Rule-Based Consistency Checking of Railway Infrastructure Designs

Bjørnar Luteberget[1(✉)], Christian Johansen[2], and Martin Steffen[2]

[1] RailComplete AS (Formerly Anacon AS), Sandvika, Norway
bjlut@railcomplete.no

[2] Department of Informatics, University of Oslo, Oslo, Norway
{cristi,msteffen}@ifi.uio.no

Abstract. Railway designs deal with complex and large-scale, safety-critical infrastructures, where formal methods play an important role, especially in verifying the safety of so-called interlockings through model checking. Model checking deals with state change and rather complex properties, usually incurring considerable computational burden (chiefly in terms of memory, known as state-space explosion problem). In contrast to this, we focus on static infrastructure properties, based on design guidelines and heuristics. The purpose is to automate much of the manual work of the railway engineers through software that can do verification on-the-fly. In consequence, this paper describes the integration of formal methods into the railway design process, by formalizing relevant technical rules and expert knowledge. We employ a variant of Datalog and use the standardized "railway markup language" railML as basis and exchange format for the formalization. We describe a prototype tool and its (ongoing) integration in industrial railway CAD software, developed under the name RailCOMPLETE®. We apply this tool chain in a Norwegian railway project, the upgrade of the Arna railway station.

Keywords: Railway designs · Automation · Logic programming · Signalling · Railway infrastructure · railML · CAD · Datalog

1 Introduction

Railway systems are complex and large-scale, safety-critical infrastructures, with increasingly computerized components. The discipline of railway engineering is characterized by heavy national regulatory oversight, high and long-standing safety and engineering standards, a need for interoperability and (national and international) standardization. Due to the high safety requirements, the railway design norms and regulations recommend the use of formal methods (of various kinds), and for the higher safety integrity levels (SIL), they "highly recommend" them (cf. e.g. [4]). Railways require thoroughly designed control systems to ensure safety and efficient operation. The railway signals are used to direct traffic, and the *signalling component layout* of a train station is crucial to its traffic capacity. Another central part of a railway infrastructure is the so-called

E. Ábrahám and M. Huisman (Eds.): IFM 2016, LNCS 9681, pp. 491–507, 2016.
DOI: 10.1007/978-3-319-33693-0_31

interlocking, which refers, generally speaking, to the ensemble of systems tasked to establish safe, conflict-free routes of trains through stations (cf. [18]).

Railway construction projects are heavy processes that integrate various fields, engineering disciplines, different companies, stakeholders, and regulatory bodies. When working out railway designs a large part of the work is repetitive, involving routine checking of consistency with rules, writing tables, and coordinating disciplines. Many of these manual checks are simple enough to be automated.

With the purpose of increasing the degree of automation, we present results on integrating formal methods into the railway design process, as follows:

– We formalize rules governing track and signalling layout, and interlocking.
– The standardized "railway markup language" railML [19] is used as basis and exchange format for the formalization.
– We model the concepts describing a railway design in the logic of Datalog; and develop an automated generation of the model from the railML representation.
– We develop a prototype tool and integrate it in existing railway CAD software.

We illustrate the logical representation of signalling principles and show how they can be implemented and solved efficiently using the Datalog style of logic programming [21]. We also show the integration with existing railway engineering workflow by using CAD models directly. This enables us to verify rules continuously as the design process changes the station layout and interlocking. Based on railML [19], our results can be easily adopted by anyone who uses this international standard. The work uses as case study the software and the design (presently under development) used in the *Arna-Fløen* upgrade project,[1] with planned completion in 2020. The Arna train station is located on Northern Europe's busiest single-track connection, which is being extended to a double-track connection. The case study is part of an ongoing project in Anacon AS (now merged with Norconsult), a Norwegian signalling design consultancy. It is used to illustrate the approach, test the implementation, and to verify that the tool's performance is acceptable for interactive work within the CAD software.

The rest of the paper is organized as follows. Section 2 discusses aspects of the railway domain relevant for this work. Section 3 proposes a tool chain that extends CAD with formal representations of signalling layout and interlocking. Section 4 presents our formalization of the rules and concepts of railway design as logical formulas amenable for the Datalog implementation and checking. Section 5 provides information about the implementation, including details about counterexample presentation and empirical evaluation using the case study. We conclude in Sect. 6 with related and future work.

2 Background on the Railway Signalling Domain

The signalling design process results in a set of documents which can be categorized into (a) track and signalling component layout, and (b) interlocking specification.

