should clearly mark and explain your use of it. We recommend that you complete all assignments without the aid of generative AI. Remember also that you will be unable to use it in the final exam.
In all files, and on the first page of both parts of the report, write your names, your student numbers and your group number.

If you have any technical questions regarding the assignment, contact Robert Mensing at r.a.mensing@utwente.nl. You can expect answers on weekdays between 8:30am and 4:30pm. Any questions received outside of these hours will not be answered until the next working day, so please plan accordingly.

Also remember that you are expected to work on the lab exercises to build your understanding of the content of the assignment. If you have content-related questions about the assignment, you should complete the lab exercises and ask questions about them in the exercise sessions.

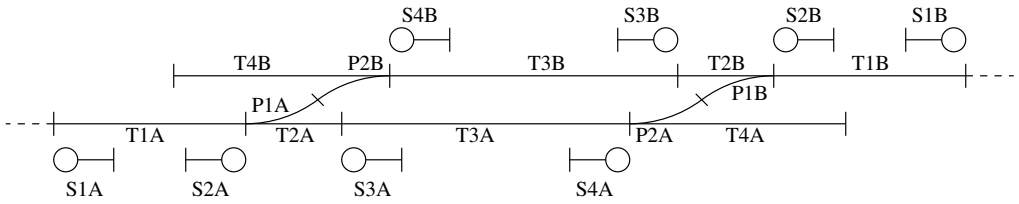

Figure 1: The layout of the passing area. The different segments of the rail road are prefixed with a "T". The switch components are prefixed with a "P" and the signaling devices are prefixed with an "S". Concretely, this picture shows two rail tracks (both consisting of several components), with switching components between them.

Part 1 - NuSMV and Temporal Logics (70 pts)

In this part, we are going to verify the system that controls the signals and switches of a passing area in a single track rail road. Such systems are called *interlockings*.

Modelling Approach

Many systems, including train interlockings, may not work correctly given random inputs (e.g. from events occuring in the environment) or depend on controlled hardware behaving in a certain way. For example, trains do not (dis)appear at random and switches take time to move between their two positions. Therefore, by modelling a train interlocking system, we also need to model the environment in which the interlocking is to be validated. We also do this in NuSMV, yielding three parts: *(i)* a model for the interlocking system itself, describing its behaviour; *(ii)* the environment, in which the interlocking should operate; and *(iii)* the specifications that the system must satisfy. We will use the NuSMV module system and put each of these parts in its own module (and file).

Description

An abstract picture of the passing area is given in Fig. 1.

- The *dashed lines* on the left and on the right are the outside world where trains arrive from and leave to.
- The *line segments* represent tracks and each track is given a label starting with `T`. Each track can be *occupied* or *not*.
- At the end of each track, a *signal* is placed that tells the train driver if it is safe to enter the next track, by showing *green*, or not, by showing *red*.
- Finally, the layout contains four *switches* (formally called sets of points or points) with labels starting with a `P`. A set of points, such as `P1A`, can be *locked in the straight position*, *moving*, or *locked in the curved position*.

When both `P1A` and `P2B` are locked in the curved position, a train can go from `T1A` to `T3B` and back. When a train goes across a set of points that is not locked or locked in the wrong position, then the train could derail, so the interlocking must prevent this. When `P1A` and `P2B` are locked

straight, then trains can safely travel from the track `T1A` across the track `T2A` (which includes the points) to track `T3A` and back. The requirement that `P2B` is locked straight is not to prevent derailment, but is meant to prevent a runaway train on `T3B` from hitting the flank of a train moving along `T2A`.

Interface and Atomic Properties

In order to be able to verify different models, environments, and set of properties independently, they must be written in different files that adhere to an API and then be put together. The API is fixed by the file `main.smv`. This file contains the `Main` module, which instantiates two modules: `Environment` and `Interlocking`. Moreover, it defines the atomic properties that can be used in formulas.

The environment controls the variables that model the world. That is:

- It sets the occupation of the tracks according to what is allowed by the signals, and
- It controls the two signals that control access to the outside world (`S1A` and `S1B`), and
- It simulates the operation of the points.

The interlocking:

- Controls the other signals directly, and
- Can issue orders to the points that the environment must follow.

