Verification of railway interlocking systems

Simon Busard, Quentin Cappart, Christophe Limbrée,
Charles Pecheur, Pierre Schaus *

Université catholique de Louvain, Louvain-La-Neuve, Belgium

{simon.busard|quentin.cappart|charles.pecheur|pierre.schaus}@uclouvain.be

christophe.limbree@student.uclouvain.be

In the railway domain, an interlocking is a computerised system that controls the railway signalling objects in order to allow a safe operation of the train traffic. Each interlocking makes use of particular data, called application data, that reflects the track layout of the station under control. The verification and validation of the application data are performed manually and is thus error-prone and costly. In this paper, we explain how we built an executable model in NuSMV of a railway interlocking based on the application data. We also detail the tool that we have developed in order to translate the application data into our model automatically. Finally we show how we could verify a realistic set of safety properties on a real-size station model by customizing the existing model-checking algorithm with PyNuSMV a Python library based on NuSMV.

Keywords: Railway interlocking, application data, automatic verification, model checking.

1 Introduction

In the railway domain, an *interlocking* is an arrangement of systems that prevents conflicting train movements in the stations. It is more specially a signalling subsystem that controls the routes, the points and the signals before allowing a train through a station. Computer-based interlockings are configured based on a set of *application data* particular to each station. In this paper, the format considered for the application data is the SSI language [3] that is the electronic interlocking used by the Belgian railways since 1992.

The safety of the train traffic relies on the correctness of the application data. The *European Railway Agency*[1] has edited norms in an effort to harmonize the signalling principles and rules at the European level [14, 5]. Those norms strongly recommend the use of formal methods.

Currently, the application data are prepared manually and are thus subject to human errors. For example, some prerequisite to the clearance (e.g. green light) of the home signal of a route can be missing. This kind of error can easily be discovered by a code review or by testing on a simulator. However, errors caused by concurrent actions (e.g. route commands) are much harder to find. In this case, the combination of possible concurrent actions explodes quickly and testing all possible combinations manually is impracticable.

As testing all the possible scenarios is impossible, the manual validation of the application data relies on a relaxed verification process:

1. The functional tests ensure that the system responds properly to the commands issued by the controller. Those tests are performed by the expert who wrote the application data.

*This research is financed by the Walloon Region as part of the Logistics in Wallonia competitiveness pole.

[1] www.era.europa.eu

J. Pang, Y. Liu, and S. Mauw (Eds.): 4th International Workshop
on Engineering Safety and Security Systems 2015 (ESSS'15)
EPTCS 184, 2015, pp. 19–31, doi:10.4204/EPTCS.184.2

© S. Busard, Q. Cappart, C. Limbrée, C. Pecheur, P. Schaus

2. The safety tests check that each command (e.g. route) is tested and all the conditions that are supposed to impact the command are tested in all their possible values. Those tests are prepared and carried out by an independent tester.
3. The application data are reviewed by the engineer in charge of the project.

During this process, all anomalies are traced in a bug management tool and must be fixed before the interlocking is commissioned.

1.1 Verification of interlockings using model checking

Our approach to improve the manual verification method is to automatically convert the application data into a model and to verify safety properties on that model with a model checker. A model checker is a tool that automatically checks whether a system meets a given property by comparing the reachable states space of the model of the system and the property. In our case, we used the NuSMV [6] symbolic model checker for which our research team[2] has a broad experience. We also used PyNuSMV, a Python library based on NuSMV that can be used to prototype new model-checking algorithms [4]. PyNuSMV gathers the flexibility of Python and the functionalities of NuSMV in order to efficiently manipulate the BDD data structures.

The safety properties are written based on the track layout of each station and on the safety rules applicable in the signalling domain.

Our approach is divided into the following steps:

1. Generate a model of the interlocking based on the application data. This is done by a translator tool.
2. Generate a model of the trains using a Domain Specific Language that encodes the track layout.
3. State all the properties that must be verified to ensure safety based on the track layout.
4. Combine the models of the interlocking, of the trains, and of the properties into an SMV model that can be processed by NuSMV.
5. Use specific model-checking procedures developed with PyNuSMV to reduce the execution time and produce additional data (route compatibility tables).

This process is shown in Figure 1. Our approach is currently only applicable to a single interlocking. Our other assumptions and abstractions are explained in Section 3.

In the next section, we describe the different components used in our model in order to present our model in Section 3. In Section 4, we explain how our model is constructed based on the application data. In Section 5, we detail the safety properties verified by our model. In Section 6, we discuss how we can improve the performance of the verification. References to related work are provided in Section 7.

