Efficient Data Validation for Geographical Interlocking Systems Technical Report Issue 1.1 – 2019-01-17

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 [22, 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 rules collection is usually quite extensive (several hundred), so that manual data validation would be 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 [6]. Therefore, it is desirable to use data validation tools processing a logical query language to specify which rules should be enforced or which rule violations should be detected. This type of tool can be validated once and for all, since new validation rules can be specified by mans of new queries, without changing the software code.

Previous Work This paper is a follow-up contribution to [14], 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 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 [14], 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 violations specified by ϕ are 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 LTL model checking, since the latter represents an NP-hard problem [7, Section 4.2].
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, and (2) significant speed-up of the checking process by processing sub-models concurrently.

[4] The text of the previous paragraph describing the general problem and the more detailed description in Section 2 have been reproduced in slightly modified form from [14], in order to make the present paper self-contained.

[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.

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. Section 5 contains references to related work and competing approaches. In Section 6, a conclusion is presented.

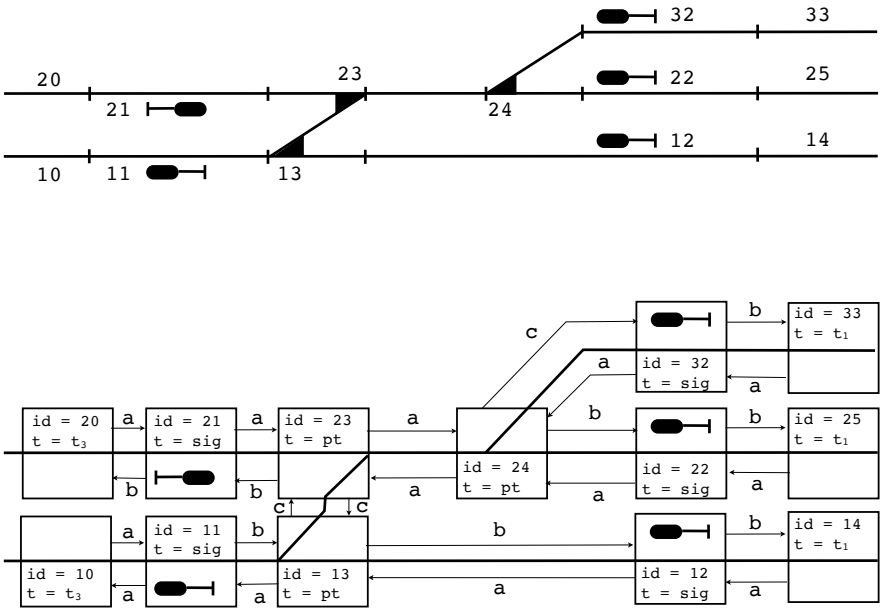

Fig. 1. Physical layout, associated software instances and channel connections.

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 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 $e, f, g, \ldots$. Only points and diamond crossings use c-channels, and d-channels are used by diamond crossings only.

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 is reached along π_1 via its a-channel. 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. □

Example 3. 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.

The corresponding rule violation is specified as

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. □

Slightly more complex rules have to be specified for ensuring the correct configuration of elements offering flank protection to routes crossing points. In Fig. 2, several variants of signals and points offering flank protection to point p_1 are shown. Note that several more variants have to be considered in practise.

Flank protection by point is the preferred solution. Driving direction AB/BA of a point p_1 can be protected from trains entering the C-stem of p_1, if another point p_2 exists that may prevent trains from entering p_1's C-stem. This is illustrated in Fig. 2 (c). Driving direction AC/CA is protected from trains entering the B-stem of p_1 by points p_2 shown in Fig. 2 (d). If flank protection by point is not possible, then protection by signal may be realised for driving directions AB/BA and AC/CA as shown in Fig. 2 (a) and (b), respectively.

Example 4. The variants of flank protection shown in Fig. 2 lead to the following rules applicable to every element p_1 with type $t = pt$. It suffices to check flank protection for one driving direction, because then it also holds for the opposite driving direction. Therefore, the rules are only formulated for the case where the B and C-stems of the point under consideration point in driving direction.

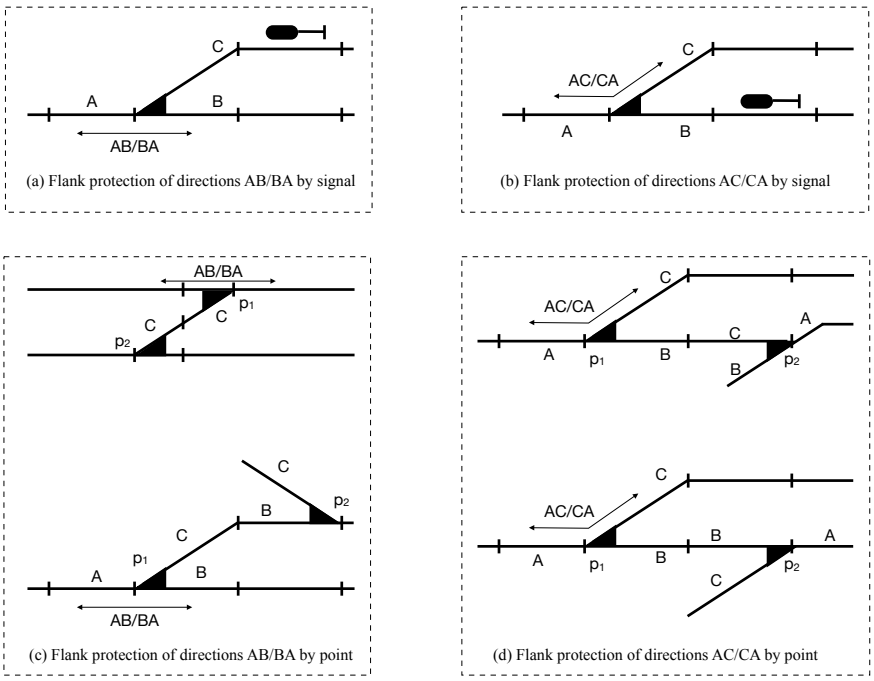

Fig. 2. Several variants of flank protection.

Rule 4.1 (protection of driving direction AB/BA) If p_1's c-channel points in downstream direction, another point p_2 with its b channel or c-channel pointing towards the C-stem of p_1 is required, or a signal with a-channel pointing towards the C-stem of p_1 is required before another point p_3 with its a-channel pointing towards the C-stem of p_1 is encountered.
The condition about p_3 ensures that the flank protection is implemented not too far away from the point p_1 to be protected: after encountering a point like p_3, two signals instead of one would be required to protect p_1, because trains could approach p_1's C-stem via the B-stem or A-stem of p_3.

Rule 4.2 (protection of driving direction AC/CA) If p_1's b-channel points in downstream direction, another point p_2 with its b channel or c-channel pointing towards the B-stem of p_1 is required, or a signal with a-channel pointing towards the B-stem of p_1 is required before another point p_3 with its a-channel pointing towards the B-stem of p_1 is encountered.