Note how the inputs are passed: the environment gets the old control values as input and the interlocking gets the newly generated environment inputs as inputs. So updates occur in two steps: first, the environment determines the new inputs; second, the interlocking reacts to the new inputs. The advantage of this is that the outputs of the interlocking can match the newly generated inputs and thus the formulas. In other words, the time needed for the interlocking to react to the environment is hidden in the update order.

The atomic properties that are available are:

T*id*_occupied	Track *id* is occupied.
S*id*_red	Signal *id* is red.
S*id*_green	Signal *id* is green.
P*id*_locked_straight	The set of points *id* is locked in the straight position.
P*id*_locked_curved	The set of points *id* is locked in the curved position.
P*id*_goal_straight	The interlocking orders the environment to move the set of points *id* into the straight position and lock it.
P*id*_goal_curved	The interlocking orders the environment to move the set of points *id* into the curved position and lock it.

Provided Models and Running the Tool

The file `assignment1.zip` contains the given files for this exercise. Open a new terminal, unzip this file and cd into the folder nusmv. In this folder, the central file `main.smv` can be found, as well as a trivial environment `trivial-env.smv`, a trivial interlocking `trivial-int.smv` a file with a few simple use case formulas `properties.smv` and a short file that includes the already mentioned parts in the proper order `trivial-case.smv`. This is done by using the C Pre Processor, so we need to pass the options `-pre cpp` to NuSMV to invoke it.[1] Another useful flag is `-dcx` which suppresses counterexample generation. Finally, if you use the `TRANS` keyword you should check if your model is deadlock free with the `-ctt` option.

You should try all of these options by issuing the command

```
NuSMV -pre cpp -dcx -ctt trivial-case.smv
```

The output ends with

```
############################################################
The transition relation is total: No deadlock state exists
############################################################
-- specification EF T3A_occupied  is false
-- specification EF T3B_occupied  is false
```

if all is well.

Provided Models

To help with the formula writing, we provide three models:

trivial Perfectly safe model because all signals are red and nothing ever happens. See `trivial-case.smv` and included files.

simple Model with a bad interlocking, which works in the given limited environment only. See `simple-case.smv` and included files.

error Model which has an undisclosed error built in. See `error-case.smv` and included files.

[1] Comments can lead to warnings from cpp because they are not legal C code. The work-around is to start comments with `--//`.

Exercises

Note: When you solve the exercises, only modify the files that exercise tells you to modify! Do not change the provided API!

i. **Write the following set of temporal logic properties** that describe the expected behavior of the system as NuSMV specifications. Put these properties in a file `properties.smv`. Do not base these properties on the actual behavior of the provided models; they should describe the **expected behavior** of such an interlocking in general. **(30 pt)**
For each property, provide a **classification of the property** as safety, liveness, or other.
The set of properties that you should specify is as follows:

(a) The signals show either green or red.

(b) The signals show red if passing the signal is unsafe due to occupied tracks or unlocked points.

(c) When a train waits in front of a signal, the signal is not stuck on red forever.

(d) The system never issues conflicting commands.

(e) The points always follow the given commands.

(f) Trains always make progress.

(g) It is always possible to reach a situation in which track `T3A` is occupied.

(h) There is a trace in which trains pass each other in the middle.

(i) Trains never drive over an unlocked point

Invent at least one other useful property yourself. **Specify** this property as a NuSMV specification, provide a **description in natural language** and **classify** the property as a liveness or safety property. In total, you should have at least 10 substantially different properties.

ii. The file `error-combined.smv` contains a model with an error. This model is combined with the formulas in the file `error-case.smv` that can be checked with the command **(5 pt)**

```
NuSMV -pre cpp -ctt error-case.smv
```

Take one of the counter-examples against a **LTL liveness property** and **explain** the order of events **in natural language**. That is, use phrases like 'train moves from track x to track y' and 'signal z turn red/green' to describe what happens in each step of the counterexample.

iii. The module that simulates the points in the given simple environment moves between positions in precisely two steps. Copy the file `simple-env.smv` to the file `delay-env.smv` and **change the points module** in such a way that the movement between positions takes an **arbitrary (i.e. nondeterministic) but finite amount of steps**. Verify that the points work as intended, ignore other errors. The resulting file `delay-env.smv` has to be handed in. The file `delay-case.smv` can be used for running NuSMV on the model. **(10 pt)**

iv. Among the errors when testing with the random delay points there have to be safety errors for the signals. Take the **LTL counter-example** to one of these errors and **explain** the order of events **in natural language**. That is, use phrases like 'train moves from track x to track y' and 'signal z turn red/green' to describe what happens in each step of the counterexample. When the trace reaches a state where the safety property is violated, explain why this is the case. **(5 pt)**

v. **Extend the model** until it handles the case of two trains correctly: **(20 pt)**