2 Interlocking components

Figure 2 shows the track layout of the station of *Namêche*, a Belgian town. This station will be our case study for explaining our approach. The whole station is controlled by a single interlocking that controls 14 routes.

On this figure, the following elements can be identified:

[2] `http://lvl.info.ucl.ac.be`

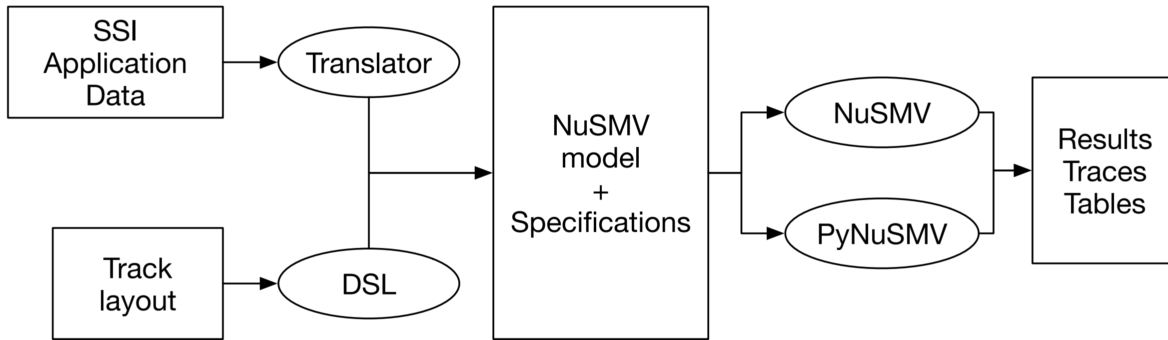

Figure 1: Steps of our approach.

- The track identifiers (e.g. 045).
- The signals (e.g. KM) that are used to grant access to the routes for the trains.
- The points (e.g. P02AM) that are the railway junctions allowing a train to move from one track to another.
- The track circuits[3] (e.g. TC01AM) that are used to detect the vacancy of a portion of the track layout.

The interlocking allows a safe train operation on a railway network or in a station by controlling the *routes*. The routes are the paths followed by the trains when running through a station. For instance, R_KM_045 is a route going from signal KM to track 045. The interlocking handles a route command in the following manner:

1. When a route is requested, it verifies whether the command is safe. This means that the track components (points and track circuits) requested should not be already reserved for another route (the points P01AM, P02BM, P04AM, P04BM, and the tracks TC01AM, TC02BM, TC04BM for R_KM_045).
2. It commands the points by controlling their actuators (points P01AM, P02BM, P04AM, P04BM to the right positions for R_KM_045).
3. It verifies the new status of the points by comparing the command and the replied status of the actuators.
4. It then grants access to the train on the route, setting the origin signal of the route to green (KM for R_KM_045).

A route is composed of several segments called subroutes, corresponding to its track segments (three for route R_KM_045). Each of them is locked when the route is set and is released when the train has fully freed the home track circuit of the subroute, releasing the corresponding points.

This process also makes use of other logical components not shown in Figure 2 like the component materialising a point locking (UIR) or the component recording the train passage on the route ($TISP$). The list of controls and verifications stated above are loaded from the application data and used by the interlocking for every route. The fact that the application data properly reflects the track layout and the signalling principles is thus crucial in the safety that the interlocking can achieve. That is why so much effort is devoted to their verification.

[3] Track circuits are sometimes called track segments.