[1] www.jernbaneverket.no/Prosjekter/prosjekter/Arna-Bergen.

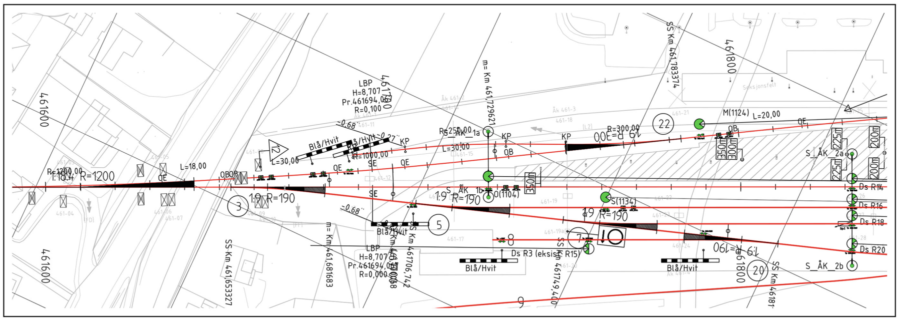


Fig. 1. Cut-out from 2D geographical CAD model (construction drawing) of preliminary design of the Arna station signalling (Color figure online).

Railway construction projects rely heavily on *computer aided design* (CAD) tools to map out railway station layouts. The various disciplines within a project, such as civil works, track works, signalling, or catenary power lines, work with *coordinated CAD models*. These CAD models contain a major part of the work performed by engineers, and are a collaboration tool for communication between disciplines. The signalling component layout is worked out by the signalling engineers as part of the design process. Signals, train detectors, derailers, etc., are drawn using symbols in a 2D geographical CAD model. An example of a layout drawing is given in Fig. 1.

2.1 Interlocking Specification

An interlocking is an interconnection of signals and switches to ensure that train movements are performed in a safe sequence [18]. Interlocking is performed electronically so that, e.g., a green light (or, more precisely, the *proceed aspect*) can only be lit under certain conditions. Conditions and state are built into the interlocking by relay-based circuitry or by computers running interlocking software. Most interlocking specifications use a *route-based tabular* approach, which means that a train station is divided into possible *routes*, which are paths that a train can take from one signal to another. These signals are called the *route entry signal* and *route exit signal*, respectively. An *elementary route* contains no other signals in-between. The main part of the interlocking specification is to tabulate all possible routes and set conditions for their use. Typical conditions are:

- *Switches* must be positioned to guide the train to a specified route exit signal.
- *Train detectors* must show that the route is free of any other trains.
- *Conflicting routes*, i.e. overlapping routes (or overlapping safety zones), must not be in use.

3 Proposed Railway Signalling Design Tool Chain

Next we describe shortly the tool chain that we propose for automating the current manual tasks involved in the design of railway infrastructures (see details in [15]). In particular, we are focused on integrating and automating those simple, yet tedious, rules and conditions usually used to maintain some form of consistency of the railway, and have these checks done automatically. Whenever the design is changed by an engineer working with the CAD program, our verification procedure would help, behind the scenes, verifying any small changes in the model and the output documents. Violations would either be automatically corrected, if possible, or highlighted to the engineer. Thus, we are focusing on solutions with small computational overhead.

3.1 Computer-Aided Design (CAD) Layout Model

CAD models, which ultimately correspond to a database of geometrical objects, are used in railway signalling engineering. They may be 2D or 3D, and contain mostly spatial properties and textual annotations, i.e., the CAD models focus on the *shapes* of objects and *where* to place them. The top level of the document, called the *model space block*, contains geometrical primitives, such as lines, circles, arcs, text, and symbols.

Geometric elements may represent the physical geometry directly, or symbolically, such as text or symbols. However, the verification of signalling and interlocking rules requires information about object properties and relations between objects such as which signals and signs are related to which track, and their identification, capabilities, and use. This information is better modelled by the railway-specific extensible hierarchical object model called railML [17].

3.2 Integrating railML and Interlocking Specifications with CAD Models

CAD programs were originally designed to produce paper drawings, and common practice in the use of CAD programs is to focus on *human-readable* documents. The database structure, however, may also be used to store machine-readable information. In the industry-standard DWG format, each geometrical object in the database has an associated *extension dictionary*, where add-on programs may store any data related to the object. Our tool uses this method to store the railML fragments associated with each geometrical object or symbol. Thus, we can compile the complete railML representation of the station from the CAD model.

