Applied Bounded Model Checking for Interlocking System Designs

Anne E. Haxthausen[1], Jan Peleska[2], and Ralf Pinger[3]

[1] DTU Compute, Technical University of Denmark, `aeha@dtu.dk`
[2] Department of Mathematics and Computer Science, Universität Bremen, Germany, `jp@informatik.uni-bremen.de`
[3] Siemens AG, Braunschweig, Germany, `Ralf.pinger@siemens.com`

Abstract. In this paper the verification and validation of interlocking systems is investigated. Reviewing both geographical and route-related interlocking, the verification objectives can be structured from a perspective of computer science into (1) verification of static semantics, and (2) verification of behavioural (operational) semantics. The former checks that the plant model – that is, the software components reflecting the physical components of the interlocking system – has been set up in an adequate way. The latter investigates trains moving through the network, with the objective to uncover potential safety violations. From a formal methods perspective, these verification objectives can be approached by theorem proving, global, or bounded model checking. This paper explains the techniques for application of bounded model checking techniques, and discusses their advantages in comparison to the alternative approaches.

Keywords: railways, interlocking systems, formal methods, verification, bounded model checking, temporal logic

1 Introduction

Formal methods have been applied for years in the railway domain and reached a level that enables the compilation of the body of knowledge in the form of an engineering handbook (in the style of [1]), recording case-based "best practices". To this end, this paper contributes knowledge concerning verification and validation (V&V) of interlocking system designs. First we outline the state-of-the-art of V&V tasks and formal methods for performing them. Then techniques for applying one of these methods (bounded model checking) are explained in more detail.

1.1 Interlocking V&V – State-of-the-art

Software controlling interlocking systems has to be verified on two levels. The first level focuses on the correctness of configuration data specifying how the topology of the railway network controlled by the interlocking system is reflected by re-usable software objects, their interfaces, and their instantiation data. Correctness of the configuration data ensures that the software has adequate control over the electro-mechanical components of the physical interlocking system. In terms of computer science, physical railway network layout may be regarded as formal models conforming to some graph grammar. The associated software configurations are correct if they conform to a similar grammar where physical language objects – e.g., a point – have been replaced by language objects representing software components – e.g., the software instance representing a point. Apart from grammatical well-formedness, additional rules concerning proper parameterisation of objects apply. All in all, checking the correctness of interlocking configurations corresponds to checks of model syntax and static semantics. The second verification level investigates the safety of trains passing through the controlled network area. The verification objective is to prove the absence of hazardous situations in the network, provided that all trains follow the restrictions (signals, speed limitations) imposed by the interlocking system. Extending the object attributes of the static software configuration by dynamic state information – e.g., whether trains reside on track elements, or the switch state of a signal or a point – the object configuration is turned into a model with both static and behavioural semantics. The latter specifies the potential dynamic changes of the interlocking system configuration – e.g, a train leaving one track segment and entering another.

Interlocking systems are designed according to different paradigms [21, Chapter 4]. Two of the most widely used ones are (a) *geographical interlocking systems* and (b) *route-based interlocking systems* using interlocking tables. For design type (a), routes through the railway network can be allocated dynamically by indicating the starting and destination points of trains intending to traverse the railway network portion controlled by the interlocking system under consideration. In the original technology, electrical relay-based circuits were applied, whose elements and interconnections where designed in one-to-one correspondence with those of the physical track layout. The electric circuit design ensured dynamic identification of free routes from starting point to destination, the locking of points and setting of signals along the route, as well as on neighbouring track segments for the purpose of flank protection. In today's software-controlled electronic interlocking systems, instances of software components "mimic" the elements of the electric circuit. Typically following the object-oriented paradigm, different components are developed, each corresponding to a specific type of physical track element, such as points, track sections associated with signals, and others with axle counters or similar devices detecting trains passing along the track. Similar to connections between electric circuit elements, instances of these software components are connected by communication channels reflecting the track network. The messages passed along these channels carry requests for route allocation, point switching and locking, signal settings, and the associated responses acknowledging or rejecting these requests. The software components are developed for re-use, so that novel interlocking software designs can be realised by means of configuration data, specifying which instances of software components are required, their attribute values, and how their communication channels shall be connected. The geographical approach to interlocking system design induces a separate verification and validation (V&V) step which is called *data validation*. Its objective is to check whether the instantiation of software components is complete, each component is equipped with the correct attribute values, and whether the channel interconnections are adequate. The data validation objectives are specified by means of rules, and the rules collection is usually quite extensive (several hundred), so that manual data validation is a cumbersome, costly, and error-prone task. Moreover, the addition of new rules often required expensive extensions of manually programmed checking software. Data validation investigates only the static semantics of the network of software components. A second V&V step is required to check whether the design will ensure the safety properties required, so that – at least under certain boundary conditions stating that train engine drivers have to respect signals and speed restrictions, as far as not automatically enforced by the underlying technology – trains moving concurrently through the railway network are protected against derailing and collisions.