For all points displayed in Fig. 1, Rule 4.1 and Rule 4.2 are fulfilled. The corresponding rule violations are specified as

Violation of Rule 4.1 If p_1's c-channel points in downstream direction, no other point p_2 with its b channel or c-channel pointing towards the C-stem

of p_1 can be found, and no signal with a-channel pointing towards the C-stem of p_1 can be found before another point p_3 with its a-channel pointing towards the C-stem of p_1 is encountered.

Violation of Rule 4.2 If p_1's b-channel points in downstream direction, no other point p_2 with its b channel or c-channel pointing towards the B-stem of p_1 can be found, and no signal with a-channel pointing towards the B-stem of p_1 can be found before another point p_3 with its a-channel pointing towards the B-stem of p_1 is encountered. □

3 Logical Foundations

In this section, the logical foundations of the model checking method for data validation are explained. The underlying theory is described without references to their practical application in the IXL context; the latter is explained in Section 4.

3.1 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 by 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.

3.2 First Order Formulae and Their Valuation

Given a Kripke Structure K with variable valuation functions $s : V \to \text{int}$ as states, arithmetic expressions over variables from V are interpreted in a given

state s by the rules shown in Table 1. These rules extend the domain of each valuation s to integer constants and arithmetic expressions over variables from V.

Table 1. Expression evaluation.

$s(d) = d$	for integer constants d
$s(x\,\omega\,e) = s(x)\,\omega\,s(e)$	for variables x and expressions e and arithmetic operators $\omega \in \{+, -, /, *, <<, >>, \%\}$

Table 2. Semantics of atomic propositions.

$s \models \texttt{true}$
$s \not\models \texttt{false}$
$s \models v\,\nu\,d$ iff $s(v)\,\nu\,d$ for comparison operators $\nu \in \{=, \neq, <, \leqslant, >, \geqslant\}$
$s \models v\,\nu\,w$ iff $s(v)\,\nu\,s(w)$

Table 3. Semantics of first-order formulae.

$s \models \neg f$ iff $s \not\models f$
$s \models f \wedge g$ iff $s \models f$ and $s \models g$
$s \models f \vee g$ iff $s \models f$ or $s \models g$

Atomic propositions are constructed by composing variables or arithmetic expressions using comparison operators. The valuation of atomic propositions is specified in Table 2, where d denotes integer constants, and v, w denote variables from V or arithmetic expressions over variables from V. We write $s \models p$ if p evaluates to `true` in state s.

An *(unquantified) first-order formula* f over V is a logical formula with atomic propositions over V as specified above, composed by logical operators $\neg, \wedge, \vee$. The domain of valuation functions s is extended once more to first-order formulae, as specified in Table 3.

3.3 Linear Temporal Logic LTL – Safety Properties and Their Violations

Linear Temporal Logic LTL *Linear Temporal Logic (LTL)* is a logical formalism aiming at the specification of computation properties. The material presented here is based on [7]. 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*.

Table 4. Semantics of LTL formulae.

$\pi^i \models_{\mathrm{LTL}} \texttt{true}$ for all $i \geqslant 0$

$\pi^i \not\models_{\mathrm{LTL}} \texttt{false}$ for all $i \geqslant 0$

$\pi^i \models_{\mathrm{LTL}} f$ iff $\pi(i) \models f$ if f is an unquantified first-order formula over V, to be evaluated as specified in Table 1, 2 and 3.

$\pi^i \models_{\mathrm{LTL}} \neg\varphi$ iff $\pi^i \not\models_{\mathrm{LTL}} \varphi$

$\pi^i \models_{\mathrm{LTL}} \varphi \wedge \psi$ iff $\pi^i \models_{\mathrm{LTL}} \varphi$ and $\pi^i \models_{\mathrm{LTL}} \psi$

$\pi^i \models_{\mathrm{LTL}} \varphi \vee \psi$ iff $\pi^i \models_{\mathrm{LTL}} \varphi$ or $\pi^i \models_{\mathrm{LTL}} \psi$

$\pi^i \models_{\mathrm{LTL}} \mathbf{X}\varphi$ iff $\pi^{i+1} \models_{\mathrm{LTL}} \varphi$

$\pi^i \models_{\mathrm{LTL}} \mathbf{G}\varphi$ iff $\pi^{i+j} \models_{\mathrm{LTL}} \varphi$ for all $j \geqslant 0$

$\pi^i \models_{\mathrm{LTL}} \mathbf{F}\varphi$ iff there exists $j \geqslant 0$ such that $\pi^{i+j} \models_{\mathrm{LTL}} \varphi$

$\pi^i \models_{\mathrm{LTL}} \varphi\mathbf{U}\psi$ iff there exists $j \geqslant 0$ such that $\pi^{i+j} \models_{\mathrm{LTL}} \psi$ and $\pi^{i+k} \models_{\mathrm{LTL}} \varphi$ for all $0 \leqslant k < j$

$\pi^i \models_{\mathrm{LTL}} \varphi\mathbf{W}\psi$ iff
$\pi^{i+k} \models_{\mathrm{LTL}} \varphi$ for all $k \geqslant 0$,
or there exists $j \geqslant 0$ such that $\pi^{i+j} \models_{\mathrm{LTL}} \psi$ and
$\pi^{i+k} \models_{\mathrm{LTL}} \varphi$ for all $0 \leqslant k < j$

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 4.[7] 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 if and only if every computation of K is a model of f:

$$K \models_{\text{LTL}} f \text{ iff } \pi \models_{\text{LTL}} f \text{ for all computations } \pi \text{ of } K$$

In the remainder of the paper, some equivalences between LTL formulae will be used in proofs. These are listed in the following lemma.

Lemma 1. *Let φ, ψ be LTL formulae. Then*

$$\begin{array}{rrr}
\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi) & \neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi & \neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi \\
\mathbf{G}\varphi \equiv \varphi\,\mathbf{W}\,\mathtt{false} & \mathbf{F}\varphi \equiv \neg\mathbf{G}\neg\varphi & \varphi\mathbf{U}\psi \equiv \varphi\mathbf{W}\psi \wedge \mathbf{F}\psi \\
\mathbf{F}\varphi \equiv \mathtt{true}\mathbf{U}\varphi & \neg\mathbf{X}\varphi \equiv \mathbf{X}\neg\varphi & \neg\mathbf{G}\varphi \equiv \mathbf{F}\neg\varphi \\
\neg(\varphi\mathbf{W}\psi) \equiv \big(\neg\psi\mathbf{U}\neg(\varphi \vee \psi)\big) & &
\end{array}$$

Proof. We prove $\neg(\varphi\mathbf{W}\psi) \equiv \big(\neg\psi\mathbf{U}\neg(\varphi \vee \psi)\big)$ by transforming the left-hand side and right-hand side into their semantic representation and proving semantic equivalence of the resulting quantified first-order expressions. The other statements are established in an analogous way.

