A Tool-Chain for the Verification of Geographic Scheme Data

Madhusree Banerjee[2], Victor Cai[2], Sunitha Lakshmanappa[2], Andrew Lawrence[2], Markus Roggenbach[1], Monika Seisenberger[1], Thomas Werner[2]

[1] Swansea University, Wales, UK
[2] Siemens Mobility Limited

Abstract. The Engineering Data Preparation System (E-DPS) is a tool-chain produced by Siemens Mobility Limited for digital railway scheme design. This paper is concerned with the creation of a tool able to formally verify that the scheme plans follow the design rules required for correct European Train Control System (ETCS) operation. The E-DPS Checker encodes the scheme plan and signalling design rules as an attributed graph and logical constraints over that graph, respectively. Logical constraints are verified by the E-DPS Checker using the satisfiability modulo theories solver Z3. This approach verifies the configuration of ETCS for a particular scheme and reduces the amount of principles testing and manual checking required. The E-DPS Checker is currently being developed to EN50128 basic integrity and has been applied to verify the correctness of a number of real-world scheme plans as part of the development process.

1 Introduction

Railway verification with formal methods has a long history [7]. Often, verification concerns the dynamic aspects of rail movement. However, there are also verification challenges with regards to the static design of railways. Given the topological track layout in form of a track plan, a scheme plan provides a signalling design (that can include elements of conceptual nature, e.g., routes, as well as where to place track side equipment, e.g., balises). In this paper, we consider the question of how to verify if scheme plans follow a set of design rules, which arise from railway standards and safety concerns. We formally represent both, scheme plans and design rules, and implement a tool chain to automatically verify if a scheme plan complies with the desired properties. As a speciality, we also represent counterexamples in a visual way, with a view to bridge the gap between the shape of the formal verification result as a logical formula and the domain specific language of railways.

Scheme plans were originally created by survey with engineers measuring the track layout and the location of equipment by hand. Modern surveys are performed using a LIDAR scanning train which generates a highly accurate digital map of the railway. The detailed map is encoded in a file format that is easy for both humans and machines to process. The existing process for checking is laborious, the data and changes are manually reviewed by inspecting the files. This is made worse by the fact that the scheme design follows an iterative process, in which a human reviewer may end up checking the same files repeatedly. As human beings are weak at performing

repetitive tasks with subtle differences between each required check, fatigue can set in and the possibility for human error increases.

Modern railway signalling systems, such as the European Train Control System (ETCS), are designed using accurate geographical maps of the railway derived from scheme plans. The maps contain the topology of the tracks, positions of signalling equipment, and conceptual constructs such as train routes. The safe operation of the signalling systems requires that the geographic maps, and the signalling schemes that they represent, reflect the safety principles of the system.

In this paper we will consider two, medium-size, real world examples of scheme plans, one which is an extension to an existing development, and one which is a new development. One of these plans has about 300 passive position beacons (so-called balises). If one was to naively checking if all pairs of balises would fulfil one layout criterion, one would have to perform nearly one hundred thousand checks. The challenge is to design a tool-chain which is capable of automatically verifying such number of checks within an acceptable time, say, within less than one hour per scheme plan. Fig. 1 presents our tool-chain.

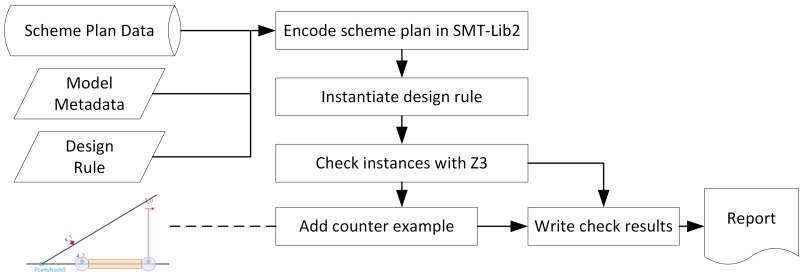

Fig. 1: Geographic Scheme Data Verification Tool-Chain

Our tool-chain takes as an input a scheme plan and formally represents it as a labelled graph in SMT-Lib2 [3]. The next step is to identify for which elements of the scheme plan a specific design rule applies: a design rule can be seen as a pattern, which can be instantiated using these elements. Each instantiation yields a check which is passed on to a verification process with the SMT Solver Z3 [12] at its heart. After possibly several calls to Z3, this process produces a three-valued output of Fail/Pass/Unknown. In case the check fails, we produce a visualisation of a counter example. The final output is a report comprising of all results of the checks for one rule.