Route-based interlocking (system type (b)) is less flexible than geographical interlocking, since it fixes all train routes through the railway network a priori, using route tables specifying the sequences of track segments to be allocated for each route. This loss of flexibility is compensated by the advantage that configuration data is considerably simpler. The route table is complemented by interlocking tables specifying the point positions and signal states to be enforced when allocating routes. The interlocking tables fix these positions both for the track elements which are part of the actual route, and the elements which are outside the route, but contribute to its safety by guaranteeing flank protection. Finally, a route conflict table identifies the routes which may never be simultaneously allocated, due to utilisation of common track elements [17]. Route-related interlocking offers simpler means for data validation, since the control software does not need to to be based on communicating software instances related to each track element. Instead, a control algorithm monitors a dynamic plant model (each track element with its free/occupied status, and the locked/unlocked states of points). Route allocation decisions can made by means of these element states and their compatibility with the interlocking table restrictions. Data validation is only concerned with choosing the proper software components (e.g., the correct types of signals and points), and their consistency with the physical network. V&V of the dynamic behaviour now has the objective to verify both the correctness of the control algorithm and the correctness of the interlocking tables. Even in presence of a completely correct algorithm, a safety violation may occur if these tables are not adequately specified; e.g., if a conflict between two routes has not been properly documented in the tables. As a consequence, the data validation activities concerning static semantics of the software components is simpler and less critical than in the case of geographical interlocking systems, but only V&V of the dynamic behaviour can verify the crucial safety properties of the interlocking tables.

1.2 State-of-the-art Formal Methods for Interlocking V&V

The European CENELEC standards applicable for the development of software in railway control systems require the application of formal specification and design models and formalised, justified V&V activities to be performed for software of the highest criticality, as applicable for interlocking systems [9]. The objective of such formalizations is to ensure that potential safety breaches caused by invalid configuration data or erroneous control algorithms can be identified in a systematic way. If formal methods application can also be "mechanised" by means of suitable tools, it contributes to the efficiency of V&V for interlocking system designs in a considerable way. As of today, three methods are applied for formal interlocking V&V: formal verification by theorem proving, by global model checking, or by bounded model checking (BMC). Each of these methods depends on the existence of models describing the static semantics of the interlocking systems, and their dynamic behaviour in combination with trains traversing the railway network.

While – just like theorem proving – global model checking may result in complete correctness proofs of data correctness and safety properties, experience (see for instance [12]) has shown that complex interlocking systems cannot be verified by means of global model checking, since this would lead to state explosions for all but the simplest interlocking systems. In contrast to this, bounded model checking investigates model properties in the vicinity of a given state only, and can therefore be applied to models of considerable size. In this contribution we describe first how BMC is applied to data validation. This is performed by checking the compliance of the data with correctness rules that may be expressed formally by some temporal logic. Next, for the verification of safety properties, BMC can be combined with inductive reasoning, and again, this results in a global proof of the desired safety properties. The bounded model checking techniques to be applied are sufficiently mature today to be applied in an industrial context.

1.3 BMC as Best Practice for Interlocking V&V

The bounded model checking solution to data validation is explained for geographical interlocking systems, since there the requirements for this validation are far more complex than for route-related interlocking. We describe how the software components instantiated according to the given configuration data can be formalised by means of a Kripke Structure whose state space is given by the software component instances, where the transition relation is induced by the communication channels connecting neighbouring objects, and the labelling function specifies the attributes associated with each instance. It is explained how typical pattern of data validation rules can be expressed by means of Linear Temporal Logic (LTL) including existential quantification of specific variable values. A trace of states fulfilling such a formula identifies a witness for a *violation* of the validation rule. Application of LTL model checking allows for easy extendability of the rule base, by simply adding new LTL formulae representing violations of the new rules. No further software extensions are required, as long as a sufficiently powerful bounded model checker for LTL exists. We further describe how the BMC approach can be rightfully applied, because each data validation rule only applies to a finite trace through the Kripke structure (while LTL property checking in general refers to infinite computations). A bounded LTL property checking algorithm is sketched which can be efficiently applied for performing the data validation activities.

In [16] we have described a formal, model-driven method for efficient development and verification of product lines of re-configurable route-related interlocking systems. This method is based on many years of research of which the most recent publications include [17] and [14, 15]. According to this method the development and verification of an interlocking system should be made in a number of steps including the following ones: (1) Specify application-specific parameters in a domain-specific railway language, and (2) from the domain-specific specification, generate a formal, behavioural model of the interlocking system and formal specification of the required safety properties. This generation should be fully automated by tools developed for the purpose. For this setting we describe how BMC may be applied in combination with inductive reasoning, in order to verify global safety properties of the interlocking system software and configuration data generated from these models. This combination of BMC and induction is well-established today in many domains, and it is known to scale up for complex "real-world" applications.

1.4 Related Work

An overview of trends in formal methods applications to railway signalling can be found in [5, 11]. Many other research groups have been using model-checking for the verification of interlocking systems. In [12] a systematic study of applicability bounds of the symbolic model-checker NuSMV and the explicit model checker SPIN showed that these popular model checkers could only verify small railway yards. Several domain-specific techniques to push the applicability bounds for model checking interlocking systems have been suggested. Here we will just mention some of the most recent ones. In [25] Winter pushes the applicability bounds of symbolic model checking (NUSMV) by optimizing the ordering strategies for variables and transitions using domain knowledge about the track layout. Fantechi suggests in [10] to exploit a distributed modelling of geographical interlocking systems and break the verification task into smaller tasks that can be distributed to multiple processors such that they can be verified in parallel. In [20], it is suggested to reduce the state space using abstraction techniques reducing the number of track sections and the number of trains.