- Extend or rewrite the environment from `delay-env.smv` (i.e. the environment in which points take a nondeterministic amount of time to move) until it supports two trains. Put the result in the file `full-env.smv` and hand it in.
- Extend or rewrite the interlocking controller until it correctly controls the passing area in the presence of two trains. This includes that your controller should ensure that the two trains never collide and that the interlocking never reaches a state where a train can no longer make progress. Put the result in the file `full-int.smv` and hand it in.

To avoid some more complicated corner cases, use the following two simplifications:

- You should restrict the arrival of new trains in such a way as to avoid a collision right on entry into the interlocking. I.e. if a train is already exiting or about to exit the interlocking, you may restrict the arrival of new trains on tracks `T0A` and `T0B` respectively. The environment already contains the definitions `may_approach_A` and `may_approach_B` that you can use for this. You are not allowed to make the arrival deterministic. You are not allowed to impose restrictions on the arrival of trains that are stricter than what is outlined above.
- The case where two trains enter the interlocking at the same time from the same side and cross it simultaneously does not count as a collision. We consider the two trains as 'one long train' here. E.g. both trains arrive at `T0A` simultaneously and then both move to `T1A` in the next step, ...

The file `full-case.smv` can be used for running NuSMV on the model.

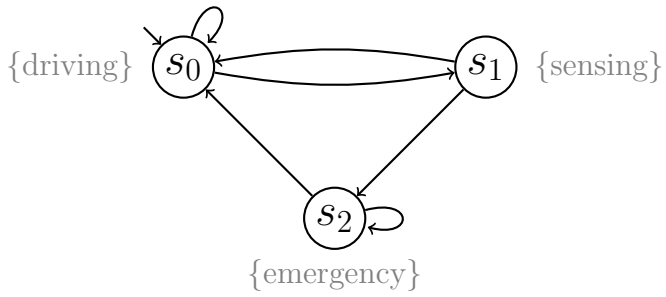

Figure 2: Kripke structure representing the simple robot

Part 2 - Model Checking Algorithms (30 pts)

In this part, we will be using model checking algorithms to verify properties about a simple robot.

Robot Model

Consider a self-driving robot that is trying to avoid obstacles while driving through its environment. While driving, it will periodically stop and scan the environment with its sensors. If it detects nothing, it will go back to driving. If it detects an obstacle nearby, it will go into an emergency mode and take action to avoid the obstacle. Once the emergency has passed, it will return to driving. The Kripke structure representing the decision procedure of this simple robot is displayed in Fig. 2.

Exericses

vi. Use Algorithm 1 (CTL Model Checking) from Chapter 5 of the course book to verify whether the robot fulfils the following CTL formula: **(10pt)**

$$AG\,(emergency \Rightarrow EX\,driving)$$

i.e., verify whether it is always possible for the robot to immediately return to driving when an emergency occurs. **Explain the steps you take**.

vii. Consider the following LTL formula, which encodes that either the robot will always continue driving or there is currently an emergency: **(20pt)**

$$(G\,driving) \vee emergency$$

Construct a tableau for this property. Then use the product automaton with the Kripke structure from Fig. 2 to examine whether the property holds for the simple robot. **Explain the steps you take.**

Hint: The product automaton can be quite large. You only need to construct the reachable part of it.

Part 3 - Summary of files to be handed in

You should hand in is a single ZIP file with the following files:

`report-part-1.pdf`	The report explaining your answers and thought processes to the NuSMV exercises.
`report-part-2.pdf`	The report explaining your solution to the model checking exercises.
	Train interlocking files
`properties.smv`	Properties for validating the passing area models.
`delay-env.smv`	Environment with one train and arbitrary delay points.
`full-env.smv`	Environment with two trains.
`full-int.smv`	Interlocking for environment with two trains.

In all files, and on the first page of both parts of the report, write your names, your student numbers and your group number.