Efficient Data Validation for Geographical Interlocking Systems

Peleska, Jan; Krafczyk, Niklas; Haxthausen, Anne Elisabeth; Pinger, Ralf

Published in:
Proceedings of 2019 International Conference on Reliability, Safety, and Security of Railway Systems

Link to article, DOI:
10.1007/978-3-030-18744-6_9

Publication date:
2019

Document Version
Peer reviewed version

Efficient Data Validation for Geographical Interlocking Systems

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

[1] University of Bremen, Department of Mathematics and Computer Science, Germany
{peleska,niklas}@uni-bremen.de
[2] DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
aeha@dtu.dk
[3] Siemens Mobility GmbH, Braunschweig, Germany
ralf.pinger@siemens.com

Abstract. In this paper, an efficient approach to data validation of geographical interlocking systems (IXLs) is presented. It is explained how configuration rules for IXLs can be specified by temporal logic formulas interpreted on Kripke structure representations of the IXL configuration. Violations of configuration rules can be specified using formulas from a well-defined subset of LTL. By decomposing the complete configuration model into sub-models corresponding to routes through the model, the LTL model checking problem can be transformed into a CTL checking problem for which highly efficient algorithms exist. Specialised rule violation queries that are hard to express in LTL can be simplified and checked faster by performing sub-model transformations adding auxiliary variables to the states of the underlying Kripke structures. Further performance enhancements are achieved by checking each sub-model concurrently. The approach presented here has been implemented in a model checking tool which is applied by Siemens for data validation of geographical IXLs.

Keywords: Data validation, Interlocking systems, LTL, CTL, Model checking

1 Introduction

Background Railway interlocking systems (IXLs) are designed according to different paradigms [13, Chapter 4]. Two of the most widely used are (a) *route-based interlocking systems* and (b) *geographical interlocking systems*. The former are based on predefined routes through the rail network and use interlocking tables specifying safety conflicts between different routes and the point positions and signal states to be enforced before a route may be entered by a train. For design type (b), 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 IXL under consideration. In the original technology, electrical relay-based circuits were applied, whose elements

and interconnections were 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.

IXL design induces a distinguished verification and validation (V&V) step which is called *data validation*. For route-based IXLs, its main objective is to ensure completeness and correctness of interlocking tables. For geographical IXLs, the 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 rule collection is usually quite extensive (several hundreds), so that manual data validation would be a cumbersome, costly, and error-prone task. Also, manually programmed checking software is not a satisfactory solution, since the addition of new rules would require frequent extensions of the code. These extensions are costly, since data validation tools need to be validated according to tool class T2, as specified in the standard [5].

Previous Work This paper is a follow-up contribution to [9], where a solution to the data validation problem for geographical IXLs by means of bounded model checking (BMC) had been presented.[4] During practical evaluation of the results described there, it turned out that the BMC approach was highly effective as a bug-finder: if violations of configuration rules were present, these were uncovered effectively and within acceptable running time. The configuration experts from Siemens, however, criticised that the tool would not prove the *absence* of configuration errors. Typical for BMC algorithms, the running time of the checks sometimes increased exponentially with the search depth, so that an exploration

[4] The text of the previous paragraph describing the general problem and the more detailed description in Section 2 have been reproduced here in slightly modified form from [9], in order to make the present paper self-contained.

of the model up to its *recurrence diameter*[5] would have resulted in unacceptable running time and storage consumption.

Main Contributions As a consequence of the experiences gained with the application of BMC technology described in [9], an alternative approach has been elaborated and implemented in a new data validation tool, the *DVL-Checker* (Data Validation Language Checker). The new approach is described in the present paper, and it is based on the following key insights which, to our best knowledge, have not been explored before for the purpose of IXL data validation.

1. Exploiting known results about the temporal logic LTL, it is shown that violations of safety-properties can be represented by a syntactic subset of LTL which is denoted as *data validation language (DVL)*. This ensures that violations of IXL configuration rules can be specified using this subset.
2. Exploiting known results about LTL and CTL, we show how LTL formulae ϕ representing safety violations (so-called *DVL-queries*) can be translated to CTL formulae $\Phi(\phi)$, such that CTL model checking of $\Phi(\phi)$ is an *over-approximation* for LTL model checking of ϕ in the sense of abstract interpretation. This means that the absence of witnesses[6] for CTL formula $\Phi(\phi)$ implies the absence of solutions for LTL formula ϕ, which proves that no rule violation specified by ϕ is present.
3. For CTL, highly efficient and well-explored global model checking algorithms can be applied. These have complexity $O(|f| \cdot (|S| + |R|))$, where $|f|$ is the number of sub-formulae in CTL formula f, $|S|$ is the size of the state space, and $|R|$ is the size of the transition relation. Moreover, the application of CTL model checking is generally more efficient than that of LTL model checking, since the latter represents an NP-hard problem [6, Section 4.2] which is PSPACE-complete [16].
4. A decomposition of the complete IXL configuration into sub-models corresponding to directed routes through the railway network allows for (1) significant reduction of false alarms that might result from the fact that CTL checking for witnesses of $\Phi(\phi)$ is an over-approximation of LTL checking for ϕ[7], and (2) significant speed-up of the checking process by processing sub-models concurrently.

Related Work Data validation for railway interlocking systems is a well-established V&V task in railway technology. At the same time, it is a very active research field, since the complexity of today's IXL configurations requires a high degree of automation for checking their correctness. There seems to be

[5] The recurrence diameter denotes the number of steps to be performed by a BMC algorithm to achieve exhaustive model exploration [3].