For the alternative approach to interlocking V&V based on theorem proving, the B-Method and its variants, such as Event-B, seem to be the formal methods most strongly favoured for railway control applications in Europe. The formal verification of behavioural properties is described, and the methods' applicability on an industrial scale has been established, for example, in [2]. In [6, 18], the application of Event-B to data validation is described. Further verification approaches using theorem proving have been based on the RAISE method, as described in [13].

An introduction into LTL can be found in [7]. The existential quantification operator for LTL, which plays a crucial role in our concept of automated data validation, has been originally introduced in [19]. Its adaptation to finite trace semantics has been performed by the authors. The original semantics and algorithms for verifying LTL formulae against finite trace segments have been devised in [3, 4]. On these finite segments only a subclass of LTL formulae can be verified, this class has been identified in [24]. Fairness properties, for example, which can be expressed in the complete LTL with infinite computations as models, are not part of this class. Our data validation properties, however, as well as the safety properties to be fulfilled by the behavioural interlocking system semantics, are all part of the so-called *Safety LTL* subset which is expressible on finite trace segments.

1.5 Paper Overview

Sections 2 and 3 describe our methods for data validation and for verifying system safety, respectively. In Section 4, the presented methods are discussed.

2 Data Validation

2.1 Kripke Structure Encodings of Static Plant Model

As sketched above, the software controlling geographical interlocking systems consists of instances communicating over channels, each instance representing a physical track element in the plant model. A subset of these channels – called primary channels in the following – reflect the physical interconnection between neighbouring track elements which are part of possible routes, to be dynamically allocated when a request for traversal from some starting point to a destination is given (Fig. 1). Other channels – called secondary channels – connect certain elements s_1 to others s_2, such that s_1 and s_2 are never neighbouring elements on a route, but s_2 may offer flank protection to s_1, when some route including s_1 should be allocated. Since geographical interlocking is based on request and response messages, each channel for sending request messages from some instance s_1 connected to an instance s_2 is associated with a "response channel" from s_2 to s_1. Primary channels are subsequently denoted by variable symbols a, b, c, d, while secondary channels are denoted by e, f, g, h.

All software instances are associated with a unique id. Depending on the track element type they are representing in the plant model, software instances carry an element type t. Depending on the type, a list of further attributes $a_1, \ldots, a_k$ may be defined for each software instance. By using a default value 0 for attributes that are not used for a certain component type, each element can be associated with the same complete list of attributes, where the ones which are

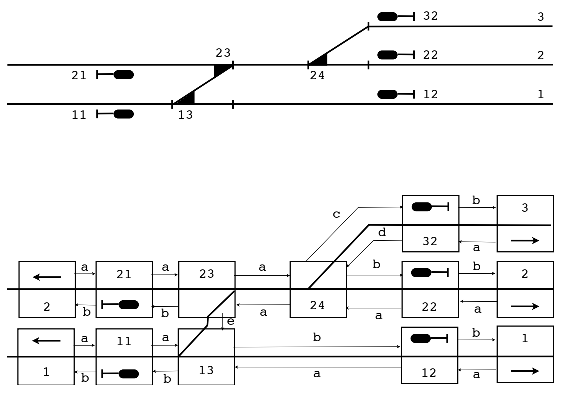

Fig. 1. Physical layout, associated software instances and channel connections.

not applicable are set to 0. Each valuation of a channel variable contains either a default value 0, meaning "no connection on this channel", or the instance identification $id > 0$ of the destination instance of the channel.

We will now formalise the static design of geographical interlocking systems as a Kripke Structure $K = (S, S_0, R, L, AP)$, with state space S, set of initial states $S_0 \subseteq S$, transition relation $R \subseteq S \times S$ and labelling function $L : S \to 2^{AP}$, where AP is a set of atomic propositions and 2^{AP} denotes its power set [7]. To this end, define a set V of variable names as introduced above, $V = \{id, t, a, b, c, d, e, f, g, h, a_1, \ldots, a_k\}$. The state space S consists of one valuation function $s : V \to \mathbf{N}_0$ for each software component. Each function maps the variables to integers identifying the associated software component (id is mapped to its unique id, t to its type, etc.). The set of initial states S_0 is defined to be the set of all states S. This allows us to start data validations at arbitrary track elements. The transition relation R defines each instance s_2 reachable from some instance s_1 via any of the channels $a, \ldots, h$ to be a possible post-state of s_1.

$$R = \{(s_1, s_2) \mid s_1(v) = s_2(id) \wedge v \in \{a, \ldots, h\}\} \tag{1}$$

The set of atomic propositions AP is defined as the collection of all propositions stating equality of some attribute $v \in V$ to one of its possible values, $AP = \{v = \xi \mid v \in V \wedge \xi \in \mathbf{N}_0\}$. The labelling function L maps each state s to the set $L(s)$ of propositions which hold true in s, that is, $\forall s \in S : L(s) = \{v = s(v) \mid v \in V\}$.