$$\begin{array}{ll}
& \pi^i \models_{\text{LTL}} \neg(\varphi\mathbf{W}\psi) \\
\Leftrightarrow & \pi^i \not\models_{\text{LTL}} \varphi\mathbf{W}\psi \qquad \text{[Semantics of } \neg\text{, Table 4]} \\
\Leftrightarrow & \neg\big(\forall\, k \geqslant 0 : \pi^{i+k} \models_{\text{LTL}} \varphi\big) \wedge \\
& \quad \neg\big(\exists\, j \geqslant 0 : (\pi^{i+j} \models_{\text{LTL}} \psi \wedge \forall\, 0 \leqslant k < j : \pi^{i+k} \models_{\text{LTL}} \varphi)\big) \\
& \qquad \text{[Semantics of } \mathbf{W}\text{, negated]} \\
\Leftrightarrow & \big(\exists\, h \geqslant 0 : \pi^{i+h} \not\models_{\text{LTL}} \varphi\big) \wedge \\
& \quad \big(\forall\, j \geqslant 0 : (\pi^{i+j} \not\models_{\text{LTL}} \psi \vee \exists\, 0 \leqslant k < j : \pi^{i+k} \not\models_{\text{LTL}} \varphi)\big) \\
& \qquad \text{[First-order logic rules for negation and quantification]} \\
\Leftrightarrow & \big(\exists\, h \geqslant 0 : \pi^{i+h} \models_{\text{LTL}} \neg\varphi\big) \wedge \\
& \quad \big(\forall\, j \geqslant 0 : (\pi^{i+j} \models_{\text{LTL}} \neg\psi \vee \exists\, 0 \leqslant k < j : \pi^{i+k} \models_{\text{LTL}} \neg\varphi)\big) \\
& \qquad \text{[LTL semantics of } \neg\text{]} \\
\Leftrightarrow & \big((\exists\, h \geqslant 0 : \pi^{i+h} \models_{\text{LTL}} \neg\varphi) \wedge (\forall\, j \geqslant 0 : \pi^{i+j} \models_{\text{LTL}} \neg\psi)\big) \vee \\
& \quad \big(\exists\, j \geqslant 0 : (\pi^{i+j} \models_{\text{LTL}} \psi \wedge \forall\ 0 \leqslant k < j : \pi^k \models_{\text{LTL}} \neg\psi \wedge \exists\, 0 \leqslant h < j : \pi^{i+h} \models_{\text{LTL}} \neg\varphi)\big) \\
& \qquad \text{[First-order logic rules for } \vee\text{, } \wedge\text{, } \forall\text{, and } \exists\text{,]} \\
& \qquad \text{[note that second disjunct implies } \exists\, h \geqslant 0 : \pi^{i+h} \models_{\text{LTL}} \neg\varphi\text{]} \\
\Leftrightarrow & \big(\exists\, h \geqslant 0 : (\pi^{i+h} \models_{\text{LTL}} (\neg\varphi \wedge \neg\psi) \wedge \forall\, 0 \leqslant k < h : \pi^{i+k} \models_{\text{LTL}} \neg\psi)\big) \\
& \qquad \text{[First-order logic rules]} \\
\Leftrightarrow & \pi^i \models_{\text{LTL}} \big(\neg\psi\mathbf{U}\neg(\varphi \vee \psi)\big) \qquad \text{[LTL semantics of } \mathbf{U}\text{, rules for } \wedge, \vee\text{]}
\end{array}$$

□

[7] 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.

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 [23] 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 [23, 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

$$f \wedge g, \quad f \vee g, \quad \mathbf{X}f, \quad f\mathbf{W}g, \quad \mathbf{G}f.$$

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 $\pi \models_{\mathrm{LTL}} \neg f$ can be equivalently expressed by a formula containing only first-order expressions composed by the operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$. This is shown in the following theorem.

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

Proof. We use structural induction over the syntax of safety LTL formulae.

Base case. If f is a first-order expression, then its negation is again a first-order expression.

Induction hypothesis. Suppose that the negation of Safety LTL formulae f, g can be expressed using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$ only.

Induction step. Since every Safety LTL formula can be expressed using operators $\wedge, \vee, \mathbf{X}, \mathbf{W}, \mathbf{G}$, we need to show that the negations of $f \wedge g$, $f \vee g$, $\mathbf{X}f$, $f\mathbf{W}g$, $\mathbf{G}f$ can also be expressed using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$. To prove this, we use the equivalences for LTL formulae established in Lemma 1.

Since $\neg(f \wedge g) \equiv \neg f \vee \neg g$ and f, g can be negated using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$ only, the induction step holds for operator $\wedge$.

Since $\neg(f \vee g) \equiv \neg f \wedge \neg g$ and f, g can be negated using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$ only, the induction step holds for operator $\vee$.

Since $\neg\mathbf{X}f \equiv \mathbf{X}\neg f$ and f can be negated using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$ only, the induction step holds for operator $\mathbf{X}$.

Since $\neg(f\mathbf{W}g) \equiv \big(\neg g\mathbf{U}\neg(f \vee g)\big) \equiv \big(\neg g\mathbf{U}(\neg f \wedge \neg g)\big)$ and f, g can be negated using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$ only, the induction step holds for operator $\mathbf{W}$.

Since $\neg\mathbf{G}f \equiv \mathbf{F}\neg f \equiv (\mathtt{true}\mathbf{U}\neg f)$ and f can be negated using first-order expressions composed by operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$ only, the induction step holds for operator $\mathbf{G}$. This completes the proof. □

As a consequence of Theorem 1, a model checker specialised on the detection of safety violations only needs to support the evaluation of first-order formulae and 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 the possibility that computations are finite, because some states may not possess any post-states under R. These states correspond to boundary elements in the IXL configuration that may be reached along paths through the rail network controlled by the IXL but do not possess outgoing main channels in the given driving direction.

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 finite paths π, we introduce the convention that

$$\pi^i = \begin{cases} \pi(i) \dots \pi(\#\pi - 1) \text{ iff } \#\pi > i \ (\#\pi \text{ is the length of } \pi) \\ \varepsilon \text{ (empty path)} \quad \text{iff } i \geqslant \#\pi \end{cases}$$

Next, the additional semantic rule

$$\varepsilon \not\models_{\mathrm{LTL}} \varphi \text{ for all LTL formulae } \varphi$$

is introduced. With this addition, the LTL semantics specified in Table 4 can be applied to finite paths as well: $\mathbf{X}\varphi$ is always false when evaluated on a path segment of length 1 or 0.

Theorem 2. *If the transition relation R of a Kripke structure K is not total and K can be represented as an acyclic finite directed graph, then the semantic extension of LTL to finite paths specified above coincides with the finite linear encodings for LTL semantics introduced in [3] that is used for bounded LTL model checking.* □

3.4 Computation Tree Logic CTL

Syntax of CTL formulae. 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: Quantor **E** denotes existential path quantification, in the sense that *"there exists a path segment starting at the current node of the computation tree, such that the formula specified after* **E** *holds on this segment."* Quantor **A** denotes universal path quantification, in the sense that *"on all path segments starting at the current node of the computation tree the formula specified after* **A** *holds."* The CTL syntax is defined by the following grammar, where

f denotes unquantified first-order formulae as specified in Section 3.2, formulae ϕ are called *state formulae*, and formulae ψ are called *path formulae*.