[6] A *witness* is a sequence of states fulfilling a temporal logic formula.

[7] This reduction of false alarms is achieved because the sub-models corresponding to directed routes do not contain as many branches as the full network, and it is well known that on linear paths, LTL and corresponding CTL formulas are equivalent.

an agreement among the research communities that hard-coded data validation programs are inefficient, due to the large number of rules to be checked and the frequent adaptations and extensions of rules that are necessary to take into account the requirements of different IXLs. These observations are confirmed by numerous publications on IXL data validation, such as [1,9,8,7,10].

It is interesting to point out that some V&V approaches for IXLs do not explicitly distinguish between data validation and the verification of dynamic IXL behaviour; this is the case, for example, in [4,10]. We agree, however, with [7], where it is emphasised that data validation should be a separate activity in the IXL V&V process. This assessment is motivated by the analogy with software verification, where the correctness of static semantics – this corresponds to the IXL configuration data – is verified before the correctness of dynamic program behaviour – this corresponds to the dynamic IXL behaviour – is analysed.

As observed in [2], data validation approaches based on the B tool family seem to be the most widely used both in industry and academia in Europe, we name [1,8,7,10] as noteworthy examples for this fact.

The methodology and tool support described in the present paper differs significantly from the B methodology: whereas the methods based on the B family require specifications in first-order logic and perform verification by theorem proving or constraint (model) checking, our approach is based on temporal logic and CTL model checking. Moreover, our methodology strictly specialises in geographical interlocking systems, while – in principle – the B-methods can be applied to any type of IXL technology. Our more restricted approach, however, comes with the advantage that rule specifications are simpler to construct than in B, since temporal logic formulae do not require quantification over variables. Moreover, the sub-model construction technique used in our methodology ensures that the proper verification by CTL model checking is always fully automatic and fast, whereas the B-approaches may require interactive user support during theorem proving [8].

Overview In Section 2, the data validation approach to geographical IXLs is explained from an engineering perspective. The mathematical foundations required to enable automated complete detection of IXL configuration rule violations are elaborated in Section 3. This is done without any reference to the intended application. The latter is described in Section 4, where the application of the mathematical theory to IXL data validation is presented in detail. In Section 5, a conclusion is presented.

A more detailed technical report containing all formal definitions, theorems, proofs, and algorithms on which the DVL-Checker is based can be found in [14].

2 Data Validation for Geographic Interlocking Systems

As indicated above, the software controlling geographical interlocking systems consists of objects communicating over channels, each instance representing a physical track element or a related hardware interface. 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 not necessarily 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 $e, f, g, \ldots$. Only points and diamond crossings use c-channels, and d-channels are used by diamond crossings only.

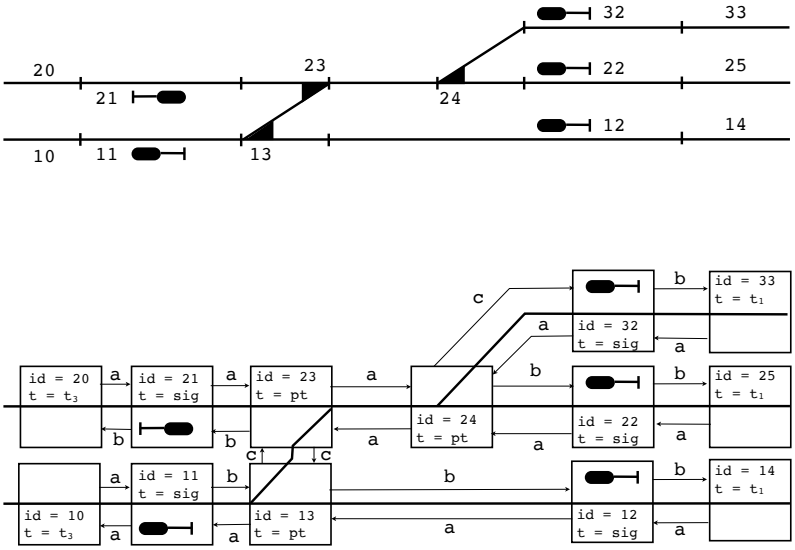

Fig. 1. Physical layout, associated software instances and channel connections.

For signals, the driving direction they apply to is along channel a. For points, the straight track (point position "+") is always represented by the channel connections from a to b and vice versa, and the diverging track (point position "−") always from a to c and vice versa. The stems of a point are denoted by *A,B,C-stems* according to the channels associated with the stem. The entry into/exit from the track network controlled by the interlocking systems is always

marked by *border elements* of a special type. In Fig. 1, these types are denoted by the fictitious identifiers t_1 and t_3. Some track sections may be crossed in both directions, so a border element may serve both as entry and exit element. This is discussed in more detail in the context of sub-model creation in Section 4.

All software instances are associated with a unique *id* and a type t corresponding to the track element type they are representing. Depending on the type, a list of further attributes $a_1, \ldots, a_k$ may be defined for each software instance. By using default value 0 for attributes that are not applicable to a certain component type, each element can be associated with the same complete list of attributes. 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.

Data validation rules state conditions about admissible sequences of element types and about admissible parameters.

Example 1. A typical pattern of data validation rules checks the existence of expected follow-up elements for an element of a given type.

Rule 1. From channel a of an element of type *sig* pointing in downstream direction, an element of the same type with its b-channel pointing upstream is found, before a border element of type t_1 or t_3 is reached.