Our paper is organised as follows. In Section 2 we briefly discuss the topics of scheme plans and SMT solving. In Section 3 we give some example design rules and detail the formalisation process that represents them in first order logic. In Section 4 we describe rule instantiation for a scheme plan and how to address our verification challenge using SMT solving. In Section 5 we present how counterexamples found in the verification process can be visualised for rail engineers. In Section 6 we provide performance results based on real world scheme plans. In Section 7 we give pointers to related literature and discuss how our approach is different.

2 Background

The E-DPS system utilises several representations for processing scheme plan data. The model on which we perform automated reasoning (cf. Scheme Plan Data of Fig. 1) is the so-called node edge model (NEM, cf. Definition 1) which allows for automated translation to SMT-Lib2 (see Section 2.1). An NEM is an attributed graph, where generic attributes store data associated with the various scheme plan elements. Fig. 2 shows such an NEM with 2 balises placed on a passing loop. The balises are represented as position objects, which have a location within the topology. All objects in the model are attributed with additional information, including the type of the signalling object, its identifier, and any other data that would be contained within the scheme plan.

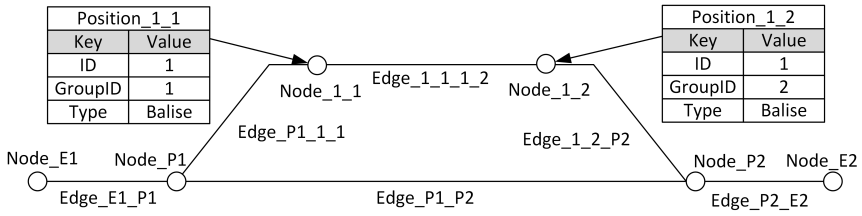

Fig. 2: Example of an NEM containing nodes attributed with balise information.

Definition 1 (Node Edge Model). *A node edge model is a triple* (N, E, P) *where* N, E, *and* P *are sets of nodes, edges and positions, respectively. Edges have a weight* $length(e)$. *Every position* p *has an associated* $node(p)$. *Nodes, edges and positions are referred to as object types in the model and can be attributed with additional information.*

2.1 SMT Solving

The Boolean satisfiability problem (SAT) [6] is foundational to theoretical computer science. It can be stated as follows: given a Boolean formula φ does there exist a model M assigning truth values to the variables of φ such that φ evaluates to true? Tools to solve this problem are commonly referred to as SAT solvers. Typically, SAT solvers produce either a satisfying assignment in the case that φ is satisfiable or a derivation demonstrating that φ is unsatisfiable.

Satisfiability modulo theories (SMT) is a generalization of SAT to solve additional types of problems. In SMT solving, the Boolean formula φ is replaced with a many-sorted first order logic formula over a set of theories T, e.g., for numbers, arrays, and strings. The result of SMT solving is three valued: *satisfiable*, if the solver could find a model, *unsatisfiable*, if the solver could show that there is no model, and *unknown*, if the solver's proof procedures were unable to come to a decision within the user-defined timeframe.

E-DPS Checker uses the SMT-Lib2 [3] standard to encode scheme plans and desirable properties. SMT-Lib2 is tool independent. For SMT solving, E-DPS Checker currently uses Z3 [12] developed by Microsoft Research. As an industrial user we want

to use tools that are widely adopted and offer a degree of stability. For performance reasons, in the E-DPS Checker we disable the ability of Z3 to generate a model, i.e. a *satisfiable* result, and can only obtain either *unsatisfiable* or *unknown* from an instance check with of Z3 (cf. Fig. 1).