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

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 path quantifier and temporal operator can occur in a row.

Table 5. Semantics of CTL formulae.

$K, s \models_{\mathrm{CTL}} f$	iff	$s \models f$ for any unquantified first-order formula f with "$\models$" as defined in Table 3
$K, s \models_{\mathrm{CTL}} \neg\phi$	iff	$K, s \not\models_{\mathrm{CTL}} \phi$
$K, s \models_{\mathrm{CTL}} \phi_1 \vee \phi_2$	iff	$K, s \models_{\mathrm{CTL}} \phi_1$ or $K, s \models_{\mathrm{CTL}} \phi_2$
$K, s \models_{\mathrm{CTL}} \phi_1 \wedge \phi_2$	iff	$K, s \models_{\mathrm{CTL}} \phi_1$ and $K, s \models_{\mathrm{CTL}} \phi_2$
$K, s \models_{\mathrm{CTL}} \mathbf{E}\,\psi$	iff	there is a path π from s such that $K, \pi^i \models_{\mathrm{CTL}} \psi$
$K, s \models_{\mathrm{CTL}} \mathbf{A}\,\psi$	iff	on every path π from s holds $K, \pi^i \models_{\mathrm{CTL}} \psi$
$K, \pi^i \models_{\mathrm{CTL}} \phi$	iff	$K, \pi(i) \models_{\mathrm{CTL}} \phi$
$K, \pi^i \models_{\mathrm{CTL}} \neg\psi$	iff	$K, \pi^i \not\models_{\mathrm{CTL}} \psi$
$K, \pi^i \models_{\mathrm{CTL}} \psi_1 \vee \psi_2$	iff	$K, \pi^i \models_{\mathrm{CTL}} \psi_1$ or $K, \pi^i \models_{\mathrm{CTL}} \psi_2$
$K, \pi^i \models_{\mathrm{CTL}} \psi_1 \wedge \psi_2$	iff	$K, \pi^i \models_{\mathrm{CTL}} \psi_1$ and $K, \pi^i \models_{\mathrm{CTL}} \psi_2$
$K, \pi^i \models_{\mathrm{CTL}} \mathbf{X}\,\psi$	iff	$K, \pi^{i+1} \models_{\mathrm{CTL}} \psi$
$K, \pi^i \models_{\mathrm{CTL}} \mathbf{G}\,\psi$	iff	for all $k \geq 0$ $K, \pi^{i+k} \models_{\mathrm{CTL}} \psi$
$K, \pi^i \models_{\mathrm{CTL}} \mathbf{F}\,\psi$	iff	there exists $k \geq 0$ such that $K, \pi^{i+k} \models_{\mathrm{CTL}} \psi$
$K, \pi^i \models_{\mathrm{CTL}} \psi_1 \mathbf{U} \psi_2$	iff	there exists $j \geq 0$ such that $K, \pi^{i+j} \models_{\mathrm{CTL}} \psi_2$ and $K, \pi^{i+k} \models_{\mathrm{CTL}} \psi_1$ for all $0 \leqslant k < j$
$K, \pi^i \models_{\mathrm{CTL}} \psi_1 \mathbf{W} \psi_2$	iff	$\pi^{i+k} \models_{\mathrm{CTL}} \psi_1$ for all $k \geqslant 0$, or there exists $j \geqslant 0$ such that $K, \pi^{i+j} \models_{\mathrm{CTL}} \psi_2$ and $K, \pi^{i+k} \models_{\mathrm{CTL}} \psi_1$ for all $0 \leqslant k < j$

Semantics of CTL formulae. The semantics of CTL formulae is explained using a Kripke structure K, specific states s of K and paths π through the computation tree of K. We write

$K, s \models_{\mathrm{CTL}} \phi$ (s a state of K, ϕ a state formula)

to express that ϕ holds in state s of K. We write

$K, \pi \models_{\mathrm{CTL}} \psi$ (π a computation of K, ψ a path formula)

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 CTL formulae is specified in Table 5, 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, as specified in Table 3.

3.5 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 [7]. In this section, however, it will be shown 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 "real" rule violations in the IXL configuration. This will be illustrated by examples, and it is explained why the choice of sub-models described in Section 4.2 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.

$$\begin{aligned} \Phi(f) &= f \text{ for all first-order expressions } f \\ \Phi(f \wedge g) &= \Phi(f) \wedge \Phi(g) \\ \Phi(f \vee g) &= \Phi(f) \vee \Phi(g) \\ \Phi(\mathbf{X}f) &= \mathbf{EX}(\Phi(f)) \\ \Phi(f\mathbf{U}g) &= \mathbf{E}(\Phi(f)\mathbf{U}\Phi(g)) \end{aligned}$$

Observe that Φ maps every LTL formula in its domain to a CTL state formula, since first-order expressions are state-formulae, and any LTL formula starting with a temporal operator is prefixed under Φ with the existential path quantor $\mathbf{E}$. 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 3. *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*

$$\pi \models_{\text{LTL}} f \quad \textit{implies} \quad K \models_{\text{CTL}} \Phi(f).$$

Proof. The proof uses structural induction over the syntax of LTL formulae representing safety violations. These are expressed by first-order formulae and operators $\wedge, \vee, \mathbf{X}, \mathbf{U}$ according to Theorem 1.

Base case. Let π be a path and f a first-order formula, such that $\pi \models_{\text{LTL}} f$. According to the semantic rules of LTL specified in Table 4, this is equivalent

to $\pi(0) \models f$. This means that state $\pi(0)$ is a model for f. Now let K be any Kripke structure possessing π as a computation. Then $\pi(0)$ is an initial state of K. Since expression evaluation is the same for LTL and CTL, $K, \pi(0) \models_{\mathrm{CTL}} f$ follows from $\pi(0) \models f$. Observing that $\Phi(f) = f$ for first-order formulae f, we have shown that $\pi \models_{\mathrm{LTL}} f$ implies $K \models_{\mathrm{CTL}} \Phi(f)$ for first-order formulae f.

Induction hypothesis. Suppose that $\pi^i \models_{\mathrm{LTL}} f$ and $\pi^i \models_{\mathrm{LTL}} g$ implies $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f)$ and $K, \pi(i) \models_{\mathrm{CTL}} \Phi(g)$, respectively, for given LTL formulae f, g expressing safety violations and any path π for any $i \geqslant 0$, and any K containing π as a computation.

Induction step. Using the induction hypothesis, it has to be shown that the validity of $\pi^i \models_{\mathrm{LTL}} f \wedge g$, $\pi^i \models_{\mathrm{LTL}} f \vee g$, $\pi^i \models_{\mathrm{LTL}} \mathbf{X}f$, and $\pi^i \models_{\mathrm{LTL}} f\mathbf{U}g$ implies $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f) \wedge \Phi(g)$, $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f) \vee \Phi(g)$, $K, \pi(i) \models_{\mathrm{CTL}} \mathbf{X}\Phi(f)$, and $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f)\mathbf{U}\Phi(g)$, respectively.