Every rule can be transformed into a *rule violation condition*. For Rule 1, the violation would be specified as

Violation of Rule 1. From channel a of an element of type *sig* pointing in downstream direction, no element of the same type with its b-channel pointing upstream is found, before a border element of type t_1 or t_3 is reached.

The configuration in Fig. 1 violates Rule 1, because, for example, the path segment $\pi_1 = s_{21}.s_{23}.s_{24}.s_{22}.s_{25}$ contains the follow-up element s_{22}, but this points with its a-channel towards signal 21. Practically, this means that the signal with id 22 does not point into the expected driving direction, so the expected route exit signal along π_1 is missing. An example of a path segment which is consistent with this rule is $\pi_2 = s_{32}.s_{24}.s_{23}.s_{13}.s_{11}.s_{10}$. □

Example 2. Another typical pattern of data validation rules refers to the element types that are required or admissible in certain segments of a route marked by elements of specific type.

Rule 2. Between channel a of an element of type *sig* and channel b of the associated downstream element of the same type *sig*, there must be at least one element of type t_3.

The corresponding rule violation can be specified as

Violation of Rule 2. Between channel a of an element of type *sig* and channel b of the associated downstream element of type *sig*, there does not exist any element of type t_3.

The configuration in Fig. 1 violates this rule, because the path segments connecting the signals of type *sig* do not contain any element of type t_3. □

3 Logical Foundations

In this section, the logical foundations of the model checking method for data validation are summarised. The underlying theory is described without references to their practical application in the IXL context; the latter is explained in Section 4. A comprehensive description of these foundations can be found in [14].

Kripke Structures A *State Transition System* is a triple $TS = (S, S_0, R)$, where S is the set of *states*, $S_0 \subseteq S$ is the set of *initial states*, $R \subseteq S \times S$ is the *transition relation*. The intuitive interpretation of R is that a state change from $s_1 \in S$ to $s_2 \in S$ is possible in TS if and only if $(s_1, s_2) \in R$. A *Kripke Structure* $K = (S, S_0, R, L, AP)$ is a state transition system (S, S_0, R) augmented with a set AP of *atomic propositions* and a *labelling function* $L : S \to 2^{AP}$ mapping each state s of K to the set of atomic propositions valid in s. Furthermore, it is required that the transition relation R is *total* in the sense that $\forall\, s \in S : \exists\, s' \in S : (s, s') \in R$. It is assumed that AP always contains the truth values `false`, `true`.

A *computation* of a state transition system (or a Kripke structure) is an infinite sequence $\pi = s_0.s_1.s_2 \cdots \in S^\omega$ of states $s_i \in S$, such that the start state is an initial state, that is, $s_0 \in S_0$, and each pair of consecutive states is linked by the transition relation, that is, $\forall\, i > 0 : (s_{i-1}, s_i) \in R$. The terms *path* or *execution* are used synonymously for computations. In the context of this paper, state spaces S consist of *valuation functions* $s : V \to D$ mapping variable names from V to their actual values in D. For the context of this paper, it suffices to consider $D = \text{int}$, because all configuration parameters used for the interlocking systems under consideration may be encoded as integers. For the Boolean values `true`, `false`, the integer values $1, 0$ are used, respectively.

First Order Formulae and Their Valuation Given a Kripke Structure K with variable valuation functions $s : V \to \text{int}$ as states, expressions are evaluated in the usual way by replacing free variables v with their actual value $s(v)$ in this state. We write $s \models f$ for an unquantified first-order expression, if and only if f becomes true when replacing all symbols v by $s(v)$. Atomic propositions are constructed by composing variables, constants, or arithmetic expressions using comparison operators. An *(unquantified) first-order formula* f over V is a logical formula with atomic propositions over V, composed by logical operators $\neg, \wedge, \vee$.

Linear Temporal Logic LTL Given a Kripke structure with state valuations over variables from V, we use unquantified first-order LTL with the following syntax.

- Every unquantified first-order formula over V as specified above is an unquantified first-order LTL formula.
- If f, g are unquantified first-order LTL formulae, then $\neg f$, $f \wedge g$, $f \vee g$, $\mathbf{X}f$ (*Next*), $\mathbf{G}f$ (*Globally*), $\mathbf{F}f$ (*Finally*), $f\mathbf{U}g$ (*Until*), and $f\mathbf{W}g$ (*Weak Until*) are also unquantified first-order LTL formulae.

Operators **X**, **G**, **F**, **U**, and **W** are called *path operators*. The models of LTL formulae are infinite paths $\pi = s_0.s_1.s_2.\cdots \in S^\omega$; we write $\pi \models_{\text{LTL}} f$ if formula f holds on path π according to the semantic rules specified in Table 1.[8] We use notation $\pi^i = s_i.s_{i+1}.s_{i+2}\ldots$ to denote the path segment of π starting at element $\pi(i)$. A Kripke structure K fulfils LTL formula f ($K \models_{\text{LTL}} f$) if and only if every computation of K is a model of f.

Table 1. Semantics of LTL formulae.