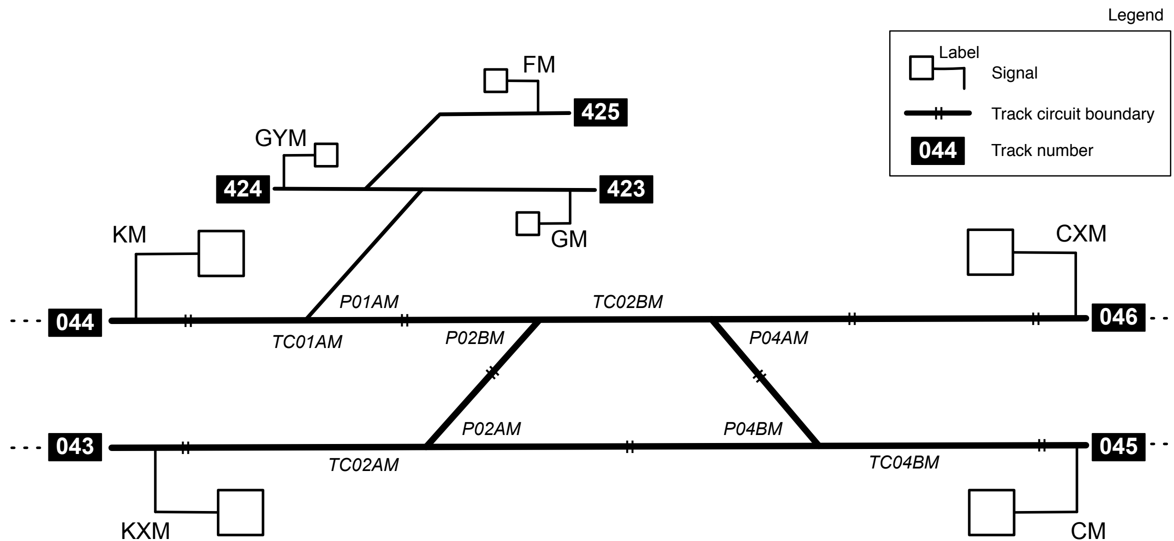

Figure 2: Layout of the Namêche station.

3 Model description

In this section, we describe how the model is designed in order to verify the application data. The complete model can be downloaded from url: `http://lvl.info.ucl.ac.be/Tools/InterlockingModel`.

In order to reduce the size of the state space, several assumptions and abstractions were made:

1. Our method is only applicable to areas controlled by a single interlocking. The case of interlockings interconnected in a network will be studied in our future works.
2. Only two signal aspects are modelled: proceed (green), and danger (red). The trains are supposed to obey the indication given by the signal.
3. The level crossing control and its interaction with the routes is not modelled.
4. The different types of directional locking are not modelled. The directional locking is the mutual exclusion mechanism put in place to prevent head-to-head collisions.
5. The trains can postpone their start when in front of a signal at proceeding aspect but never stop afterwards. The train speed is not modelled.

Our interlocking (SSI) is route based which means:

- A route must be successfully controlled by the controller before a train can run through the station.
- The routes interact with the track side components (e.g.: points, signals).

- The routes using shared resources (e.g.: points) make use of locking variables in order to prevent collisions.
- The path followed by the train is based on the status of the track side components controlled by the routes.

The Figure 3 shows how central the idea of route is in our model. The model is decomposed into NuSMV modules: the interlocking modules and the simulation modules. White modules model the interlocking software components while gray modules model components added to interact with the interlocking.

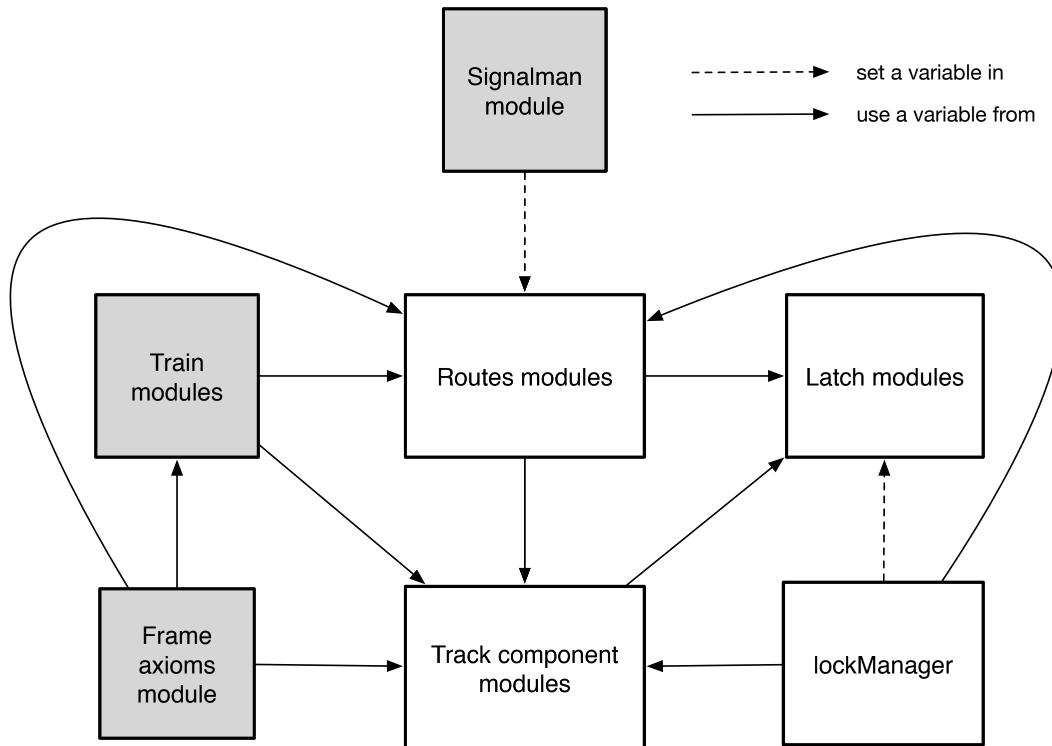

Figure 3: Modules view of the model.