If $\pi^i \models_{\mathrm{LTL}} f \wedge g$, this is equivalent to $\pi^i \models_{\mathrm{LTL}} f$ and $\pi^i \models_{\mathrm{LTL}} g$ according to the LTL semantics specified in Table 4. According to the induction hypothesis, this implies $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f)$ and $K, \pi(i) \models_{\mathrm{CTL}} \Phi(g)$. According to the CTL semantics specified in Table 5, this is equivalent to $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f) \wedge \Phi(g)$.

If $\pi^i \models_{\mathrm{LTL}} f \vee g$, this is equivalent to $\pi^i \models_{\mathrm{LTL}} f$ or $\pi^i \models_{\mathrm{LTL}} g$ according to the LTL semantics specified in Table 4. According to the induction hypothesis, this implies $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f)$ or $K, \pi(i) \models_{\mathrm{CTL}} \Phi(g)$. According to the CTL semantics specified in Table 5, this is equivalent to $K, \pi(i) \models_{\mathrm{CTL}} \Phi(f) \vee \Phi(g)$.

If $\pi^i \models_{\mathrm{LTL}} \mathbf{X}f$, this is equivalent to $\pi^{i+1} \models_{\mathrm{LTL}} f$ according to the LTL semantics specified in Table 4. According to the induction hypothesis, this implies $K, \pi(i+1) \models_{\mathrm{CTL}} \Phi(f)$. Since $\Phi(f)$ is a state formula, this is equivalent to $K, \pi^{i+1} \models_{\mathrm{CTL}} \Phi(f)$ according to the CTL semantics specified in Table 5. Furthermore, the CTL semantics states that this is equivalent to $K, \pi \models_{\mathrm{CTL}} \mathbf{X}\Phi(f)$. This establishes the existence of a computation in K (namely π) where $\mathbf{X}\Phi(f)$ is fulfilled. This fact is equivalent to $K \models_{\mathrm{CTL}} \mathbf{EX}\Phi(f)$ which is again equivalent to $K \models_{\mathrm{CTL}} \Phi(\mathbf{X}f)$ by definition of Φ.

If $\pi^i \models_{\mathrm{LTL}} f\mathbf{U}g$, this is equivalent to

$$\exists\, j \geqslant 0 : \big(\pi^{i+j} \models_{\mathrm{LTL}} g \wedge \forall\, 0 \leqslant k < j : \pi^{i+k} \models_{\mathrm{LTL}} f\big)$$

according to the LTL semantics specified in Table 4. This implies

$$\exists\, j \geqslant 0 : \big(\pi(i+j) \models_{\mathrm{CTL}} \Phi(g) \wedge \forall\, 0 \leqslant k < j : \pi(i+k) \models_{\mathrm{CTL}} \Phi(f)\big)$$

according to the induction hypothesis. Since $\Phi(f), \Phi(g)$ are state formulae, the CTL semantics specified in Table 5 states that this equivalent to

$$\exists\, j \geqslant 0 : \big(\pi^{i+j} \models_{\mathrm{CTL}} \Phi(g) \wedge \forall\, 0 \leqslant k < j : \pi^{i+k} \models_{\mathrm{CTL}} \Phi(f)\big),$$

which establishes

$$M, \pi \models_{\mathrm{CTL}} \Phi(f)\mathbf{U}\Phi(g).$$

Now we have shown the existence of a computation of K (namely π) where $\Phi(f)\mathbf{U}\Phi(g)$ holds; this can be re-phrased as $K \models_{\mathrm{CTL}} \mathbf{E}(\Phi(f)\mathbf{U}\Phi(g))$ which is

equivalent to $K \models_{\mathrm{CTL}} \Phi(f \mathbf{U} g)$ according to the definition of Φ. This completes the induction step and the proof of Theorem 3. □

3.6 CTL Model Checking

Basic Concept of Classical CTL Model Checking. The CTL model checking algorithm used for IXL data validation is based on the "classical" algorithm described in [7, Chapter 4]. It is specialised, however, on the CTL syntax required for uncovering safety violations. From Theorem 1 and Theorem 3 we know that for this purpose, only unquantified first-order formulae and the CTL operators $\wedge, \vee, \mathbf{EX}, \mathbf{EU}$ need to be supported. The algorithm's main concepts are summarised as follows.

- The CTL specification formula is decomposed into its (binary) syntax tree.
- Starting at the leaves of the syntax tree (the leaves represent unquantified first-order formulae), the algorithm processes a sequence of sub-formulae ϕ_i in bottom-up manner. This is implemented by means of a recursive in-order traversal of the syntax tree.
- The goal of each processing step is to annotate all states $s \in S$ satisfying $s \models_{\mathrm{CTL}} \phi_i$ with the new sub-formula ϕ_i. To this end, a labelling function $L_\phi : S \to CTL$ is used.
- The algorithm stops when the last formula ϕ_i having been processed coincides with the specification ϕ.
- The result of the algorithm is the set $S_\phi = \{s \in S \mid \phi \in L_\phi(s)\}$.
- The Kripke model (S, S_0, R, L, AP) satisfies ϕ if its initial states are a subset of S_ϕ.
- For the DVL Checker, the initial model states are the entry elements into the track network controlled by the interlocking system.

Overview over the algorithm. In Fig. 3, the entry function of the recursive algorithm is shown. *checkCTL* returns the set $\{s \in S \mid \phi \in \mathrm{label}(s)\}$ of all states satisfying the given formula ψ. It remains to check whether the initial states S_0 of the Kripke Structure K form a subset of $\{s \in S \mid \phi \in \mathrm{label}(s)\}$.

In Fig. 4, the main function *calcLabel* of the algorithm is shown. It traverses the syntax tree representation of the formula ψ to be checked and calls recursively itself or special sub-functions for processing sub-formulae.

Complexity Considerations. Studying the algorithms below, it is easy to see that the running time for checking $K \models_{\mathrm{CTL}} f$ is $O(|f| \cdot (|S| + |R|))$, 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. This is a well-known result which is elaborated, for example, in [7, Theorem 1]. 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 NP-hard [7, Section 4.2].

---

function *checkCTL*(**in** (S, S_0, R, L, AP) : KripkeStructure; **in** ϕ : CTL) : $\mathbb{P}(S)$
begin
  label : $S \to 2^{\mathrm{CTL}}$;
  label := $\{s \mapsto \texttt{true} \mid s \in S\}$;
  calcLabel$((S, S_0, R, L, AP), \phi,$ label);
  checkCTL := $\{s \in S \mid \phi \in \mathrm{label}(s)\}$;
end

Fig. 3. Main algorithm for CTL property checking against Kripke structures.