$\pi^i \models_{\text{LTL}}$ `true` for all $i \geqslant 0$
$\pi^i \not\models_{\text{LTL}}$ `false` for all $i \geqslant 0$
$\pi^i \models_{\text{LTL}} f$ iff $\pi(i) \models f$ if f is an unquantified first-order formula over V
$\pi^i \models_{\text{LTL}} \neg\varphi$ iff $\pi^i \not\models_{\text{LTL}} \varphi$
$\pi^i \models_{\text{LTL}} \varphi \wedge \psi$ iff $\pi^i \models_{\text{LTL}} \varphi$ and $\pi^i \models_{\text{LTL}} \psi$
$\pi^i \models_{\text{LTL}} \varphi \vee \psi$ iff $\pi^i \models_{\text{LTL}} \varphi$ or $\pi^i \models_{\text{LTL}} \psi$
$\pi^i \models_{\text{LTL}} \mathbf{X}\varphi$ iff $\pi^{i+1} \models_{\text{LTL}} \varphi$
$\pi^i \models_{\text{LTL}} \mathbf{G}\varphi$ iff $\pi^{i+j} \models_{\text{LTL}} \varphi$ for all $j \geqslant 0$
$\pi^i \models_{\text{LTL}} \mathbf{F}\varphi$ iff there exists $j \geqslant 0$ such that $\pi^{i+j} \models_{\text{LTL}} \varphi$
$\pi^i \models_{\text{LTL}} \varphi\mathbf{U}\psi$ iff there exists $j \geqslant 0$ such that $\pi^{i+j} \models_{\text{LTL}} \psi$ and $\pi^{i+k} \models_{\text{LTL}} \varphi$ for all $0 \leqslant k < j$
$\pi^i \models_{\text{LTL}} \varphi\mathbf{W}\psi$ iff $\pi^{i+k} \models_{\text{LTL}} \varphi$ for all $k \geqslant 0$, or there exists $j \geqslant 0$ such that $\pi^{i+j} \models_{\text{LTL}} \psi$ and $\pi^{i+k} \models_{\text{LTL}} \varphi$ for all $0 \leqslant k < j$

Safety Properties A *safety property* P is a collection of computations $\pi \in S^\omega$, such that for every $\pi' \in S^\omega$ with $\pi' \notin P$, the fact that π' does *not* fulfil P can already be decided on a finite prefix of π'. It has been shown in [15] that every safety property P can be characterised by a *Safety LTL* formula f, so that the computations in P are exactly those fulfilling f. The Safety LTL formulae are specified as follows [15, Theorem 3.1]:

(1) Every unquantified first-order formula is a Safety LTL-formula. (2) If f, g are Safety LTL-Formulae, then so are $f \wedge g$, $f \vee g$, $\mathbf{X}f$, $f\mathbf{W}g$, and $\mathbf{G}f$.

[8] The operators $\vee$, **G**, **F**, **U** are redundant and can be expressed using the remaining LTL operators alone. Therefore, they are sometimes introduced as syntactic abbreviations. For the purpose of this paper, however, it is better to represent their semantics in an explicit way.

Observe that in these safety formulae, the negation operator must only occur in first-order sub-formulae. Suppose that a safety property P is specified by Safety LTL formula f. When looking for a path π *violating* f, the violation $\pi \models_{\text{LTL}} \neg f$ can be equivalently expressed by a formula containing only first-order expressions composed by the operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$. This is stated in the following theorem proven in [14].

Theorem 1. *Let f be a Safety LTL formula. Then $\neg f$ can be equivalently expressed using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$.* □

Safety Violation Formulae on Finite Paths It will be explained in Section 4 how IXL configurations may be interpreted as Kripke structures. This interpretation needs one relaxation of the Kripke structure definition $K = (S, S_0, R, L, AP)$: we admit state transition systems (S, S_0, R) whose transition relation is no longer total. This leads to finite computations, because some states do not possess any post-states under R.

From Theorem 1 above we know that the LTL formulae we are interested in – these express safety violations – can be represented by $\wedge, \vee, \mathbf{X}, \mathbf{U}$. For these operators, the LTL semantics can be easily extended to finite computations by declaring the evaluation result to be `false` if the end of the path has been reached before the truth of the formula could be shown. The LTL semantics on finite computations has been investigated in [3], we only need a simplified version thereof, because it will only be applied to acyclic sub-models of IXL configurations.

Computation Tree Logic CTL While LTL formulae have computations of Kripke structures as models, CTL has trees of computations as models. As a consequence, two new *path quantors* are introduced in addition to the path operators already known from LTL: quantors $\mathbf{E}$ and $\mathbf{A}$ denote existential and universal path quantification, respectively. The CTL syntax is defined by the following grammar, where f denotes unquantified first-order formulae as specified above, formulae ϕ are called *state formulae*, and formulae ψ are called *path formulae*.

$$\begin{aligned}&\text{CTL-formula} ::= \phi\\ &\phi ::= f \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathbf{E}\,\psi \mid \mathbf{A}\,\psi\\ &\psi ::= \phi \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\,\phi \mid \mathbf{F}\,\phi \mid \mathbf{G}\,\phi \mid \phi\,\mathbf{U}\,\phi \mid \phi\,\mathbf{W}\,\phi\end{aligned}$$

According to this grammar, the path operators $\mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}, \mathbf{W}$ can never be prefixed by another temporal operator in CTL. Only pairs consisting of a path quantifier and a temporal operator can occur in a row.

The semantics of CTL formulae is explained using a Kripke structure K, specific states s of K and computations π of K. We write $K, s \models_{\text{CTL}} \phi$ to express that ϕ holds in state s of K. We write $K, \pi \models_{\text{CTL}} \psi$ to express that ψ holds along path π through K. For CTL formulae ϕ we say ϕ *holds in the Kripke*

model K and write $K \models_{\text{CTL}} \phi$ if and only if $K, s_0 \models_{\text{CTL}} \phi$ holds in every initial state s_0 of K. The semantics of the subset of CTL formulae we are interested in is specified in Table 2, where f denotes unquantified first-order formulae, ϕ, ϕ_i denote state formulae, and ψ, ψ_j denote path formulae. First-order formulae are interpreted just as in LTL.