Latch modules: The latches are the global variables shared by the route modules, the point modules, and the lock manager module. The UIR variable is an example of a latch. It is used to lock a point when it is part of a commanded route. The module acts as a record which state is updated by the lock manager module.

Track component modules: The track modules represent a physical component controlled by the interlocking:

- The track segments that hold the state of a track (occupied or clear).
- The points are commanded to the left, or the right position according to the route.

Lock manager module: This module assume the task of locking and unlocking the subroutes and the points when a route is commanded and ran through by a train. This module is a straight emanation from the application data.

Controller module: This module simulates the behaviour of a human commanding the routes. It also ensures that only one route command is issued per transition of the whole system. This module ensures that this behaviour is not violated.

Train module: This module is used to simulate the movement of a train over the adjacent track segments forming the track layout of the station. The track layout is first encoded by mean of a DSL listing all the components like the signals, the track circuits, and the switches. Each component is linked to its siblings taking into account the train direction, and the position of the switches ahead of the train. The graph of the station is then built automatically and all the possible successive train positions are translated into the transitions of the NuSMV module allowing the train simulation.

Route modules: The route lifecycle is described in Section 2. The route modules are a straight translation of the application data from the SSI language to NuSMV. The state machine of a route includes the following states: idle, commanded, proved, and occupied by a train.

Frame axioms module: This module performs three different tasks:

- Changing the status of the track components according to the train movements.
- Triggering a wheel detector when a track segment is occupied.
- Updating the point position after a command.

This module depends on both the application data (routes) and the track layout (trains) to know when the actions must be done and what are the modifications to do.

Given that we want to verify the consistency between the application data and the real track layout, we have to consider a separate source for the application data and the layout. Therefore, unlike the other modules, the train module is not generated from the application data.

Put together, these modules constitute a model simulating the behaviour of an interlocking system as described in the application data and the behaviour of trains according to the track layout. On this model, we can assert and automatically check safety properties with respect to the application data. These properties can be expressed on the state of the trains. For example, a collision occurs if two trains are both located on the same segment. For instance, in Figure 2, such a collision will occur if the application data could allow routes R_KM_045 and R_CM_044 to be set together.

4 Automatic translation of application data

Among all the application data, only a subset is necessary to verify the security of an interlocking system. The rest is either not related to the security or abstracted in our model. Let us now describe the application data used in our model.

Each point can move under a set of conditions. Listing 1 shows how these conditions are represented in the SSI code for a particular point. There are two positions for a point: normal and reverse.[4] Here, the

[4] Normal stands for left and reverse for right.

point P_01AM can be set in a normal position (P_01AM**N**) only if it is is free to move (U_IR(01AM) f). There is a similar rule for the reverse position (P_01AM**R**) .

```
1 *P_01AMN  U_IR(01AM) f /* condition for normal position */
2 *P_01AMR  U_IR(01AM) f /* condition for reverse position */
```

Listing 1: SSI code: Conditions allowing a point to move.

Each route has a set of conditions under which the route request can be granted, and a set of actions that have to be done to fulfil the request. For example, Listing 2 states that the route from Signal CM to Track 044 can only be set if it is not already set (line 2) and if the points are free to be commanded and moved to a certain position (lines 3 and 4). The resulting actions are the setting of the route (line 6), the command of the points (lines 7 and 8) and the locking of the points (line 9). The route and the components requested can be seen on Figure 2.

```
1 *Q_R(CM_044) /* Request for the route CM_044 */
2    if     R_CM_044 xs,
3           P_01AM cfr, P_02BM cfr, P_04AM cfr,
4           P_04BM cfr, P_01BM cfr, P_02AM cfr,
5           U_IR(01AM) f, U_IR(02BM) f, U_IR(04BM) f
6    then   R_CM_044 s
7           P_01AM cr, P_02BM cr, P_04AM cr,
8           P_04BM cr, P_01BM cr, P_02AM cr,
9           U_IR(01AM) l, U_IR(02BM) l, U_IR(04BM) l
```

Listing 2: SSI code: Request for setting a route.