Besides the layout, the design of a railway station consists also of a specification for the interlocking. This specification models the behavior of the signalling, and it is tightly linked to the station layout. A formal representation of the interlocking specification is embedded in the CAD document in a similar way as for the railML infrastructure data, using the document's global extension dictionary. Thus, the single CAD document showing the human-readable layout of the train station also contains a machine-readable model which fully describes both the component layout and the functional specification of the interlocking. This allows a full analysis of the operational aspects of the train station directly from a familiar editable CAD model.

3.3 Overall Tool Chain

Figure 2 shows the overall tool chain. The software allows checking of rules and regulations of static infrastructure (described in this paper) inside the CAD environment, while more comprehensive verification and quality assurance can be performed by special-purpose software for other design and analysis activities.

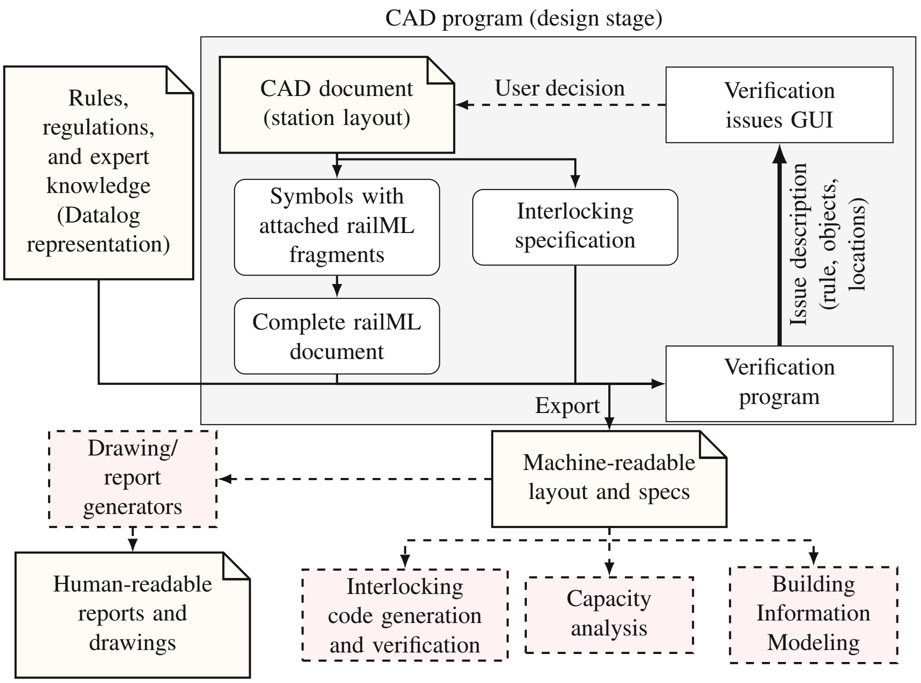

CAD program (design stage)
Rules, regulations, and expert knowledge (Datalog representation)
CAD document (station layout)
User decision
Verification issues GUI
Symbols with attached railML fragments
Interlocking specification
Issue description (rule, objects, locations)
Complete railML document
Verification program
Export
Drawing/ report generators
Machine-readable layout and specs
Human-readable reports and drawings
Interlocking code generation and verification
Capacity analysis
Building Information Modeling

Fig. 2. Railway design tool chain. The CAD program box shows features which are directly accessible at design time inside the CAD program, while the export creates machine-readable (or human-readable) documents which may be further analyzed and verified by external software (shown in dashed boxes).

Generally, analysis and verification tools for railway signalling designs can have complex inputs, they must account for a large variety of situations, and they usually require long running times. Therefore, we limit the verification inside the design environment to static rules and expert knowledge, as these rules require less dynamic information (timetables, rolling stock, etc.) and less computational effort, while still offering valuable insights. This situation may be compared to the tool chain for writing computer programs. Static analysis can be used at the detailed design stage (writing the code), but can only verify a limited set of properties. It cannot fully replace testing, simulation and other types of analysis, and must as such be seen as a part of a larger tool chain.

Other tools, that are external to the CAD environment, may be used for these more calculation heavy or less automated types of analysis, such as:

– Code generation and verification for interlockings, possible e.g. through the formal verification framework of Prover Technology.
– Capacity analysis and timetabling, performed e.g. using OpenTrack, LUKS, or Treno.
– Building Information Modeling (BIM), including such activities as life-cycle information management and 3D viewing, are already well integrated with CAD, and can be seen as an extension of CAD.