3 Design Rules and Their Formalization

The Radio Block Centre (RBC) is one of the many safety-critical components of ETCS level 2. Data preparation for the RBC includes the provision of a scheme plan, detailing, e.g., the specific locations of balises. The placement of track equipment must meet strict layout requirements to ensure safe operation. There are various sources of these layout requirements. For example, the *Futur* series of RBC designed by Siemens comes with vendor-specific requirements. These ensure the correctness of product-specific implementation of ETCS functions. Other examples are project or area specific requirements that are determined by local infrastructure managers or standard bodies. In the following we provide two example rules concerning the placement of balises. *The BG-03 design rule*[3] states that design placement of balises should avoid points and crossings. Designed spacing shall be constrained by:

1. ≥ 1.0m between balise and point toes.
2. ≥ 1.0m between balise and point frog.
3. ≥ 1.4m lateral separation between a balise on one path and the centre line of the other path.
4. No balises between the toe and frog of set of points.

The rationale for this rule is as follows. If a balise is present very near to a point node or a diamond node, then the metal in the point may interfere with the reading of the balise. Also, the lateral separation between two balises should be greater than 1.4 m. As, if the two balises are placed too close to each other, then the train can read the information from the wrong balise which can lead to wrong-side failure and in turn could also lead to collision between trains.

The BG-05 design rule[4] states that the designed minimum spacing between adjacent balise groups shall be constrained by: ≥ MIN_BG_SEPARATION between adjacent end balises, one at each end of the two groups. MIN_BG_SEPARATION is a numeric value which describes the minimum distance which must be present between adjacent end balises, one at each end of the two groups. This rule shall prevent trains from missing a reading, as could happen if adjacent end balises of two balise groups are placed too close to each other. Additionally, balises are expensive and there is an engineering trade off to be made between the accuracy of train positioning and cost.

Our process of formalising properties consists of several steps, that finally lead to an XML file of design rules, see Fig. 1. In the first step, railway engineers perform a

[3] The lateral separation part is derived from [2] Subset-036 v3.1.0 Table 1 'One Balise and one Antenna Unit'. The other parts are derived from [2] section 5.7.10.

[4] This requirement originates from subset-040 (Dimensioning and Engineering rules) [1] section 4.1.1.from the ETCS specification documents.

refinement of the *source requirements*, which usually come in the form of text documents. In this step, they re-describe the desired behaviours or restrictions in terms of the data structures or functions of the target RBC or interlocking. This yields an *intermediate document*.

In the second step, software engineers provide a clear mapping between the terms of the intermediate document and NEM elements. Using the BG-05 requirement as an example, the terms 'balise group' and 'balise' correlate to the 'BaliseGroup' and 'Balise' object types of the NEM.

The requirements of the intermediate document are still given as natural language descriptions, which – for the sake of formal verification – need to be captured in an unambiguous mathematical notation. Here, we chose many-sorted first order logic which enables us to define operators over the generic sorts of the NEM. These operators include, e.g.: $\mathrm{distance}$ which is a function representing the distance between two positions or nodes; and $\mathrm{adjacent}$, which is a predicate that indicates that there is a path between two objects with no other objects of the same type on that path.

To determine which balise pairs should be checked for BG-05, we need to consider several conditions: balises b, b' should not be in the same balise group; b, b' should be at the ends of their respective balise groups; and b, b' should be adjacent. These considerations finally lead to our formalisation of BG-05:[5]

$$\forall b, b' \in \mathbf{Set_{Balise}}.\mathrm{adjacent}(b, b') \rightarrow \mathrm{balise\_group\_id}(b) \neq \mathrm{balise\_group\_id}(b') \rightarrow \mathrm{distance}(b, b') \geq \mathbf{MIN\_BG\_SEPARATION}$$

Such design rule formulae are stored as an XML file for use by the checker tool, cf. Fig. 1. Modelling assumptions and mappings are recorded in an accompanying document. In a final step, the railway engineers who authored the intermediate document review the formalisation. This includes checking example schemes plans that are expected to pass or fail the design rules.