After being locked for a route, the different track components must be freed. According to Listing 3, the subroute U_04M_CM can only be freed if the subroute U_07M_04M is free and if the track T_04BM is clear. There is a set of similar rules for the liberation of the other subroutes and for other components.

```
1 U_04M_CM f if U_07M_04M f, T_04BM c
```

Listing 3: SSI code: Freeing a subroute.

All these data are used to build the NuSMV model. Each interlocking system has its own application data. In other words, we need to build a particular model for each interlocking system. To overcome this issue, we designed a translator which automatically parses the application data and generates the NuSMV model. In this way, we can directly obtain an executable model for each interlocking system. For instance, the NuSMV module corresponding to the route request of Listing 2 is showed on Listing 4.

```
1  MODULE R_CM_044(mainP)
2  VAR
3  cmd : boolean;
4  state : {s, xs}; -- set or unset
5  ASSIGN
6  init(cmd) := FALSE;
7  init(state) := xs;
8  next(state) :=
9     case
10       -- conditions to set the route
11       state = xs & cmd &  -- route not already set
12       mainP.UIR_04BM.st = f & -- track component is free
```

```
13        mainP.UIR_02BM.st = f &
14        mainP.UIR_01AM.st = f &
15        mainP.P_01AM.cfr & -- free to go to reverse position
16        mainP.P_02BM.cfr &
17        mainP.P_04AM.cfr &
18        mainP.P_04BM.cfr &
19        mainP.P_01BM.cfr &
20        mainP.P_02AM.cfr : s;
21        -- conditions to release the route
22        (...)
23    esac;
24 (...)
```

Listing 4: A route command in the NuSMV model

As we can see, this module contains the necessary conditions to set the route. Let us note that the examples presented here do not show all the application data used in our model; other structures such as the train detectors are also used.

5 Safety properties

The safety properties verified on our model are expressed by mean of invariants (properties that are true in any state of the system) and CTL (Computation Tree Logic) formulas. A first set of properties is used to verify that the interaction between the interlocking and the train never ends up in an unsafe sequence causing train collisions or derailments. A second set of properties is used to detect which are the errors in the application data leading to an unsafe behaviour of the interlocking.

Listing 5 shows a sample of properties covering these sets, for route R_CM_044 of the Namêche model.

```
1  INVARSPEC ! (train_1.front = derailed | train_2.front = derailed)
2  INVARSPEC ! (train_1.front = train_2.front)
3  INVARSPEC ! ((train_1.T_01AM | train_2.T_01AM) & P_01AM.willMove)
4  INVARSPEC ! (R_CM_044.st =  s & R_KM_045.st = s)
5  INVARSPEC ! (U_CM_04M.st = l & U_04M_CM.st = l)
6  CTLSPEC AG (T_04BM.st = o & TRP_CM.krc = s -> AX (!R_CM_043.L_CS & !R_CM_044.
     L_CS))
7  INVARSPEC ! (R_CM_044.st = s & U_CM_04M.st = l  & U_04M_07M.st = f)
8  INVARSPEC ! (UIR_01AM.st = l & P_01AM.willMove)
9  INVARSPEC (P_01AM.cmd = P_01BM.cmd)
10 INVARSPEC (R_CM_044.L_CS -> (T_04BM.st = c & T_02BM.st = c & T_01AM.st = c))
```

Listing 5: Safety properties for the Namêche model

Safe interaction between the interlocking and the train This first set embeds the properties verifying that an active simulation of two trains running through the network controlled by the interlocking does not lead to unsafe situations.

- Line 1: Trains never derail. Trains derail when entering a point not set in a position allowing the train to continue its path.
- Line 2: Trains never collide. The property is expressed by stating that the heads of the trains cannot occupy the same position at the same time.
- Line 3: A point (P_01AM) is not allowed to move when its home track-circuit is occupied.

Application data correctness A mistake or an omission in the application data causes the violation of some properties in the first set (e.g. a train collision). However, given a trace leading to a train collision, the identification of which part of the application data is faulty is not trivial. Each property of this second set concerns a route, a part of a route, or a point. As a result, finding the faulty part of the application data in case of violation is easier, as explained hereunder.