Now the violation of any data validation rule may be defined as a LTL formula specifying witnesses of such an unwanted sequence of neighbouring elements. This will be illustrated in the following by a collection of validation examples.

2.2 LTL Syntax

The LTL formulae specifying witnesses for rule violations use symbols from V as free variables. The atomic propositions involved may consist of arithmetic expressions and comparison operators $=, <, >, \leq, \geq, \neq$. The valid LTL formulae are constructed according to the following rules.

- Every atomic proposition is a LTL formula.
- If φ, ψ are LTL formulae, then[4] $\neg\varphi$, $\phi \wedge \psi$, $\phi \vee \psi$, $(\exists b : \varphi)$, $\mathbf{F}\varphi$, $\mathbf{G}\varphi$, $\mathbf{X}\varphi$, $(\varphi \mathbf{U} \psi)$ are LTL formulae. It is assumed that bound variable symbol b is not contained in V.

2.3 Bounded Trace Semantics for LTL

The semantic rules for evaluating LTL formulae on finite trace segments $s_i \dots s_k$ are specified using notation $\langle\varphi\rangle_i^k$. The recursive rules for evaluating the truth value of $\langle\varphi\rangle_i^k$ can be directly transformed into an algorithm unrolling $\langle\varphi\rangle_i^k$ into a proposition no longer involving any temporal operators ($\mathbf{F}, \mathbf{G}, \mathbf{X}, \mathbf{U}$), but referring to variable valuations in states $s_i, s_{i+1} \dots, s_k$ and Boolean operators $\neg, \wedge, \vee$ only. Observe that we omit the semantics for $\mathbf{G}$ here, because our witnesses violating data rules are always represented by finite trace segments $s_i.s_{i+1} \dots s_k$ without loops, whereas $\mathbf{G}\varphi$ only holds true if the trace segment has a lasso shape, where previous state on the segment is re-visited, thereby creating a cycle. The BMC semantics of $\mathbf{G}$ is discussed in detail in [3, 4].

The remaining transformation rules applicable for data validation are (symbols p denote atomic propositions)

$$\begin{aligned}
\langle\varphi\rangle_i^k = \text{false} \quad & \text{if} \quad i > k && (2)\\
\langle p\rangle_i^k \quad & \text{iff} \quad p[s_i(v)/v \mid v \in var(p)] && (3)\\
\langle\neg\varphi\rangle_i^k \quad & \text{iff} \quad \neg\ \langle\varphi\rangle_i^k && (4)\\
\langle\varphi \wedge \psi\rangle_i^k \quad & \text{iff} \quad \langle\varphi\rangle_i^k \ \wedge\ \langle\psi\rangle_i^k && (5)\\
\langle\varphi \vee \psi\rangle_i^k \quad & \text{iff} \quad \langle\varphi\rangle_i^k \ \vee\ \langle\psi\rangle_i^k && (6)\\
\langle(\exists b : \varphi)\rangle_i^k \quad & \text{iff} \quad \langle\varphi\rangle_i^k \wedge \bigwedge_{j=i}^{k-1} (s_j(b) = s_{j+1}(b)) && (7)\\
\langle\varphi \mathbf{U} \psi\rangle_i^k \quad & \text{iff} \quad \langle\psi\rangle_i^k \vee (\langle\varphi\rangle_i^k \wedge \langle\varphi[b'/b \mid b \in \text{bound}(\varphi)]\mathbf{U}\psi\rangle_{i+1}^k) && (8)\\
\langle\mathbf{X}\varphi\rangle_i^k \quad & \text{iff} \quad \langle\varphi\rangle_{i+1}^k && (9)\\
\langle\mathbf{F}\varphi\rangle_i^k \quad & \text{iff} \quad \bigvee_{j=i}^{k} \langle\varphi\rangle_j^k && (10)
\end{aligned}$$

In this specification of semantic transformations, Equation (2) describes a termination condition: if $i > k$, the formula is evaluated on an empty trace segment,

[4] We do not need to consider the weak until operator $\mathbf{W}$, or the release operator $\mathbf{R}$.

and this is false by definition. Equation (3) associates truth value true with an atomic proposition if it evaluates to true after having replaced all variables v by their actual value $s_i(v)$ in the initial state s_i of the trace segment under consideration. In Equation (7) it is shown how a formula using existential quantification with bound variable b is transformed into a proposition. Note that b occurs free in right-hand side formula, and extends domain of $s_j, s_{j+1}, \ldots, s_k$ by b. The conjunction over terms $s_j(b) = s_{j+1}(b)$ specifies that, once the value of b has been fixed for some state s_j, the same value has to be used in all states along the trace segment. The recursive definition of the until operator in Equation (8) requires to use fresh bound variable symbols in each transformation step of the formula. This is illustrated in Example 1.