The transfer of data from the CAD design model to other tools is possible by using standardized formats such as railML, which in the future will also include an interlocking specification schema [3].

4 Formalization of Rule Checking

To achieve our goal of automating checking of the consistency of railway designs we need formal representations of both the designs and the consistency rules.

The logical representation of the designs (called the model) and of the rules (called properties) are fed into the verification engine (SAT/SMT or Datalog) which is doing satisfiability checking, thus looking for an interpretation of the logical variables that would satisfy the formulas. More precisely, the rules are first negated, then conjoined with the formulas representing the model. Therefore, looking for a satisfying interpretation is the same as looking for a way to violate the rules. When found, the interpretation contains the information about the exact reasons for the violation. The reasons, or counter-example, involves some of the negated rules as well as some parts of the model.

We formalize the correctness properties (i.e., technical rules and expert knowledge) as predicates over finite and real domains. Using a logic programming framework, we will include the following in the logical model:

1. Predicate representation of input document *facts*, i.e. track layout and interlocking.
2. Predicate representation of derived concept *rules*, such as object properties, topological properties, and calculation of distances.
3. Predicate representation of technical rules.

Each of these categories are described in more detail below, after we present the logical framework we employ.

4.1 Datalog

Declarative logic programming is a programming language paradigm which allows clean separation of logic (meaning) and computation (algorithm). This section gives a short overview of Datalog concepts. See [21] for more details. In its most basic form Datalog is a database query, as in the SQL language, over a finite set of atoms which can be combined using conjunctive queries, i.e. expressions in the fragment of first-order logic which includes only conjunctions and existential quantification.

Conjunctive queries alone, however, cannot express the properties needed to verify railway signalling. For example, given the layout of the station with tracks represented as edges between signalling equipment nodes, graph reachability queries are required to verify some of the rules. This corresponds to computing the transitive closure of the graph adjacency relation, which is not expressible in first-order logic [13, Chap. 3]. Adding fixed-point operators to conjunctive queries is a common way to mitigate the above problem while preserving decidability and polynomial time complexity.

The Datalog language is a first-order logic extended with *least fixed points.* We define the Datalog language as follows: *Terms* are either constants (atoms) or variables. *Literals* consist of a *predicate* P with a certain arity n, along with terms corresponding to the predicate arguments, forming an expression like $P(\vec{a})$, where $a = (a_1, a_2, \ldots, a_n)$. *Clauses* consist of a *head* literal and one or more *body* literals, such that all variables in the head also appear in the body. Clauses are written as

$$R_0(\vec{x}) \;:-\; \exists \vec{y} : R_1(\vec{x}, \vec{y}), R_2(\vec{x}, \vec{y}), \ldots, R_k(\vec{x}, \vec{y}).$$

Datalog uses the Prolog convention of interpreting identifiers starting with a capital letter as variables, and other identifiers as constants. E.g., the meaning of the clause $a(X, Y) :- b(X, Z), c(Z, Y)$ is $\forall x, y : ((\exists z : (b(x, z) \wedge c(z, y))) \rightarrow a(x, y))$.

Clauses without body are called *facts*, those with one or more literals in the body are called *rules.* No nesting of literals is allowed. However, recursive definitions of predicates are possible. In the railway domain, this can be used to define the *connected* predicate, which defines whether two objects are connected by railway tracks:

$$\begin{aligned} \mathit{directlyConnected}(a, b) &:- \mathit{track}(t), \mathit{belongsTo}(a, t), \mathit{belongsTo}(b, t). \\ \mathit{connected}(a, b) &:- \mathit{directlyConnected}(a, b). \\ \mathit{connected}(a, b) &:- \mathit{directlyConnected}(a, x), \mathit{connection}(x, c), \\ & \mathit{connected}(c, b). \end{aligned}$$

Here, the *connection* predicate contains switches and other connection types. Further details of relevant predicates are given in the sections below.

Another common feature of Datalog implementations is to allow negation, with *negation as failure* semantics. This means that negation of predicates in rules is allowed with the interpretation that when the satisfiability procedure cannot find a model, the statement is false. To ensure termination and unique solutions, the negation of predicates must have a *stratification*, i.e. the dependency graph of negated predicates must have a topological ordering (see [21, Chap. 3] for details).