- Line 4: Incompatible routes (R_KM_045 and R_CM_044) are never enabled at the same time.
- Line 5: Subroutes in opposite directions (U_CM_04M and U_04M_CM) are never locked at the same time.
- Line 6: The origin signal of a route (R_CM_043 or R_CM_044) immediately goes back to danger after the train has started to run through it. The formula states that in all states (AG) where the train is occupying the track-circuit and has activated the passage detector, the signal is closed in the next state for all possible executions (AX).
- Line 7: The subroutes are released in correct order. In this case, subroute U_04M_07M is not released before subroute U_CM_04M.
- Line 8: A point (P_01AM) will not move when its locking variable (UIR_01AM) is set.
- Line 9: Connecting points (P0_1AM and P_01BM) are always commanded to the same position.
- Line 10: In order to clear the origin signal of a route (R_CM_044.L_CS), all the track circuits must be clear (not occupied by a train).

6 Verification of properties

The model of Namêche station comprises 14 routes, 7 points, and 7 track-circuits for which 132 invariants and 7 CTL formulas were written. These formulas were written manually based on the track layout of Figure 2 for the sake of independence from the application data.

In order to test our model and the adequacy of the properties used on it, we have introduced errors in the application data. The violations were successfully detected and traces were generated. For instance, assigning a wrong locking variable to Point P_01AM can lead to a situation where it is moved under a running train, failing Property 3 in Listing 5. As another example, a missing locking of subroute U_CM_04M leads to a collision between two trains, one going from Signal CM to Point P_04BM and one going from P_04BM to CM, detected by Property 2 in Listing 5.

If we only consider the invariants, the verification with NuSMV takes less than fifteen minutes. However, with CTL formulas it takes few days to complete. To overcome this problem, we implemented custom model-checking algorithms with PyNuSMV.

First, our model includes fairness constraints[5] used to force the trains to eventually progress on the tracks. The trains are modelled such that they can choose to wait at a green signal for an arbitrary long time; fairness constraints are specified to ensure that we consider only executions of the model along which the trains eventually choose to cross a green signal. NuSMV performs model checking of CTL with fairness, which incurs additional computations of fair states and in the verification of some (liveness) formulas. However, the CTL formulas verified on the model are not impacted by the presence of these fairness constraints because all reachable states of the model are fair and no liveness formula is verified.

[5] In the framework of CTL, fairness constraints are sets of states that must be met infinitely often along executions of interest [7].

Thus, to accelerate the verification, standard BDD-based algorithms for CTL without fairness constraints have been used within PyNuSMV instead of those of NuSMV (see for example [7] for more information on these standard algorithms).

Furthermore, these CTL formulas follow the pattern $AG\phi$, where ϕ is a CTL sub-formula, expressing that ϕ is true in all reachable states. NuSMV verifies such formulas by starting from the BDD representing the set of states satisfying ϕ and performing a backward traversal of the system to accumulate the states satisfying the property; then NuSMV compares these states with the BDD of initial states. Nevertheless, this particular case can be improved by checking that all the reachable states satisfy ϕ. This comparison is performed by comparing the BDD of the states satisfying ϕ and the BDD of reachable states, requiring one operation on the two BDDs, instead of the fix-point computation performed by the standard algorithm implemented in NuSMV. Note that this custom approach needs to compute the BDD representing the set of reachable states, but this BDD must be computed to verify the invariants. This BDD is computed by NuSMV itself by performing a standard forward traversal of the model.

Thanks to these custom algorithms, it has been possible to verify the 7 CTL formulas of the previous section within 10 hours, while NuSMV took about 100 hours to verify them. Figure 4 recaps the execution time in function of the number of routes considered.

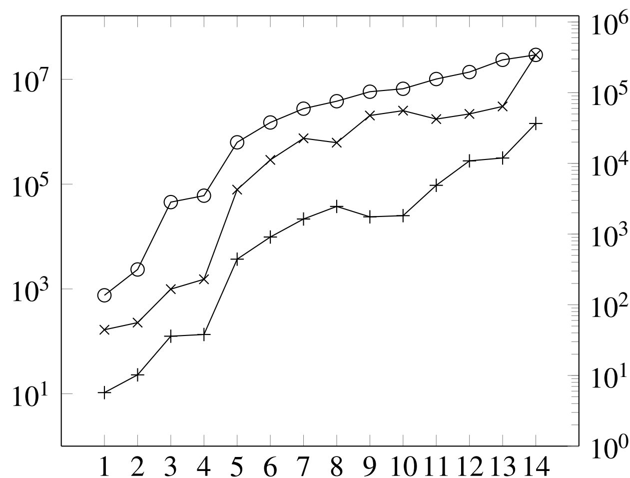

Figure 4: Evolution of number of reachable states ($\circ$) on the left y-axis and verification time for NuSMV ($\times$) and PyNuSMV ($+$) on the right y-axis (in seconds) in terms of number of considered routes.