4 Verification Approach

The verification process starts by encoding both the scheme plan and the design rule into SMT-Lib2 internally inside the checker. Each element of the scheme plan is encoded as an NEM object in SMT-Lib2 representation. The transformation process is complex with several stages and additional entities. Here, we provide the core constructions. Once encoded, the design rules are instantiated for a particular scheme plan, removing the quantifiers and constructing a number of design rule instances. Following the instantiation, the checker performs an iterative deepening search to analyse the instances with reachability axioms, cf. Fig. 1.

4.1 Quantifier Instantiation and Sub-formulae Elimination

Our E-DPS Checker carries out a number of preprocessing steps in C# prior to executing the SMT solver, see step "Instantiate the design rule" in Fig. 1. Quantifiers

[5] Due to our definition of adjacency (no intermediate balise objects), we do not need a specific 'end balise' relation in the formalisation of this design rule.

fall into the semi-decidable fragment of first order logic and therefore in general are hard to reason about for SMT solvers. When the value of the quantified variable is from some finite domain like a finite set then it is possible to iterate over all concrete values of the variable and write an equivalent logical expression without the quantifier: $\forall x(x \in S \rightarrow P(x))$ can equivalently be replaced by $\wedge_{t \in S}\ P[t/x]$. Formulae with false premises are trivially true and can be eliminated from any conjunction: $P_1 \wedge \ldots \wedge P_{n-1} \wedge (\bot \rightarrow P_n)$ can equivalently be replaced by $P_1 \wedge \ldots \wedge P_{n-1}$. In practice, our E-DPS Checker generates a set of formulae referred to as assumptions during model translation. These are used during the false premise elimination phase to reduce the number of checks required from the order of hundreds of thousands to thousands of checks.

4.2 Reachability

A large number of the design rules formalized as part of the checker development require that signalling elements have either a minimum or maximum separation and are referred to as reachability constraints. The standard way to reason about reachability is to use a breadth-first search algorithm to find the shortest path between two nodes. The E-DPS Checker takes a declarative approach that simulates the performance benefits of breadth-first search. It uses SMT solving to construct a traversal tree starting from the source node of the search and incrementally deepens the tree until it can infer the required information. This has taken inspiration from iterative deepening depth-first search [9] which is an approach to simulate the behaviour of breadth-first search with a depth-first search algorithm. It has as an additional benefit that Z3 computes a witness to the design rule in the form of a traversal tree or path, which would be missing if an imperative algorithm was used.

Definition 2 (Traversal Tree). *A k-traversal tree* $\mathrm{traversal\_tree_k}(n_0, n_{end})$ *from a source node* n_0 *to a destination node* n_{end} *is the set of all path segments up to length* k*. A path segment in this context may end on the destination node or earlier.*

In our first attempt, we naively formalized the standard definition of reachability, however, this caused the SMT solver to blindly generate all paths from a source node to the destination and was highly inefficient. The initial focus of the checker was to prove reachability constraints with minimal separation with other kinds of properties being in scope over the longer term. To this end, we have developed an axiomatisation of reachability that could infer minimal separation by only analysing the immediate vicinity around the source node.

Definition 3 (Reachability). *We define a k-shortest segment in a k-traversal tree as:* $\forall k.\, \forall n_0, n_e.\, \forall p \in \mathrm{traversal\_tree_k}(n_0, n_e).\, \forall q \in \mathrm{traversal\_tree_k}(n_0, n_e)(\mathrm{length}(p) \leq \mathrm{length}(q)) \rightarrow p \in \mathrm{shortest\_seg_k}(n_0, n_e)$.

Distance is defined using 3 axioms, here we only present the axiom for validating minimum separation, the other axioms follow a similar pattern. The axiom states that if we have a k-shortest segment p and the end of the segment is not the destination node, then we can infer that the distance between n_0 and n_e is at least $\mathrm{length}(p)$*:* $\forall k. \forall n_0, n_e. \forall p \in \mathrm{shortest\_seg_k}(n_0, n_e)\,(\mathrm{end}(p) \neq n_e \rightarrow \mathrm{distance}(n_0, n_e) \geq \mathrm{length}(p))$.

The impact of this axiomatisation is that the majority of checks can be achieved through a low-bound in less than a second per instance.