Table 2. Semantics of CTL subset required for data validation.

$K, s \models_{\text{CTL}} f$	iff	$s \models f$ for any unquantified first-order formula f
$K, s \models_{\text{CTL}} \phi_1 \vee \phi_2$	iff	$K, s \models_{\text{CTL}} \phi_1$ or $K, s \models_{\text{CTL}} \phi_2$
$K, s \models_{\text{CTL}} \phi_1 \wedge \phi_2$	iff	$K, s \models_{\text{CTL}} \phi_1$ and $K, s \models_{\text{CTL}} \phi_2$
$K, s \models_{\text{CTL}} \mathbf{E}\,\psi$	iff	there is a path π from s such that $K, \pi^i \models_{\text{CTL}} \psi$
$K, \pi^i \models_{\text{CTL}} \phi$	iff	$K, \pi(i) \models_{\text{CTL}} \phi$
$K, \pi^i \models_{\text{CTL}} \psi_1 \vee \psi_2$	iff	$K, \pi^i \models_{\text{CTL}} \psi_1$ or $K, \pi^i \models_{\text{CTL}} \psi_2$
$K, \pi^i \models_{\text{CTL}} \psi_1 \wedge \psi_2$	iff	$K, \pi^i \models_{\text{CTL}} \psi_1$ and $K, \pi^i \models_{\text{CTL}} \psi_2$
$K, \pi^i \models_{\text{CTL}} \mathbf{X}\,\psi$	iff	$K, \pi^{i+1} \models_{\text{CTL}} \psi$
$K, \pi^i \models_{\text{CTL}} \psi_1 \mathbf{U} \psi_2$	iff	there exists $j \geq 0$ such that $K, \pi^{i+j} \models_{\text{CTL}} \psi_2$ and $K, \pi^{i+k} \models_{\text{CTL}} \psi_1$ for all $0 \leqslant k < j$

Over-approximation of LTL Safety Violation Formulae by CTL Full LTL and CTL have different expressiveness, and neither one is able to express all formulae of the other with equivalent semantics [6]. It can be shown, however, that any safety violation specified by an LTL formula f on a path π can also be detected by applying CTL model checking to a translated formula $\Phi(f)$ on any Kripke structure K containing π as a computation. This is, however, an *over-approximation*, in the sense that witnesses for $\Phi(f)$ in K will not always correspond to single paths π where $\pi \models_{\text{LTL}} f$ holds. It will be shown in Section 4 how the choice of sub-models significantly reduces the number of such false alarms.

Recalling from Theorem 1 that any safety violation can be specified using first-order formulae and operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$, we specify a partial transformation function $\Phi : \text{LTL} \nrightarrow \text{CTL}$ as follows, where f denotes a first-order formula and ψ_1, ψ_2 denote formulas containing path operators.

$$\Phi(f) = f$$
$$\Phi(\psi_1 \wedge \psi_2) = \Phi(\psi_1) \wedge \Phi(\psi_2) \qquad \Phi(\psi_1 \vee \psi_2) = \Phi(\psi_1) \vee \Phi(\psi_2)$$
$$\Phi(\mathbf{X}\psi_1) = \mathbf{EX}(\Phi(\psi_1)) \qquad \Phi(\psi_1 \mathbf{U} \psi_2) = \mathbf{E}(\Phi(\psi_1)\mathbf{U}\Phi(\psi_2))$$

With this transformation at hand, the following theorem states that the absence of witnesses for $\Phi(f)$ in K guarantees the absence of a rule violation f on π.

Theorem 2. *Let π be any path and f an LTL formula specifying a safety violation on π. Let K be a Kripke structure over state space S containing π as a computation. Then $\pi \models_{\mathrm{LTL}} f$ implies $K \models_{\mathrm{CTL}} \Phi(f)$.* □

With Theorem 2 at hand, we can apply the classical CTL model checking algorithms from [6] with small modifications related to first-order expressions, the resulting algorithms are specified in [14]. There, it is also shown that the algorithms are sound and complete for Kripke structures with finite computations. The algorithms have running time $O(|f| \cdot (|S| + |R|)$, where $|f|$ is the number of sub-formulae in CTL formula f, $|S|$ is the size of the state space, and $|R|$ is the size of the transition relation. As a consequence, the running time is affected by the model size in a linear way only, while model size may affect the running time of BMC in an exponential way. The running time is also lower than using LTL model checking algorithms directly, since the latter are PSPACE-complete [16].

4 Model Checking of IXL Configurations

IXL Configurations as Kripke Structures The configurations for geographical IXLs described in Section 2 give rise to Kripke structures $K = (S, S_0, R, L, AP)$ with variable symbols from some set V as follows (symbol d denotes `int`-values).

$$
\begin{array}{rcl}
V & = & \{id, t\} \cup C \cup A \\
C & = & \{c \mid c \text{ is a primary or secondary channel symbol}\} \\
A & = & \{a \mid a \text{ is an attribute}\} \\
S & = & \{s : V \to \text{int} \mid \text{There exists a configuration instance with} \\
 & & \qquad \text{id, type, channel, and attribute valuation } s\} \\
S_0 & = & S \\
R & = & \{(s, s') \mid \exists\, c \in C : s(c) = s'(id)\} \\
AP & = & \{id = d \mid \exists\, s \in S : s(id) = d\} \cup \{t = d \mid \exists\, s \in S : s(t) = d\} \cup \\
 & & \{c = d \mid c \in C \wedge \exists\, s \in S : s(c) = d\} \cup \\
 & & \{a = d \mid a \in A \wedge \exists\, s \in S : s(a) = d\} \\
L & : & S \to 2^{AP};\ \ s \mapsto \{v = d \mid v \in V \wedge s(v) = d\}
\end{array}
$$