In order to assess the performance of the custom model-checking algorithms, we have also compared the verification time needed by both tools for models with a reduced number of routes. As we can see in Figure 4, PyNuSMV reduces drastically the execution time when many routes are involved. More precisely, Figure 4 shows the time needed to verify the properties of the previous section with NuSMV and PyNuSMV when the number of considered routes increases. It shows that NuSMV and PyNuSMV behave in similar ways, but PyNuSMV gains an order of magnitude by using custom algorithms.

PyNuSMV can also be used to extract from the model the set of compatible routes. We say that a route is compatible with another route if they can be commanded at the same time, that is, if a train can pass through the first one while another train passes through the second one. Table 1 shows the compatibility table for Namêche (Figure 2). It shows, for example, that Route R_CM_043 and Route R_CXM_044 are compatible while Routes R_KM_045 and R_KM_046 are not. This means that the interlocking system works such that whenever Route R_KM_045 is set, Route R_KM_046 cannot be commanded.

	R_CM_044	R_CM_043	R_CXM_044	R_CXM_043	R_FM_424	R_GM_424	R_GM_044	R_GYM_423	R_GYM_425	R_KM_045	R_KM_046	R_KM_423	R_KXM_045	R_KXM_046
R_CM_044					V	V		V	V					
R_CM_043			V		V	V	V	V	V		V	V		
R_CXM_044					V	V		V	V				V	
R_CXM_043					V	V	V	V	V			V		
R_FM_424							V			V	V	V	V	V
R_GM_424										V	V		V	V
R_GM_044									V				V	V
R_GYM_423										V	V		V	V
R_GYM_425										V	V	V	V	V
R_KM_045														
R_KM_046													V	
R_KM_423													V	V
R_KXM_045														
R_KXM_046														

Table 1: The compatibility table of the station of *Namêche*.

The value V in the table means that the corresponding routes are compatible, otherwise they are not compatible. Given that the table is symmetric, only the top half is presented. Thanks to PyNuSMV, such a compatibility table can be extracted by inspection of the set of reachable states. This table can then be used to check that the routes that should not be compatible are not, giving essential information on the application data under interest. Compatibility sets of more than two routes can be produced in the same way: for the Namêche station, 32 sets of three compatible routes exist (e.g. Routes R_CM_043, R_GYM_425 and R_KM_423 can be commanded at the same time), but no set of more than three compatible routes exists.

7 Related work

In [9], Huber and King demonstrates how five vital safety properties can be verified automatically on SSI application data. They implemented a model checker for Geographic Data by replacing the parser and compiler of NuSMV. The resulting tool, gdlSMV, directly reads Geographic Data and builds a corresponding representation on which model checking is performed using NuSMV's symbolic model checking algorithms. In [10], Mirabadi and Yazdi also use the NuSMV model checker and implement a control table verifier that analyses the contents of control table besides the safe train movement conditions and checks for any conflicting settings in the table. In [15], Winter and Robinson modelled the interlocking by means of the formal notation ASM that are more readable. The formal model is translated in NuSMV code and the Safety requirements are expressed in CTL.

In [13, 11, 12], Moller, Nga Nguyen, Roggenbach, Schneider and Treharne propose to combine the state-based and the event-based (a train passing) approaches by using CSP||B. The overall specification combines two communicating models, one made of CSP process descriptions and one made of a collection of B machines. They also propose the OnTrack tool-set that automates workflows for rail-

way verification, starting with graphical scheme plans and finishing with automatically generated formal models set up for verification. In [2], Abo and Voisin explain how Systerel uses the B language and the OVADO tool to verify large interlocking application data set.

In [8], Fantechi, Fokkink and Morzenti give an overview of the trend in railway interlocking verification.

Compared with the previous works, we presented here an unified approach aiming to verify completely the safety of an interlocking system using model checking. More concretely, our approach has the following features:

- A verification of the correctness of the application data.
- A verification of their consistency with the track layout.
- An automatic generation of the models used for the the verification from the application data.
- A Domain Specific Language used to easily encode a track layout into the models.

Taken separately, such features have already been discussed and considered. But to the best of our knowledge, there is no work that merges all of them into a single framework.

8 Conclusions and future work

In this paper, we have explained how we built a model of a railway interlocking in order to verify the correctness of its application data. We have explained how each module of the model can be automatically generated based on the application data by our generator. We have given the list of safety properties that were verified on our model. Those safety properties are designed to cover the tests and verifications that are currently performed manually on the application data. We have shown that the verification of those properties by a model checker brings improvement in the safety and in the efficiency of the verification process ruling the validation of the application data. Finally, we have shown that the verification of large amount of properties (137) is feasible on a realistic size interlocking by mean of custom algorithms based on PyNuSMV.