---

procedure *calcLabel*(**in** (S, S_0, R, L, AP) : KripkeStructure;
  in ϕ : CTL;
  inout label : $S \to 2^{\mathrm{CTL}}$)
begin
  if ϕ is a first-order formula **then**
    calcLabelFO$((S, S_0, R, L, AP), \phi,$ label);
  elseif $\phi = \phi_0 \wedge \phi_1$ **then**
    calcLabel$((S, S_0, R, L, AP), \phi_0,$ label);
    calcLabel$((S, S_0, R, L, AP), \phi_1,$ label);
    calcLabelAND$((S, S_0, R, L, AP), \phi_0, \phi_1,$ label);
  elseif $\phi = \phi_0 \vee \phi_1$ **then**
    calcLabel$((S, S_0, R, L, AP), \phi_0,$ label);
    calcLabel$((S, S_0, R, L, AP), \phi_1,$ label);
    calcLabelOR$((S, S_0, R, L, AP), \phi_0, \phi_1,$ label);
  elseif $\phi = \mathbf{EX}\phi_0$ **then**
    calcLabel$((S, S_0, R, L, AP), \phi_0,$ label);
    calcLabelEX$((S, S_0, R, L, AP), \phi_0,$ label);
  elseif $\phi = \mathbf{E}(\phi_0 \mathbf{U} \phi_1)$ **then**
    calcLabel$((S, S_0, R, L, AP), \phi_0,$ label);
    calcLabel$((S, S_0, R, L, AP), \phi_1,$ label);
    calcLabelEU$((S, S_0, R, L, AP), \phi_0, \phi_1,$ label);
  endif
end

Fig. 4. Label calculation – control algorithm driven by formula syntax.

```
procedure calcLabelFO(in (S, S_0, R, L, AP) : KripkeStructure;
                      in φ : First-order formula;
                      inout label : S → 2^CTL)
begin
  foreach s ∈ S do
      if s ⊨ φ then
        label(s) := label(s) ∪ {φ};
      endif
  enddo
end
```

Fig. 5. Algorithm for labelling states with first-order formulae.

```
procedure calcLabelOR(in (S, S_0, R, L, AP) : KripkeStructure;
                      in φ_0 : CTL;  in φ_1 : CTL;
                      inout label : S → 2^CTL)
begin
  foreach s ∈ S do
      if φ_0 ∈ label(s) ∨ φ_1 ∈ label(s) then
        label(s) := label(s) ∪ {φ_0 ∨ φ_1};
      endif
    enddo
end
```

Fig. 6. Algorithm for labelling states with $\phi_0 \vee \phi_1$) formulae.

```
procedure calcLabelAND(in (S, S_0, R, L, AP) : KripkeStructure;
                       in φ_0 : CTL;  in φ_1 : CTL;
                       inout label : S → 2^CTL)
begin
  foreach s ∈ S do
      if φ_0 ∈ label(s) ∧ φ_1 ∈ label(s) then
        label(s) := label(s) ∪ {φ_0 ∧ φ_1};
      endif
    enddo
end
```

Fig. 7. Algorithm for labelling states with $\phi_0 \wedge \phi_1$) formulae.

```
procedure calcLabelEX(in (S, S_0, R, L, AP) : KripkeStructure;
                      in φ : CTL;
                      inout label : S → 2^CTL)
begin
  foreach s ∈ S do
      if ∃ s' ∈ S : R(s, s') ∧ φ ∈ label(s') then
        label(s) := label(s) ∪ {EXφ};
      endif
  enddo
end
```

Fig. 8. Algorithm for labelling states with $\mathbf{EX}\phi$ formulae.

```
procedure calcLabelEU(in (S, S_0, R, L, AP) : KripkeStructure;
                      in φ_0 : CTL;  in φ_1 : CTL;
                      inout label : S → 2^CTL)
begin
  T := ⟨s ∈ S | φ_1 ∈ label(s)⟩;
  foreach s ∈ T do
    label(s) := label(s) ∪ {E(φ_0 U φ_1)};
  enddo
  while T ≠ ⟨⟩ do
    s := head(T);
    T := tail(T);
    foreach u ∈ {v ∈ S | R(v, s)} do
      if E(φ_0 U φ_1) ∉ label(u) ∧ φ_0 ∈ label(u) then
        label(u) := label(u) ∪ {E(φ_0 U φ_1)};
        T := T ⌢ ⟨u⟩;
      endif
    enddo
  enddo
end
```

Fig. 9. Algorithm for labelling states with $\mathbf{E}(\phi_0\mathbf{U}\phi_1)$ formulae.

Theorem 4. *Let $\pi \in S^* \cup S^\omega$ be a finite or infinite path and f an LTL formula specifying a rule 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 that the CTL model checking algorithms above find a witness for $\Phi(f)$.*

Proof. For infinite paths π the statement follows from Theorem 3, since the algorithms above are sound and complete for CTL model checking [7]. Analysis of algorithms calcLabelEX and calcLabelEU above shows that they can be applied to finite paths in a way that is consistent with the semantic LTL extension to finite paths specified in Section 3.3. □

4 Model Checking of IXL Configurations

4.1 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).

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

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 3.2, 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 is quite complicated, however, 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 5. 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 [14], this problem has been overcome by using existentially quantified LTL with rigid variables as introduced in [19]. 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 3.5. Therefore, we will now introduce sub-models of full configuration models where the problem of infeasible paths no longer occurs.

4.2 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 boundary 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 boundary 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. Note that for points and diamond crossings, several *dirX*-attributes can have value 1.
7. Every element is associated with Boolean attributes upA, upB, upC, upD ("upstream A, B, C, D"). For a given element s in a sub-model, $upA = 1$ if and only if there exists a predecessor element s' which is linked by its a-channel to s. For B, C, D, the attribute values are analogously defined.
8. Further auxiliary attributes are added to each sub-model state as described in Section 4.4 below.

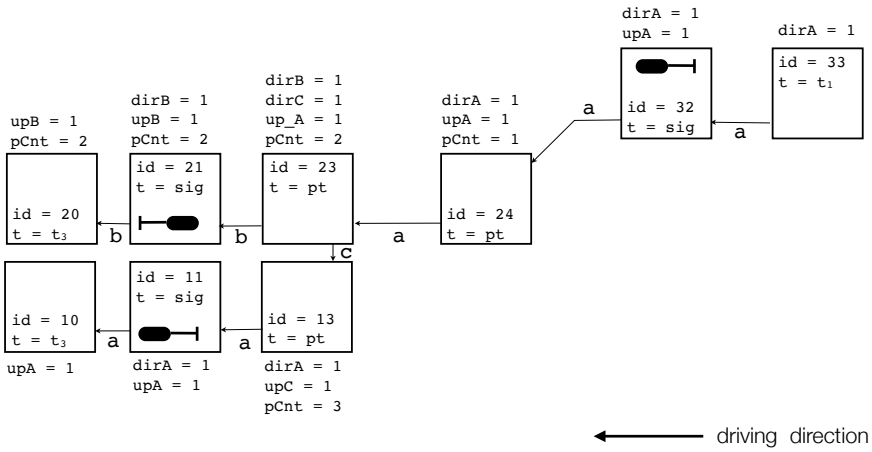

Fig. 10. Sub-model created from border element s_{33} in Fig. 1. Only attributes with positive value are shown.