Example 1. Consider the BMC evaluation of property $(\exists b : y = b \wedge \mathbf{X}(y = b+1))\mathbf{U}(x > 10)$ on trace segment $s_0.s_1.s_2$, that is $\langle(\exists b : y = b \wedge \mathbf{X}(y = b+1))\mathbf{U}(x > 10)\rangle_0^2$. Applying the rules above, this is unrolled to

$$\begin{array}{l}
\langle(\exists b : y = b \wedge \mathbf{X}(y = b+1))\mathbf{U}(x > 10)\rangle_0^2 \equiv \\
\langle(x > 10)\rangle_0^2 \vee \\
(\langle(\exists b : y = b \wedge \mathbf{X}(y = b+1))\rangle_0^2 \wedge \\
\langle(\exists b' : y = b' \wedge \mathbf{X}(y = b'+1))\mathbf{U}(x > 10)\rangle_1^2) \equiv \\
(s_0(x) > 10) \vee \\
(\langle(y = b) \wedge \mathbf{X}(y = b+1))\rangle_0^2 \wedge \\
\bigwedge_{j=0}^{1} (s_j(b) = s_{j+1}(b)) \wedge \\
\langle(\exists b' : y = b' \wedge \mathbf{X}(y = b'+1))\mathbf{U}(x > 10)\rangle_1^2) \equiv \\
(s_0(x) > 10) \vee \\
((s_0(y) = s_0(b)) \wedge (s_1(y) = s_1(b) + 1)) \wedge \\
\bigwedge_{j=0}^{1} (s_j(b) = s_{j+1}(b)) \wedge \\
((s_1(x) > 10) \vee \\
(s_1(y) = s_1(b') \wedge s_2(y) = s_2(b') + 1) \wedge (s_1(b') = s_2(b')) \wedge \\
\langle(\exists b'' : y = b'' \wedge \mathbf{X}(y = b''+1))\mathbf{U}(x > 10)\rangle_2^2) \equiv \\
(s_0(x) > 10) \vee \\
((s_0(y) = s_0(b)) \wedge (s_1(y) = s_1(b) + 1)) \wedge \\
\bigwedge_{j=0}^{1} (s_j(b) = s_{j+1}(b)) \wedge \\
((s_1(x) > 10) \vee \\
((s_1(y) = s_1(b')) \wedge (s_2(y) = s_2(b') + 1) \wedge (s_1(b') = s_2(b')) \wedge \\
((s_2(x) > 10) \vee ((s_2(y) = s_2(b'')) \wedge \text{false})))
\end{array}$$

2.4 Data Validation by Bounded Model Checking

Bounded model checking in general is concerned with the solution of so-called *bounded model checking instances*, that is , constraints of the form

$$J(s_0) \wedge \bigwedge_{i=1}^{k} R(s_{i-1}, s_i) \wedge G(s_0, \ldots, s_k) \qquad (11)$$

For solving these constraints, SMT solvers are used. When applied in the context of BMC, $J(s_0)$ is a proposition specifying the starting state from where a witness should be found within a bounded number of steps k. For the purpose of data validation, $J(s_0)$ admits any track element – respectively, its software instance – s_0 as a starting point, since all validation rules have to be applied to track segments starting at any element s_0 in the interlocking system area. Therefore $J(s_0)$ can be expressed by

$$J(s_0) \equiv \bigvee_{s \in S} \left(\bigwedge_{v \in V} s_0(v) = s(v) \right) \tag{12}$$

This initial condition states that any initial valuation[5] of variables $v \in V$ must coincide with any of the software instances $s \in S$ representing track elements.

Proposition $G(s_0, \dots, s_k)$ specifies the unwanted property of the trace segment of length k to be found, that is, a sequence of track element-related software instances $s_0, \dots, s_k$ violating some validation rule $\neg G$. As described in Section 2.5, such an unwanted property G to be uncovered will be specified in LTL. Therefore bounded LTL model checkers parse the original LTL formulae and apply the transformation rules specified in formulae (2 — 10), in order to produce an equivalent propositional formula G, as illustrated in Example 1 above.

The conjuncts $R(s_{i-1}, s_i)$ enforce that only solutions of $G(s_0, \dots, s_k)$ are considered that correspond to trace segments whose elements are related by the transition relation. For our purpose of data validation this means, that each pair (s_{i-1}, s_i) of a solution trace $s_0, \dots, s_k$ is connected by some primary or secondary channel $a, \dots, h$. Therefore the conjunction is expressed by (see Equation (13))

$$\bigwedge_{i=1}^{k} R(s_{i-1}, s_i) \equiv \bigwedge_{i=1}^{k} \left(\bigvee_{c \in \{a, \dots, h\}} s_{i-1}(c) = s_i(id) \right) \tag{13}$$

Summarising, data validation for geographical interlocking requires to solve BMC instances of the form

$$\bigvee_{s \in S} \left(\bigwedge_{v \in V} s_0(v) = s(v) \right) \wedge \bigwedge_{i=1}^{k} \left(\bigvee_{c \in \{a, \dots, h\}} s_{i-1}(c) = s_i(id) \right) \wedge \mathrm{Trans}(\phi) \tag{14}$$

where ϕ specifies a violation of some validation rule in LTL, and $\mathrm{Trans}(\phi)$ denotes the transformation of LTL formulae into propositional formulae according to the rules specified in formulae (2 — 10).