Each K-state in S is represented by a valuation function s mapping id, type, channel, and attribute symbols to corresponding integer values, such that there is a configuration element with exactly these values. The atomic propositions consist of all equalities $v = d$, where v is a symbol of V and d an integer value occurring for v in at least one configuration element. Every K-state is an initial state, because configuration rules are checked from any element as starting point. Two elements s, s' are linked by the transition relation whenever s has a channel c connected to s'; this is expressed by $s(c)$ carrying the id of s'. The labelling function maps each state s exactly to the propositions $v = s(v),\ v \in V$ that are

valid in this state. Using the state valuation rules specified in Section 3, this can be equivalently expressed by $L(s) = \{v = d \mid s \models v = d\}$.

With the Kripke structure at hand, IXL configuration rules can be expressed by LTL Safety formulas, so rule violations may be expressed in LTL using first-order formulas and operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$, as shown in Section 3. Specifying rule violations on Kripke structure K representing a complete IXL configuration, however, is quite complicated, because most rules refer to routes traversed in a certain driving direction, whereas K's transition relation connects any pair of configuration elements linked by any channel. This results in computations that do not correspond to any "real" route through the network.

Example 3. The Kripke structure corresponding to the configuration shown in Fig. 1 has a path $s_{10}.s_{11}.s_{13}.s_{23}.s_{21}.s_{20}$, because all elements in this sequence are linked by some channel a, b, c. This path, however, cannot be realised as a train route, due to the topology of points s_{13} and s_{23}. □

In [9], this problem has been overcome by using existentially quantified LTL with rigid variables as introduced in [12]. Apart from the fact that quantified LTL formulae are harder to create and understand, this would not allow for the over-approximation by means of CTL as described in Section 3. Therefore, we will now introduce sub-models of full configuration models where the problem of infeasible paths no longer occurs.

Sub-models The *border elements* of an IXL configuration can be identified by the fact that only one of the main channels a, b is connected to another element, while the other channel is undefined. Element 20 in Fig. 1, for example, is a border element, because it has channel a connected to element 21, while channel b remains unconnected. Points or diamond crossings are never used as border elements, so only channels a, b need to be considered when identifying them in the Kripke structure K representing the complete configuration. Each border element introduces a well-defined driving direction specified by the channel which is defined and, therefore, "points into" the network specified by the configuration.

A sub-model is now created for every border element s_b as a Kripke structure $K(s_b)$ according to the following rules.

1. The driving direction corresponds to the direction specified by the defined channel a or b of border element s_b.
2. The sub-model is induced by the largest acyclic directed graph G with initial element s_b, such that
   - each element which is reachable in driving direction is part of this graph,
   - for points entered by their B-stem or C-stem, the only continuation is via the element connected to the points' A-stem,
   - for points entered by their A-stem, the continuations are via the elements connected to the points' B-stem or C-stem,
   - for diamond crossings entered via A,B,C,D-stem, the only possible continuations are via elements connected to the D,C,B,A-stems, respectively.

- The graph expansion stops when an element is reached for the second time.
- The graph expansion stops when a border element is reached by its defined channel, so that no outgoing channel is available.

3. The states of $K(s_b)$ are the nodes of G.
4. Every state is an initial state.
5. The transition relation of $K(s_b)$ contains all pairs of states (s, s'), such that there exists an edge from s to s' in G.
6. Every element of the sub-model is equipped with additional attributes *dirA*, *dirB*, *dirC*, *dirD* with value 1 if its respective channel a, b, c, or d points in driving direction; otherwise the attribute carries value 0.
7. Further auxiliary attributes are added to each sub-model state as described in Section 4 below.

Example 4. The complete IXL configuration depicted in Fig. 1 has border elements $s_{10}, s_{20}, s_{33}, s_{25}, s_{14}$. The sub-model resulting from border element s_{33} is shown in Fig. 2, together with the new auxiliary attributes $dirA, \ldots$ (the meaning of attribute *pCnt* is explained in Section 4 below). Element s_{33} induces the driving direction along its channel a; since it is a border element, its channel b is not linked to another element. □

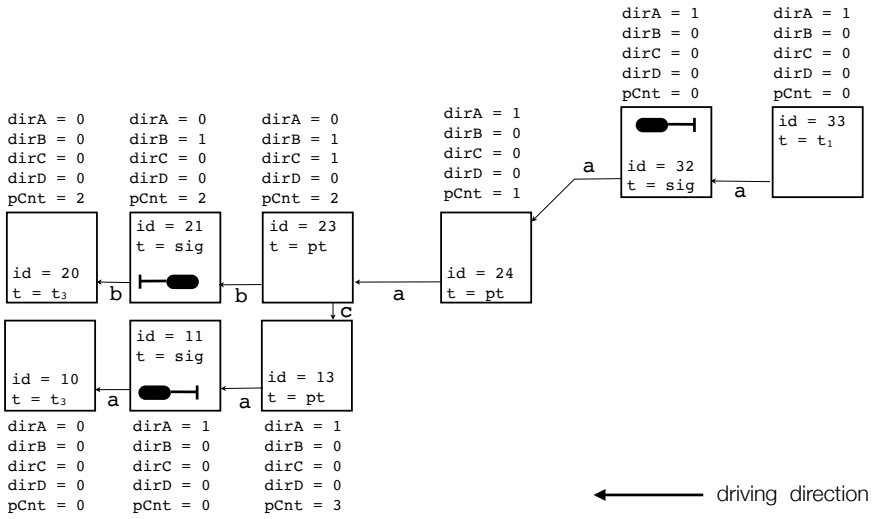

Fig. 2. Sub-model created from border element s_{33} in Fig. 1.