Example 6. 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. 10, together with the new auxiliary attributes *dirA*, ... (the meaning of attribute *pCnt* is explained in Section 4.4 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. □

4.3 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 7. The rule violation specified in Example 1, when applied to a sub-model as the one depicted in Fig. 10, 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 3.5 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. 10 is the path $s_{32}.s_{24}.s_{23}.s_{21}.s_{20}$, and this is also a witness for ϕ_1, so in this example, the CTL over-approximation does not produce any false alarms in this case. □

Example 8. 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 3.5 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. 10, 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. □

The rule violations 4.1 and 4.2 associated with flank protection as described in Example 4 may be formalised as follows. In the formulas below, we use abbreviation

$$boundary \equiv (dirA + dirB + dirC + dirD = 0).$$

boundary evaluates to `true` if and only if the element is a boundary element, since they are the only ones without outgoing channels in driving direction.

Example 9. The rule violations 4.1 and 4.2 associated with flank protection as described in Example 4 may be formalised as follows.

$$\begin{aligned}\phi_{4.1} \equiv t = pt \wedge \mathrm{dirC} = 1 \wedge \mathbf{X}\big(upC = 1 \wedge \\ ((t \neq pt \vee dirA = 0) \wedge (t \neq sig \vee dirA = 1)) \\ \mathbf{U}(boundary \vee (t = pt \wedge dirA = 0))\big)\end{aligned}$$

Condition $\mathbf{X}upC = 1$ means that we are only interested in paths where the successor element of the point p_1 is connected to p_1's C-stem. The left operand of the **U**-operator specifies that no protecting points or signals are found. The right hand side of the **U**-operator specifies that we stop looking for suitable flank protection as soon as we have found a point offering no protection (this is equivalent to its a-channel pointing back towards p_1) or if the end of the route has been reached.

This LTL formula is translated via Φ defined in Section 3.5 into CTL formula

$$\begin{aligned}\Phi(\phi_{4.1}) \equiv t = pt \wedge \mathrm{dirC} = 1 \wedge \mathbf{EX}\big(upC = 1 \wedge \\ \mathbf{E}((t \neq pt \vee dirA = 0) \wedge (t \neq sig \vee dirA = 1)) \\ \mathbf{U}(boundary \vee (t = pt \wedge dirA = 0))\big)\end{aligned}$$

The formalisation of rule violation 4.2 (erroneous protection for driving directions AC/CA) is specified in LTL as follows. Example 4 may be formalised as follows.

$$\begin{aligned}\phi_{4.2} \equiv t = pt \wedge \mathrm{dirB} = 1 \wedge \mathbf{X}\big(upB = 1 \wedge \\ ((t \neq pt \vee dirA = 0) \wedge (t \neq sig \vee dirA = 1)) \\ \mathbf{U}(boundary \vee (t = pt \wedge dirA = 0))\big)\end{aligned}$$

and in translated form as

$$\begin{aligned}\Phi(\phi_{4.2}) \equiv t = pt \wedge \mathrm{dirB} = 1 \wedge \mathbf{EX}\big(upB = 1 \wedge \\ \mathbf{E}((t \neq pt \vee dirA = 0) \wedge (t \neq sig \vee dirA = 1)) \\ \mathbf{U}(boundary \vee (t = pt \wedge dirA = 0))\big)\end{aligned}$$

□

4.4 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.

Consider again the Example 3 in Section 2, where the number of elements of a certain type located between two reference elements needs to be counted. In principle, violation formulas associated with rules of that kind 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 [17]. 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. This is illustrated in Fig. 10.

Example 10. With auxiliary attribute *pCnt* at hand, the violation of Rule 3 from Example 3 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. 10. 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. □

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

4.5 Parallelisation

The concept to use sub-models for verifying DVL-queries allows for parallelisation of checking activities. The concurrent checker design is shown as a UML activity chart [21, Chapter 15] in Fig. 11. The concurrent checking process receives file names of the DVL-query and the IXL configuration model to be verified. After the query file has been successfully parsed (error exits are not shown in Fig. 11), action PrepareJobs identifies all jobs to be performed and places them into a queue. Each job consists of a triple (query,id,direction), where id is the identification of a boundary element. Attribute direction is A or B, depending on whether channel a or b of the boundary element is defined. After the job queue has been completely filled, the worker threads of a thread pool are activated (signal START).

Example 11. The sub-model depicted in Fig. 10 is identified by $id = 33$ and $direction = A$. □

The checker threads process these jobs concurrently, until the job queue is empty. Each thread (see Fig. 12) creates the sub-model identified by boundary element id and direction. After that, the CTL checker functions described in Section 3.6 are executed in action CTLChecker, and any witness found in the sub-model for the given query is written to the output interface.

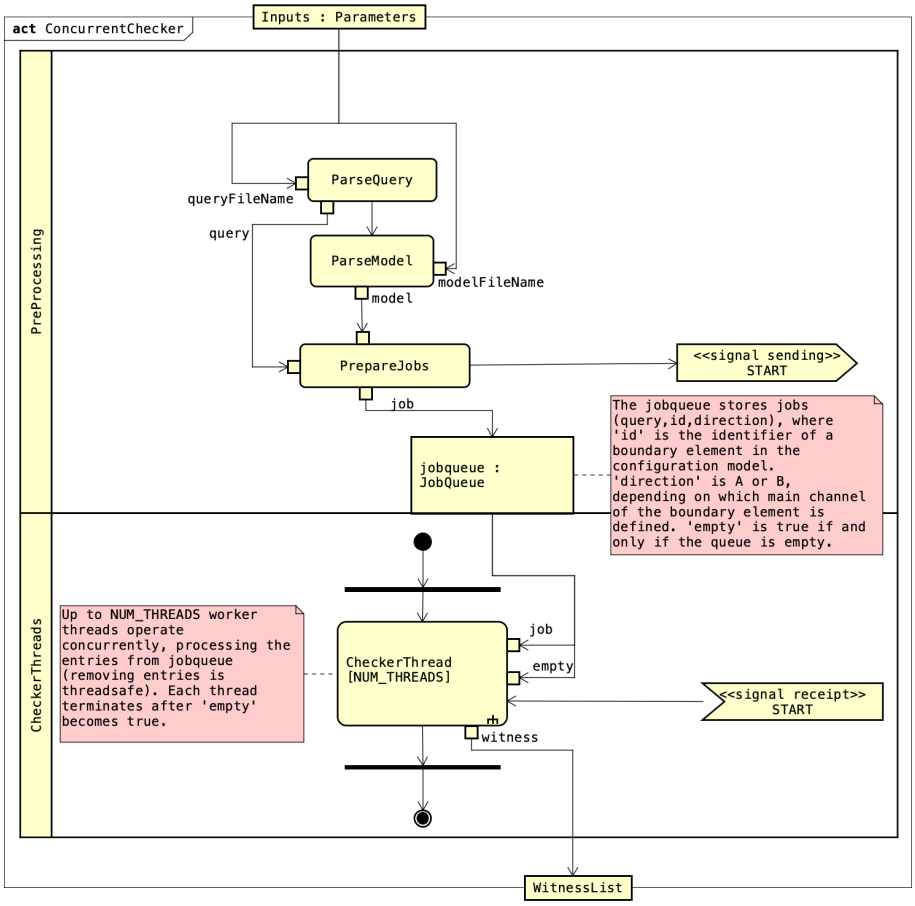

Fig. 11. Concurrent checker design.