If the bounded model checker is able to calculate a witness, that is, a solution of Formula (14) within k steps, an error has been found, so the bounded model

[5] Recall from Section 2.1 that V contains the variable symbols for element identification (id), element type (t), channels connecting to neighbouring elements ($a, \dots, h$), and additional type-specific attributes ($a_1, \dots, a_k$). For some $v \in V$, notation $s_0(v)$ denotes the value of variable v to be determined by the SMT solver for the initial state s_0.

checker is a valuable tool for *bug finding*. If, however, no witness can be found within k steps, it remains to be determined whether some witness might be found if k is increased. This question has been answered for the general case of arbitrary Kripke Structures and LTL formulae in [4, 3]. If k corresponds to the so-called *diameter* of the Kripke Structure under consideration, and no solution could be found for this k, bounded model checking provides a global *proof* of non-existence for such a witness. While the diameter is often too large to be applied for BMC in practice, it is of feasible size in the case of data validation, because it roughly corresponds to the maximal length of track segments from some element to the boundary of the interlocking system area.

2.5 Applications

We will now describe several examples illustrating the expressiveness of LTL for the verification of data validation rules.

Example 2. The simplest validation rules state that instances representing elements of a certain type $t = \tau$ must have certain attributes with values in a specific range, such as $a_i \in [x_0, x_1]$. A violation of this property is readily expressed by LTL formula $\mathbf{F}(t = \tau \wedge (a_i < x_0 \vee x_1 < a_i))$.

Example 3. The following rule checks the correctness of channel connections. *"If there exists a channel from s_1 to s_2, there must exist a channel in the reversed direction"*. A violation of this rule can be specified in natural language as *"There exists an instance s_1 which is not the auxiliary initial state, so that s_1 is connected to some instance s_2, but all channels emanating from s_2 lead to instances different from s_1"*. In LTL this is expressed as

$$\mathbf{F}(\exists i : id = i \wedge id > 0 \wedge \mathbf{X}(a \neq i \wedge b \neq i \wedge \ \ldots \ \wedge h \neq i))$$

A witness for such a rule violation reaches an element s with positive id (so it does not equal s_0) and at least one of its reachable neighbours (which, by definition of R, are only reachable if there is a connecting channel from s to this neighbour) has no channel with destination s.

Example 4. The following rule pattern frequently occurs when checking configuration data with respect to software component instances representing illegal sequences of track elements along a route. *"Following a track element of type τ_1 along its a-channel, and only regarding primary channel connections, an element of type τ_2 must occur, before an element of type τ_3 is found"*. The violation of this rule is specified by *"Find a track element of type τ_1 and follow it along its a-channel, so that only elements of type $t \neq \tau_2$ may be found along its primary channel directions, until an element of type τ_3 is encountered"*.

$$\begin{aligned}
&\mathbf{F}(t = \tau_1 \wedge \exists x : (a = x \wedge \mathbf{X}(id = x \wedge ((t \neq \tau_2 \wedge \\
&\qquad \exists y : ((a = y \vee b = y \vee c = y \vee d = y) \wedge \mathbf{X}(id = y))) \\
&\quad \mathbf{U}(t = \tau_3)))))
\end{aligned}$$

2.6 Tool Support

In principle, data validation by means of LTL can be performed with any LTL model checker that is able to encode the Kripke Structure representing the static semantics of the geographical interlocking system as described in Section 2.1. A reference implementation has been performed by the authors using the model-based testing and bounded model checking tool RT-Tester, described in more detail in [22]. RT-Tester performs automated test data generation or calculation of BMC witnesses by solving constraints of the form specified in Equation (11), with the help of the SONOLAR SMT solver described in [23].

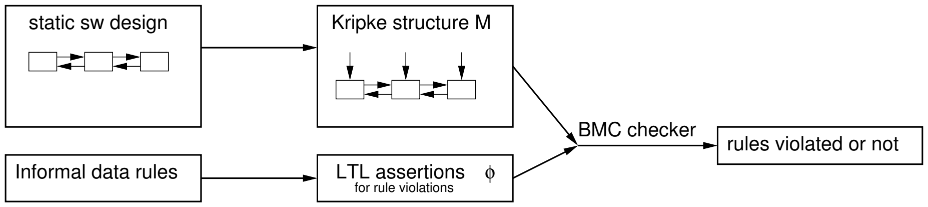

Fig. 2. Tool support for data validation work flow.

Figure 2 shows the data validation work flow and indicates the interaction between tool components.

- The static software design of the geographical interlocking system is represented by encodings $s \in S$ of software instances corresponding to track elements. In the reference implementation described here, this is encoded in XML.
- A parser front-end of RT-Tester developed for this XML encoding reads the design and transforms it into the internal model representation of the tool. This is an abstract syntax tree data structure that allows for syntactic representation of a wide variety of formalisms, such as UML/SysML, Matlab/Simulink, process algebras, and the proprietary interlocking system format described here.
- RT-Tester allows for utilisation of different transition relation generators, associated with the semantics of each supported modelling formalism. One of these generators creates the initial state condition and transition relation for the Kripke Structure introduced in Section 2.1, according to equations (12) and (13).
- The data validation rules are transformed by experts into LTL formulae ϕ representing rule violation.
- The LTL parser of RT-Tester reads the formulae, and they are transformed by the tool into propositional formulae.
- The diameter k of the track network is determined.