Lemma 1 (Monotonicity of Minimum Separation). *The minimum separation axiom is monotonic: if the two nodes are inferred to be minimally separated at k then minimum separation would also hold at all subsequent $k' > k$:*

$$\forall k, n_0, n_e. \exists p \in \text{shortest\_seg}_{\text{k}}(n_0, n_e)\,(\text{end}(p) \neq n_e \wedge \text{distance}(n_0, n_e) \geq \text{length}(p)) \rightarrow \\ \exists q \in \text{shortest\_seg}_{\text{k+1}}(n_0, n_e)\,(\text{distance}(n_0, n_e) \geq \text{length}(q)).$$

The other reachability axioms have similar correctness arguments.

4.3 Instance Checking Process

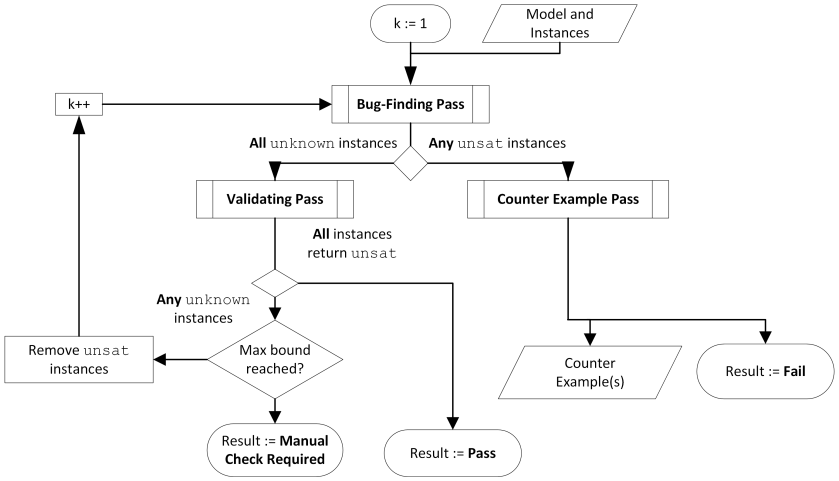

Fig. 3: Instance Checking Process

The E-DPS Checker runs an iterative process as shown in Fig. 3 to infer whether a given design rule instance holds, resulting in one of three overall results for a design rule **Manual Check Required**/**Pass**/**Fail**. A result of **Manual Check Required** indicates that the solver was unable to decide one or more instances, however the rest of the instances complied with the design rule. A **Pass** requires that all design rule instances were successfully validated, whereas a **Fail** indicates that there were one or more instances where a counter example was produced demonstrating that the design rule does not hold. Checking the individual instances one at a time increases the performance, the model axioms are only instantiated with the instances being checked and the topological constructs under the bound of the search. The process starts by assigning 1 to the bound k and takes as input the SMT-Lib2 representation of the NEM model and design rule instances. Then up to three runs of Z3 are made:

Bug-finding Pass: Checks whether the design rule instance is incompatible with the encoded node edge model. If the SMT solver returns `unsat` then there exists a counter example which can be discovered by the counter example pass. The checker has inferred that the instance does not comply with the design rule.

Validating Pass: Checks whether the negated design rule instance is incompatible with the encoded node edge model. If the SMT solver returns `unsat` then the design rule instance and the model are compatible. The checker has inferred that the instance complies with the design rule.

Counter Example Pass: Generates an unsat core which forms a counter example indicating which model elements caused the violation of the design rule.

If the bound has not been reached, then all the validated instances are removed, the current value of the bound k is incremented, and the process is rerun. The maximum bound is a user configurable constant.