4.6 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 [14] 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.

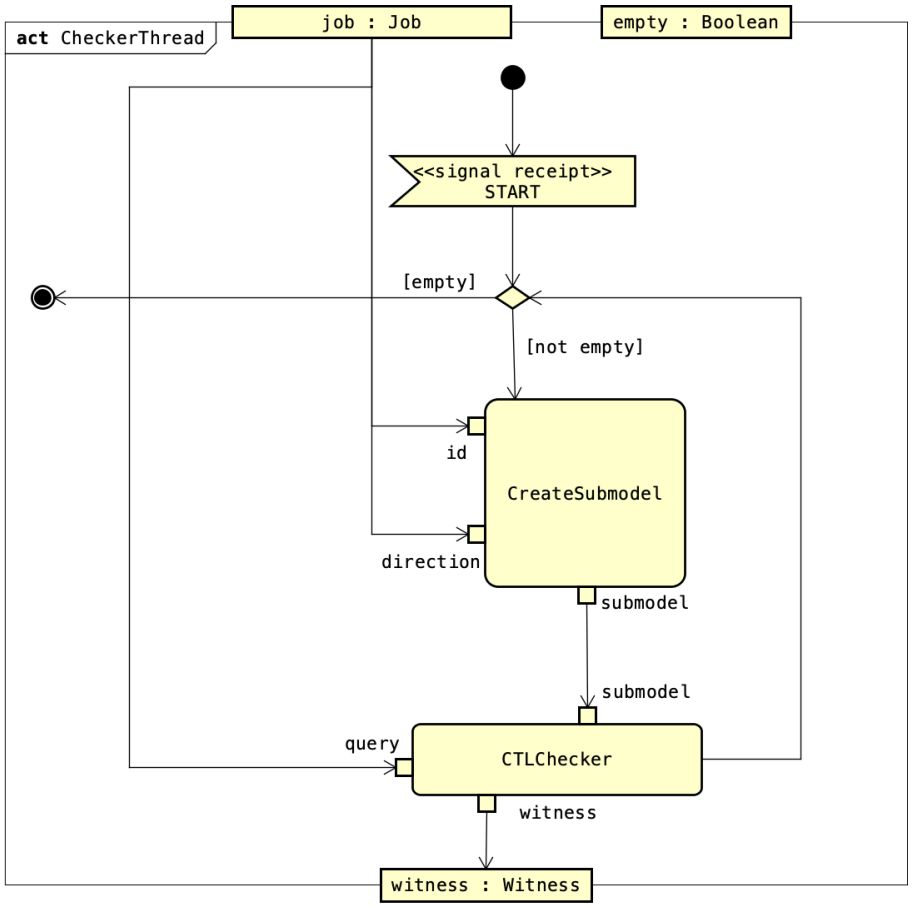

Fig. 12. Internal structure of a checker thread.

5 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 require a high degree of automation for checking their correctness. There seems to be 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 [1,14,13,11,16].

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 [5,16]. We agree, however, with [11] (and have state this, for example, in [15]), 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 to 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 [18,1,13,11,16] 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, our approach is based on temporal logic and model checking. Moreover, our methodology is strictly specialised on geographic 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 the 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 [13].

An general overview of trends in formal methods applications to railway signalling can be found in [4,9,2]. Many other research groups have been using model-checking for the behavioural verification of interlocking systems. In [10] 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 [24] Winter pushes the applicability bounds of symbolic model checking with NuSMV by optimising the ordering strategies for variables and transitions using domain knowledge about the track layout. Fantechi suggests in [8] 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 shrink the state space using abstraction techniques reducing the number of track sections and the number of trains. In [15], we have shown that bounded model checking in combination with k-induction can cope with the size of real-world route-based interlocking systems for verifying their behaviour. As an alternative to the B-family, the RAISE tool offers the possibility to perform combined verification by theorem proving and model checking [12].

6 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 interactively. 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 to check 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.

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. Bjørner, D.: 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)
5. 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
6. CENELEC: EN 50128:2011 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems (2011)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)
8. Fantechi, A.: Distributing the Challenge of Model Checking Interlocking Control Tables. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, Lecture Notes in Computer Science, vol. 7610, pp. 276–289. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-34032-1_26

9. Fantechi, A., Fokkink, W., Morzenti, A.: Some Trends in Formal Methods Applications to Railway Signaling. In: Formal Methods for Industrial Critical Systems, pp. 61–84. John Wiley & Sons, Inc. (2012), http://dx.doi.org/10.1002/9781118459898.ch4
10. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model Checking Interlocking Control Tables. In: Schnieder, E., Tarnai, G. (eds.) Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2010), Braunschweig, Germany. Springer (2011)
11. 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)
12. Geisler, S., Haxthausen, A.E.: Stepwise development and model checking of a distributed interlocking system - using RAISE. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E.P. (eds.) Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10951, pp. 277–293. Springer (2018), https://doi.org/10.1007/978-3-319-95582-7_16
13. 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)
14. 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)
15. Hong, L.V., Haxthausen, A.E., Peleska, J.: Formal modelling and verification of interlocking systems featuring sequential release. Sci. Comput. Program. 133, 91–115 (2017), http://dx.doi.org/10.1016/j.scico.2016.05.010
16. Keming, W., Zheng, W., Chuandong, Z.: Formal modeling and data validation of general railway interlocking system. WIT Transactions on The Built Environment 181 (2019)
17. 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
18. Lecomte, T., Burdy, L., Leuschel, M.: Formally checking large data sets in the railways. CoRR abs/1210.6815 (2012)
19. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems - specification. Springer (1992)
20. Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H.: Defining and Model Checking Abstractions of Complex Railway Models using CSP‖B. In: Biere, A., Nahir, A., Vos, T. (eds.) Hardware and Software: Verification and Testing. Lecture Notes in Computer Science, vol. 7857, pp. 193–208. Springer Berlin Heidelberg (2013)
21. Object Management Group: OMG Unified Modeling Language (OMG UML), superstructure, version 2.5.1. Tech. rep., OMG (2017)
22. Pachl, J.: Railway Operation and Control. VTD Rail Publishing (January 2002)
23. 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

24. Winter, K.: Symbolic Model Checking for Interlocking Systems. In: Railway Safety, Reliability and Security: Technologies and System Engineering, pp. 298–315. IGI Global (2012)