– The SMT solver tries to find a solution for the BMC instance shown in Equation (14). If a solution can be found, a violation of rule $\neg\phi$ has been uncovered. If no solution can be found, it has been *proven* that this rule is nowhere violated, because k is the diameter of the network.

3 Verification of System Safety

This section describes our method for formally verifying safety of an interlocking system.

3.1 Formalization of the Verification Task

According to our method, the input of this verification step should consist of: (1) a formal, state-based, behavioural model $\mathcal{M}$ of the interlocking system and its physical environment and (2) safety conditions Φ expressed as a conjunction of propositions over the state variables in $\mathcal{M}$. The verification goal is then to verify that the safety conditions Φ hold for any reachable state in $\mathcal{M}$.

As will be explained below, a model checker tool should be used for automated verification of such a goal. Therefore, the model $\mathcal{M}$ and the formula Φ should be expressed in the input language of the chosen model checker.

3.2 Verification Strategy

There is an established approach to apply bounded model checking in combination with inductive reasoning, in order to prove global system properties; this approach is called *k-induction.* For proving that safety condition Φ holds for all reachable states of $\mathcal{M}$, this method proceeds as follows.

1. First prove that $\Phi \wedge \Psi$ holds for the $k > 0$ first execution cycles after initialisation, i.e. $\Phi \wedge \Psi$ holds for $k > 0$ successive[6] states $\sigma_0, \ldots, \sigma_{k-1}$ of which σ_0 is the initial state of $\mathcal{M}$.
2. Next prove the following for an arbitrary execution sequence of $k+1$ successive states $\sigma_t, \ldots, \sigma_{t+k}$ of which the first σ_t is an arbitrary state (reachable or not from the initial state σ_0): if $\Phi \wedge \Psi$ holds in the k first states $\sigma_t, \ldots, \sigma_{t+k-1}$, then $\Phi \wedge \Psi$ must also hold for the $k + 1^{st}$ state σ_{t+k}.

Here Ψ is an auxiliary property that holds for reachable states. (Note that Ψ is simultaneously proven by the given induction principle.) The proofs of the base case and the induction step should be performed by a bounded model checker tool. An example of such a tool is described in [8]. This tool treats the two proof obligations by exploring corresponding propositional satisfiable problems and solving these by a SAT solver. Note that the induction steps argue over an execution sequence of k+1 states of which the first state, σ_t, may be unreachable,

[6] Two states σ_i and σ_{i+1} are successive, if there is a transition from σ_i to σ_{i+1} according to $\mathcal{M}$.

although it would have been sufficient only to consider sequences for which σ_t is reachable. For sequences starting at an unreachable state, the induction step may fail and the property checker produces a *false negative*. To avoid this, the desired property Φ is strengthened with auxiliary property Ψ that is false for those unreachable states, σ_t, for which the induction step would otherwise fail.

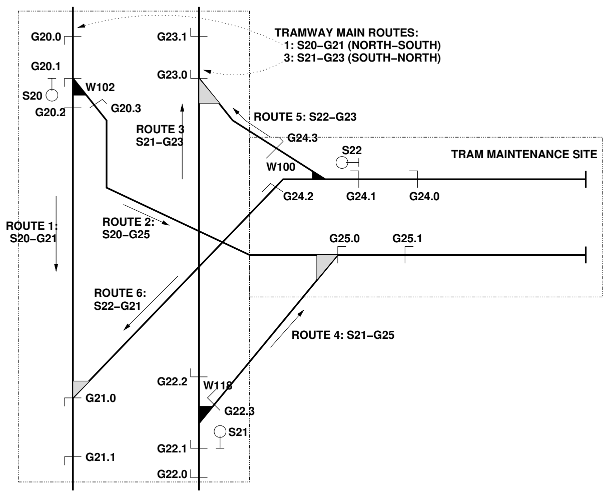

Fig. 3. A tramway network.

3.3 Case Study

A reference publication for this verification technique has been published in [8]. It describes a real-world route-related tramway control system. For the network in Figure 3, the model of the tramway control system was verified to be safe, using k-induction. The safety conditions Φ was a conjunction of 15 conditions ensuring no collisions and no derailments of trams, and the auxiliary condition Ψ was a conjunction of conditions expressing state relations needed as assumptions in the induction step, in order to rule out unreachable states that would have given rise to false negatives otherwise. It turned out that a value of $k = 3$ sufficed to carry out the induction. The proofs of the base case and the induction step were performed by a bounded model checker, which used 392 seconds to perform the proofs. For more details about the case study, see e.g. [8, 17].