Specifying Rule violations on Sub-models The description of rule violations in LTL becomes rather straightforward when specified for sub-models; this is illustrated in the following examples.

Example 5. The rule violation specified in Example 1, when applied to a sub-model as the one depicted in Fig. 2, may be expressed in unquantified first-order LTL as $\phi_1 \equiv t = sig \wedge \mathrm{dirA} = 1 \wedge \mathbf{X}\big((t \neq sig \vee dirA = 0)\mathbf{U}(t = t_1 \vee t = t_3)\big)$. This LTL formula is translated via Φ defined in Section 3 into CTL formula $\Phi(\phi_1) \equiv t = sig \wedge \mathrm{dirA} = 1 \wedge \mathbf{EX}\Big(\mathbf{E}\big((t \neq sig \vee dirA = 0)\mathbf{U}(t = t_1 \vee t = t_3)\big)\Big)$. The only witness for $\Phi(\phi_1)$ in the sub-model shown on Fig. 2 is the path $s_{32}.s_{24}.s_{23}.s_{21}.s_{20}$, and this is also a witness for ϕ_1, so the CTL over-approximation does not produce any false alarms in this case. □

Example 6. The rule violation specified in Example 2, when applied to a sub-model, may be expressed in unquantified first-order LTL as $\phi_2 \equiv t = sig \wedge \mathrm{dirA} = 1 \wedge \mathbf{X}\big(t \neq t_3\mathbf{U}(t = sig \wedge \mathrm{dirA} = 1)\big)$. This LTL formula is translated via Φ defined in Section 3 into CTL formula $\Phi(\phi_2) \equiv t = sig \wedge \mathrm{dirA} = 1 \wedge \mathbf{EX}\Big(\mathbf{E}\big(t \neq t_3\mathbf{U}(t = sig \wedge \mathrm{dirA} = 1)\big)\Big)$. It is easy to see that for the sub-model shown in Fig. 2, the only witness is given by path $s_{32}.s_{24}.s_{23}.s_{13}.s_{11}$, so, again, there are no false alarms possible for this rule violation. □

Query Simplification by Auxiliary Parameters We have seen that auxiliary attributes can be introduced during sub-model creation, in order to facilitate the construction of rule violation formulae. Moreover, these attributes may be used to speed up the checking process.

Example 7. Another typical pattern of data validation rules restricts the number of elements of a certain type that may be allocated between two elements of another type. The following fictitious rule illustrates this pattern (the real rules are slightly more complex and refer to other element types).

Rule 3. From channel a of a signal of type *sig* pointing in downstream direction, no more than k points ($t = pt$) are allowed, before the corresponding signal with type *sig* and channel b pointing in upstream direction is reached.

Violation of Rule 3. From channel a of a signal of type *sig* pointing in downstream direction, more than k points ($t = pt$) are encountered, before the corresponding signal with type *sig* and channel b pointing in upstream direction is reached. □

In principle, rule violations as the one specified in Example 7 could be specified using *Counting LTL*, an extension of LTL allowing to check whether a path fulfils constraints referring to the number of states fulfilling certain properties [11]. Checking Counting LTL formulae, however, is EXPSPACE-complete, and therefore, we cannot expect to find model checking algorithms for Counting LTL that are as efficient as the CTL-algorithms presented above.

Instead, a new auxiliary attribute *pCnt* is introduced during sub-model creation. In every state of the sub-model, this attribute contains the number of points encountered in driving direction so far. It is reset to 0, as soon as a downstream signal is reached. This is illustrated in Fig. 2.

Example 8. With auxiliary attribute *pCnt* at hand, the violation of Rule 3 from Example 7 is specified in LTL as
$\phi_3 \equiv t = sig \wedge dirA = 1 \wedge \mathbf{X}\big((t \neq sig \vee dirA = 0)\mathbf{U}pCnt > k\big)$.
Translated to CTL, this results in

$$\Phi(\phi_3) \equiv t = sig \wedge dirA = 1 \wedge \mathbf{EX}\Big(\mathbf{E}\big((t \neq sig \vee dirA = 0)\mathbf{U}pCnt > k\big)\Big)$$

Assuming that $k \geqslant 3$, there are obviously no witnesses for $\Phi(\phi_3)$ in the sub-model from Fig. 2. For $k = 2$, checking $\Phi(\phi_3)$ results in witness $s_{32}.s_{24}.s_{23}.s_{13}$, and again, this is also a witness for the LTL formula ϕ_3. □

By analogy with the example shown here, further auxiliary attributes are added by the DVL-Checker during sub-model creation.

Parallelisation The concept to use sub-models for verifying DVL-queries allows for parallelisation of checking activities. The concurrent checking process receives file names of the DVL-query and the IXL configuration model to be verified. After the query and the configuration have been successfully parsed, all jobs to be performed are placed into a queue. A job consists of a triple (query,id,direction), where id is the identification of a border element. Attribute direction is A or B, depending on whether channel a or b of the border element is defined.