5 Counterexample Visualisation

Our tool-chain has the capability to visualise counterexamples, see the box "Add counterexample data" in Fig. 1. Through a focus group study [10] with the intended user

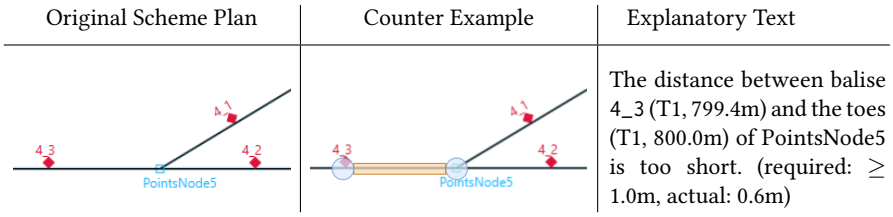

Fig. 4: Counterexample Visualisation for a violation of the BG-03 design rule

group of railway engineers, we determined of how to visualise counterexamples and accompany them with an explanatory text, see Fig. 4 for a typical example of our final design: on the left, it shows a scheme plan with a wrongly placed balise (4_3), in the middle it shows the visualisation of the counter example highlighting in NEM the nodes involved (in blue) and the distance that needs changing (in yellow), on the right a text provides an explanation of the mistake.

In case the algorithm in Fig. 3 returns that a design rule is incompatible with a given scheme plan, the *Counter Example Pass* generates a so-called unsat core[6] from which we extract information about the found counterexample. For instance, the unsat core for the violation shown in Figure 4 looks as follows:

```
(distance_to_distancebound_2 def_edge_T1_node_4_3_PointsNode5
 def_pointsnode_PointsNode5 def_position_Balise_4_3 rule2)
```

[6] Given an unsatisfiable Boolean propositional formula in conjunctive normal form, a subset of clauses whose conjunction is unsatisfiable is called an unsatisfiable core.

The transformation from such an unsat core to its visualisation involves several steps. Thanks to a consistent naming scheme of the axioms in SMT-Lib2 it is possible to identify which elements of the NEM are violating a rule. For instance, labelled elements of the scheme plan have axioms names with the prefix `def_`. Thus, by parsing the above unsat core, we know that the violating elements are PointsNode5 and the balise `4_3`, and that the violating distance involves the edge called `T1_node_4_3_PointsNode5`.[7] This allows us to produce explanatory text. This information is also stored in an XML report, which is read and rendered by the E-DPS Editor. Here, an XAML file contains a specification of how to display the various elements contained in such a report.

6 Analysis of Performance on Real World Examples

To analyse performance of the automated checks, we have applied balise group placement rules BG-03 and BG-05 against real-world example railway scheme plans.[8] Our real-world railway scheme plans include all the physical components (tracks, signals, points etc.) as well as the logical components like routes, subroutes and locking conditions. For the performance analysis we have considered two real-world railway scheme plans, one example from new development (ND) and another an extension to an existing development (ED). Below are the features of the example railway scheme plan considered for the performance analysis:

Railway Scheme Plan	Tracks	Signals	Block Markers	Points	Balises
ND	10	27	21	12	72
ED	42	44	0	32	237

The following table documents the comparison of automated testing against manual process of checking of the design rules against the real-world railway scheme plan. Timing is provided in seconds. Note that the data listed for the manual checking process is not actually a measured time; it is rather an estimation given by the time to perform the required calculations multiplied by the number of checks needed. Here, we made the assumption that it takes a human operation one second to perform one arithmetic operation.

Railway Scheme Plan	Design Rule	Automated Check (s)	Manual Check (s)
ND	BG-03 (part 1)	31.724	576
ND	BG-03 (part 2)	63.807	209162
ND	BG-05	191.574	5184
ED	BG-03 (part 1)	34.392	7584
ED	BG-03 (part 2)	85.096	1824962
ED	BG-05	207.949	56169

[7] In more complex scheme plans, the connection between the violating elements can consist of several edges, in our example it involves only one edge.

[8] The following is the configuration of the machine used to run the automated tests: ZBook Fury 15 G7 Mobile Workstation, Microsoft Windows 10 Enterprise OS, x64-based PC, Intel® Core™ i7-10850H CPU @ 2.7GHz with 6 cores.

The primary benefit with automated checking is that it takes less time to perform the checks. As can be seen from the table above, automated checking is faster by at least one order of magnitude. A further benefit is that it is guaranteed that all the faults will be found by automated checking. A human checker might overlook a combination. Also, human errors due to repetitive work are eliminated. Furthermore, automated checks cater for re-design of scheme plans. Though in the examples of ED and ND, the automated checking did not reveal any new mistakes, it is often the case that errors in scheme plans are found at later stages of the scheme design and were costly to resolve. Automated checking guarantees that errors are found early on.