In our future work, we will focus on the automatic generation of the safety properties based on the track layout. The aim is to use a railway description language such as RailML [1] from which we can generate the safety rules. Furthermore, the properties we verify on the model have been limited to invariants and safety properties; verifying liveness properties such as *any train entering a route will eventually leave it* needs more effort, and further work is needed to efficiently verify such properties. Besides, until now the model is only designed to verify single interlockings. This assumption holds for relatively small stations such as our case study but is not true for larger stations where several interlockings communicate together. The next step will be to extend the model in order to embed the verification of a set of communicating interlockings. Finally, as we plan to verify larger interlocking systems, we will have to work on increasing the efficiency of model-checking algorithms.

References

[1] (2015): *The XML-Interface for Railway Applications - `http://www.railml.org`*. Available at `http://www.railml.org`.

[2] Robert Abo & Laurent Voisin (2013): *Data Formal Validation of Railway Safety-Related Systems: Implementing the OVADO Tool*. *FM-RAIL-BOK, Workshop 2013, Madrid*, pp. 27–32, doi:10.1007/978-3-319-05032-4_17.

[3] Hubert Bellon (2014): *Data Preparation Guide for Interlocking used in the Belgian Railways*. Infrabel - technical references N20.

[4] Simon Busard & Charles Pecheur (2013): *PyNuSMV: NuSMV as a Python Library*. In Guillaume Brat, Neha Rungta & Arnaud Venet, editors: Nasa Formal Methods 2013, LNCS 7871, Springer-Verlag, pp. 453–458, doi:10.1007/978-3-642-38088-4_33.

[5] CENELEC (2011): *EN50128 - Railway applications - Communication, Signalling and Processing Systems - Software for Railway Control and Protection Systems*.

[6] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani & Armando Tacchella (2002): *NuSMV 2: An OpenSource Tool for Symbolic Model Checking*. In Ed Brinksma & KimGuldstrand Larsen, editors: Computer Aided Verification, Lecture Notes in Computer Science 2404, Springer Berlin Heidelberg, pp. 359–364, doi:10.1007/3-540-45657-0_29.

[7] Edmund M. Clarke, Orna Grumberg & Doron Peled (2001): *Model checking*. MIT Press, doi:10.1016/B978-044450813-3/50026-6. Available at `http://books.google.de/books?id=Nmc4wEaLXFEC`.

[8] Alessandro Fantechi, Wan Fokkink & Angelo Morzenti (2012): *Some Trends in Formal Methods Applications to Railway Signaling*, pp. 61–84. John Wiley & Sons, Inc., doi:10.1002/9781118459898.ch4.

[9] Michael Huber & Steve King (2002): *Towards an Integrated Model Checker for Railway Signalling Data*. Springer-Verlag Berlin Heidelberg 2002, p. 20, doi:10.1007/3-540-45614-7_12.

[10] Ahmad Mirabadi & Mohammad B. Yazdi (2009): *Automatic Generation and Verification of Railway Interlocking Control Tables using FSM and NuSMV*. Transport Problems : an International Scientific Journal 4, pp. 103–110. Available at `http://www.transportproblems.polsl.pl/pl/Archiwum/2009/zeszyt1/2009t4z1_13.pdf`.

[11] Faraon Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve Schneider & Helen Treharne (2012): *Combining Event-based and State-based Modeling for Railway Verification*. Computing Sciences Report.

[12] Faraon Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve Schneider & Helen Treharne (2012): *CSP||B Modelling for Railway Verification: The Double Junction Case Study*. Proceedings of the 12th International Workshop on Automated Verification of Critical Systems.

[13] Faron Moller, HoangNga Nguyen, Markus Roggenbach, Steve Schneider & Helen Treharne (2013): *Defining and Model Checking Abstractions of Complex Railway Models Using CSP||B*. In Armin Biere, Amir Nahir & Tanja Vos, editors: Hardware and Software: Verification and Testing, Lecture Notes in Computer Science 7857, Springer Berlin Heidelberg, pp. 193–208, doi:10.1007/978-3-642-39611-3_20.

[14] George Raymond (2014): *Where are the CENELEC standards going ?* IRSE News Issue 203, pp. 21–23.

[15] Kirsten Winter & Neil J. Robinson: *Modelling Large Railway Interlockings and Model Checking Small Ones*. In: In Michael Oudshoorn, editor, Twenty-Fifth Australasian Computer Science Conference (ACSC2003), pp. 309–316.