A predefined number of worker threads process these jobs concurrently, until the job queue is empty. Each thread pops a job from the (thread-safe) queue and creates the sub-model identified by id and direction. After that, the CTL checker functions are executed, and any witness found in the sub-model for the given query is written to the output interface.

Evaluation The efficiency of the CTL model checking algorithms in combination with the parallelisation allows for checking queries interactively, because the results are obtained in less than five seconds on standard PC hardware, even for the largest configurations used by Siemens. No false alarms have been encountered with the DVL-queries checked so far on the IXL configurations provided by Siemens.

The bounded model checking version used before as described in [9] could also produce witnesses for faulty configurations in acceptable time (less than 10 seconds), but was unable to prove the *absence* of errors, due to running time that was exponential in the length of the search paths and very high memory consumption.

5 Conclusion

We have presented an efficient model checking approach for data validation of geographical interlocking systems, which is fast enough to uncover violations of configuration rules or prove the absence of rule violations directly while designing the IXL configuration. The checking speed has been achieved by translating LTL formulae specifying rule violations to CTL formulae and using the "classical" global CTL model checking algorithms. It has been shown that for the class of LTL formulae specifying rule violations, CTL model checking is an over-approximation for the (slower) alternative checking for witnesses of LTL formulae directly. Therefore, the absence of CTL witnesses proves the absence of path segments fulfilling the original rule violation formula specified in LTL. Further speed-up has been achieved by running checks concurrently on configuration sub-models augmented by auxiliary attributes, instead of performing a single check on the full model.

The concepts and algorithms presented here have been implemented in the DVL-Checker tool which is used by Siemens for the validation of IXL configurations in new interlocking systems provided by Siemens for Belgian railways.

During the checks performed so far, no false alarms due to CTL over-approximation have been observed. It is planned to implement an automated detection of potential false alarms in the future: when CTL model checking results in an alarm for some rule violation $\Phi(\phi)$, it can be checked whether one of the finite paths π through the sub-model fulfil the original LTL formula ϕ. Since each sub-model is an acyclic directed graph, the paths π can be enumerated with low effort, and the LTL checks can also be parallelised. For checking the validity of ϕ on a finite path, the linear encodings of bounded LTL described in [3] are very efficient.

References

1. Badeau, F., Doche-Petit, M.: Formal Data Validation with Event-B. arXiv:1210.7039 [cs] (Oct 2012), http://arxiv.org/abs/1210.7039, arXiv: 1210.7039
2. Basile, D., ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F., Piattino, A., Trentini, D., Ferrari, A.: On the Industrial Uptake of Formal Methods in the Railway Domain. In: Furia, C.A., Winter, K. (eds.) Integrated Formal Methods. pp. 20–29. Lecture Notes in Computer Science, Springer International Publishing (2018)
3. Biere, A., Heljanko, K., Junttila, T., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Logical Methods in Computer Science 2(5) (Nov 2006), http://arxiv.org/abs/cs/0611029, arXiv: cs/0611029
4. Celebi, B.T., Kaymakci, O.T.: Verifying the accuracy of interlocking tables for railway signalling systems using abstract state machines. Journal of Modern Transportation 24(4), 277–283 (Dec 2016), https://doi.org/10.1007/s40534-016-0119-1
5. CENELEC: EN 50128:2011 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems (2011)

6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)
7. Fredj, M., Leger, S., Feliachi, A., Ordioni, J.: OVADO. In: Fantechi, A., Lecomte, T., Romanovsky, A. (eds.) Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification. pp. 87–98. Lecture Notes in Computer Science, Springer International Publishing (2017)
8. Hansen, D., Schneider, D., Leuschel, M.: Using B and ProB for Data Validation Projects. In: Butler, M., Schewe, K.D., Mashkoor, A., Biro, M. (eds.) Abstract State Machines, Alloy, B, TLA, VDM, and Z. pp. 167–182. Lecture Notes in Computer Science, Springer International Publishing (2016)
9. Haxthausen, A.E., Peleska, J., Pinger, R.: Applied bounded model checking for interlocking system designs. In: Counsell, S., Núñez, M. (eds.) SEFM Workshops. Lecture Notes in Computer Science, vol. 8368, pp. 205–220. Springer (2013)
10. Keming, W., Zheng, W., Chuandong, Z.: Formal modeling and data validation of general railway interlocking system. WIT Transactions on The Built Environment 181 (2019)
11. Laroussinie, F., Meyer, A., Petonnet, E.: Counting LTL. In: Markey, N., Wijsen, J. (eds.) TIME 2010 - 17th International Symposium on Temporal Representation and Reasoning, Paris, France, 6-8 September 2010. pp. 51–58. IEEE Computer Society (2010), https://doi.org/10.1109/TIME.2010.20
12. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems - specification. Springer (1992)
13. Pachl, J.: Railway Operation and Control. VTD Rail Publishing (January 2002)
14. Peleska, J., Krafczyk, N., Haxthausen, A.E., Pinger, R.: Efficient data validation for geographical interlocking systems. Tech. rep., Embedded Systems Testing Benchmarks Site (2019-01-13), http://www.informatik.uni-bremen.de/agbs/jp/papers/dvl2019.pdf
15. Sistla, A.P.: Safety, liveness and fairness in temporal logic. Formal Aspects of Computing 6(5), 495–511 (Sep 1994), http://link.springer.com/article/10.1007/BF01211865
16. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM 32(3), 733–749 (1985), https://doi.org/10.1145/3828.3837