7 Related Work

Formal methods have been applied to verify both traditional and more modern signalling systems. For instance, the specification of ETCS has been verified in [13], and European Rail Traffic Management System in [5]. In [8], Idani et al. developed an approach to modelling railway topologies and signalling systems. Similarly to them, our work involves both graphical DSLs and formal methods, but they check *dynamic* properties while our work verifies *static* properties of the infrastructure. In [11], Luteberget developed a tool suite named *Junction*, which features verification of infrastructure data for consistency and compliance with rules and regulations encoded in a knowledge base. We took inspiration from this work for visualising counterexamples. One difference is that we are using first order logic to express properties, while Luteberget is restricted to Horn clauses. Our choice of technology is driven by the need to have a uniform formal methods framework: Siemens Mobility has started to build a number of tools around SMT solving.

8 Summary and Future Work

We have presented a tool-chain that scales to the verification of static properties of real-world scheme plans. The tool-chain is based on SMT solving and utilizes the Z3 solver. Thanks to optimisations of properties and the strategy of which properties to check first, verification time is kept small. The formal method is made applicable by counterexample visualisation which was developed involving railway engineers as end users.

Parallel deployment with the manual process is future work. We further plan to utilize satisfiability modulo monotonic theories [4] for model generation, currently Z3 is unable to generate such models. This will enable extending this approach to automated test data generation and automatic scheme plan design. The design rule instantiation approach currently relies on the trustworthiness of the C# code, further investigation of the interplay between pre-computing solutions and checking solutions is desirable. Another approach to further increase integrity will be to pursue proof checking and proof reconstruction.

Acknowledgement The authors would like to thank Peter Woodbridge, Simon Chadwick and Mark Thomas for providing valuable advice and feedback.

References

1. Dimensioning and Engineering rules. Tech. rep., https://www.era.europa.eu/system/files/2023-01/sos3_index013_-_subset-040_v340.pdf
2. FFFIS for Eurobalise. Tech. rep., https://www.era.europa.eu/system/files/2023-01/sos3_index009_-_subset-036_v310.pdf
3. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa (2021), http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2021-05-12.pdf
4. Bayless, S., Bayless, N., Hoos, H., Hu, A.: SAT Modulo Monotonic Theories. Proceedings of the AAAI Conference on Artificial Intelligence **29**(1) (2015)
5. Berger, U., James, P., Lawrence, A., Roggenbach, M., Seisenberger, M.: Verification of the European Rail Traffic Management System in Real-Time Maude. Science of Computer Programming **154**, 61–88 (2018)
6. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press (2021)
7. Fantechi, A.: Twenty-Five Years of Formal Methods and Railways: What Next? In: Counsell, S., Núñez, M. (eds.) Software Engineering and Formal Methods. pp. 167–183. Springer, Cham (2014)
8. Idani, A., Ledru, Y., Ait Wakrime, A., Ben Ayed, R., Bon, P.: Towards a Tool-Based Domain Specific Approach for Railway Systems Modeling and Validation. In: Collart-Dutilleul, S., Lecomte, T., Romanovsky, A. (eds.) Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification. pp. 23–40. Springer International Publishing, Cham (2019)
9. Korf, R.E.: Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence **27**(1), 97–109 (1985)
10. Krueger, R.A.: Focus groups: A practical guide for applied research. Sage publications (2014)
11. Luteberget, B.: Automated Reasoning for Planning Railway Infrastructure. Ph.D. thesis, Faculty of Mathematics and Natural Sciences, University of Oslo (2019)
12. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
13. Platzer, A., Quesel, J.D.: European Train Control System: A Case Study in Formal Verification. In: International Conference on Formal Engineering Methods, ICFEM 2009: Formal Methods and Software Engineering. Lecture Notes in Computer Science, vol. 5885, pp. 246–265. Springer Berlin Heidelberg (2009)