4 Conclusion

In this paper the application of bounded model checking for verification and validation of interlocking systems has been described. In contrast to global model checking which usually leads to state space explosions when applied to complex interlocking systems, bounded model checking allows for application in large and complex interlocking system layouts. It has been shown how the technique can be applied on two levels. First, in the form of LTL property checking, for the purpose of configuration data validation. Next, in combination with inductive reasoning, for the purpose of verifying safety properties for the dynamic behaviour of trains traversing the track network. Tool applications and measurements show that both application scenarios scale up for application in an industrial context.

Acknowledgments The first author has been supported by the RobustRailS project funded by the Danish Council for Strategic Research. The second and third authors have been supported by the openETCS project funded by the European ITEA2 organisation.

References

1. Guide to the software engineering body of knowledge. Technical report, IEEE Computer Society, 2004. Available under `http://www.computer.org/portal/web/swebok/htmlformat`.
2. Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. Météor: A successful application of b in a large project. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 369–387, Berlin Heidelberg, 1999. Springer.
3. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, UK, 1999. Springer-Verlag.
4. Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5):1–64, 2006.
5. D. Bjørner. New Results and Current Trends in Formal Techniques for the Development of Software for Transportation Systems. In *Proceedings of the Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'2003), Budapest/Hungary.* L'Harmattan Hongrie, May 15-16 2003.
6. Mathieu Clabaut, Christophe Metayer, and Eric Morand. 4B-2 formal data validation – formal techniques applied to verification of data properties. In *Embedded Real Time Software and Systems ERTS*, 2010.
7. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.
8. R. Drechsler and D. Große. System level validation using formal techniques. *IEE Proc.-Comput. Digit. Tech.*, 152(3):393–406, May 2005.
9. European Committee for Electrotechnical Standardization. *EN 50128:2011 – Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems.* CENELEC, Brussels, 2011.
10. Alessandro Fantechi. Distributing the Challenge of Model Checking Interlocking Control Tables. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*, volume 7610 of *Lecture Notes in Computer Science*, pages 276–289. Springer Berlin Heidelberg, 2012.

11. Alessandro Fantechi, Wan Fokkink, and Angelo Morzenti. Some Trends in Formal Methods Applications to Railway Signaling. In *Formal Methods for Industrial Critical Systems*, pages 61–84. John Wiley & Sons, Inc., 2012.
12. Alessio Ferrari, Gianluca Magnani, Daniele Grasso, and Alessandro Fantechi. Model Checking Interlocking Control Tables. In E. Schnieder and G. Tarnai, editors, *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2010), Braunschweig, Germany*. Springer, 2011.
13. A. E. Haxthausen and J. Peleska. Formal Development and Verification of a Distributed Railway Control System. *IEEE Transaction on Software Engineering*, 26(8):687–701, 2000.
14. Anne E. Haxthausen. Towards a Framework for Modelling and Verification of Relay Interlocking Systems. In *16th Monterey Workshop: Modelling, Development and Verification of Adaptive Systems: the Grand Challenge for Robust Software*, number 6662 in Lecture Notes in Computer Science, pages 176–192. Springer, 2011.
15. Anne E. Haxthausen. Automated Generation of Safety Requirements from Railway Interlocking Tables. In *5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'2012), Part II*, number 7610 in Lecture Notes in Computer Science, pages 261–275. Springer, 2012.
16. Anne E. Haxthausen and J. Peleska. Efficient Development and Verification of Safe Railway Control Software. In *Railways: Types, Design and Safety Issues*, pages 127–148. Nova Science Publishers, Inc., 2013.
17. Anne E. Haxthausen, Jan Peleska, and Sebastian Kinder. A Formal Approach for the Construction and Verification of Railway Control Systems. *Formal Aspects of Computing*, 23(2):191–219, 2011.
18. Thierry Lecomte, Lilian Burdy, and Michael Leuschel. Formally checking large data sets in the railways. *CoRR*, abs/1210.6815, 2012.
19. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.
20. Faron Moller, Hoang Nga Nguyen, Markus Roggenbach, Steve Schneider, and Helen Treharne. Defining and Model Checking Abstractions of Complex Railway Models using CSP||B. In Armin Biere, Amir Nahir, and Tanja Vos, editors, *Hardware and Software: Verification and Testing*, volume 7857 of *Lecture Notes in Computer Science*, pages 193–208. Springer Berlin Heidelberg, 2013.
21. Jörn Pachl. *Railway Operation and Control.* VTD Rail Publishing, January 2002.
22. Jan Peleska. Industrial-strength model-based testing - state of the art and current challenges. In Alexander K. Petrenko and Holger Schlingloff, editors, Proceedings Eighth Workshop on *Model-Based Testing,* Rome, Italy, 17th March 2013, volume 111 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–28. Open Publishing Association, 2013.
23. Jan Peleska, Elena Vorobev, and Florian Lapschies. Automated test case generation with SMT-solving and abstract interpretation. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *Nasa Formal Methods, Third International Symposium, NFM 2011*, volume 6617 of *LNCS*, pages 298–312, Pasadena, CA, USA, April 2011. Springer.
24. A. P. Sistla. Liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–512, 1994.
25. Kirsten Winter. Optimising ordering strategies for symbolic model checking of railway interlockings. In *5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'2012), Part II*, number 7610 in Lecture Notes in Computer Science, pages 246–260. Springer, 2012.