Datalog is sufficiently expressive to describe static rules of signalling layout topology and interlocking. For geometrical properties, it is necessary to take sums and differences of lengths, which requires extending Datalog with arithmetic operations. A more expressive language is required to cover all aspects of railway design, e.g. capacity analysis and software verification, but for the properties in the scope of this paper, a concise, restricted language which ensures termination and short running times has the advantage of allowing tight integration with the existing engineering workflow.

4.2 Input Documents Representation

Track and Signalling Objects Layout in the railML Format. Given a complete railML infrastructure document, we consider the set of XML elements in it that correspond to identifiable objects (this is the set of elements which inherit properties from the type `tElementWithIDAndName`). The set of all IDs which are assigned to XML elements form the finite domain of constants on which we base our predicates (IDs are assumed unique in railML).

$$\text{Atoms} := \{a \mid \text{element}.\texttt{ID} = a\}\,.$$

We denote a railML element with $ID = a$ as element_a. All other data associated with an element is expressed as predicates with its identifying atom as one of the arguments, most notably the following:

– Element type (also called class in railML/XML):

$$\begin{aligned} \mathit{track}(a) &\leftarrow \text{element}_a \text{ is of type } \texttt{track}, \\ \mathit{signal}(a) &\leftarrow \text{element}_a \text{ is of type } \texttt{signal}, \\ \mathit{balise}(a) &\leftarrow \text{element}_a \text{ is of type } \texttt{balise}, \\ \mathit{switch}(a) &\leftarrow \text{element}_a \text{ is of type } \texttt{switch}. \end{aligned}$$

– Position and absolute position (elements inheriting from `tPlacedElement`):

$$\begin{aligned} \mathit{pos}(a,p) &\leftarrow (\text{element}_a.\texttt{pos} = p), \quad a \in \text{Atoms}, p \in \mathbb{R}, \\ \mathit{absPos}(a,p) &\leftarrow (\text{element}_a.\texttt{absPos} = p), \quad a \in \text{Atoms}, p \in \mathbb{R}. \end{aligned}$$

– Direction (for elements inheriting from `tOrientedElement`):

$$\mathit{dir}(a,d) \leftarrow (\text{element}_a.\texttt{dir} = d), \quad a \in \text{Atoms}, d \in \text{Direction},$$

where $\text{Direction} = \{\mathit{up}, \mathit{down}, \mathit{both}, \mathit{unknown}\}$, indicating whether the object is visible or functional in only one of the two possible travel directions, or both.

– Signal properties (for elements of type `tSignal`):

$$\begin{aligned}&\mathit{signalType}(a,t) \leftarrow (\text{element}_a.\texttt{type} = t), t \in \{\text{main, distant, shunting, combined}\}\,,\\ &\mathit{signalFunction}(a,f) \leftarrow (\text{element}_a.\texttt{function} = f),\\ &\qquad\qquad a \in \text{Atoms}, f \in \{\text{home, intermediate, exit, blocking}\}\,.\end{aligned}$$

The *switch* element is the object which connects tracks with each other and creates the branching of paths. A switch belongs to a single track, but contains *connection* sub-elements which point to other connection elements, which are in turn contained in switches, crossings or track ends. For connections, we have the following predicates:

– Connection element and reference:

$$\begin{aligned}\mathit{connection}(a) &\leftarrow \text{element}_a \text{ is of type } \texttt{connection},\\ \mathit{connection}(a,b) &\leftarrow (\text{element}_a.\texttt{ref} = b).\end{aligned}$$

– Connection course and orientation:

$$\begin{aligned}\mathit{connectionCourse}(a,c) &\leftarrow (\text{element}_a.\texttt{course} = c), c \in \{\text{left, straight, right}\}\\ \mathit{connectionOrientation}(a,o) &\leftarrow (\text{element}_a.\texttt{orientation} = o),\\ &\qquad a \in \text{Atoms}, o \in \{\text{outgoing, incoming}\}\,.\end{aligned}$$

To encode the hierarchical structure of the railML document, a separate predicate encoding the parent/child relationship is added:

– Object belongs to (e.g. a is a signal belonging to track b):

$$\begin{aligned}\mathit{belongsTo}(a,b) \leftarrow\ & b \text{ is the closest XML ancestor of } a \text{ whose element}\\ & \text{type inherits from } \texttt{tElementWithIDAndName}.\end{aligned}$$

Interlocking. An XML schema for tabular interlocking specifications is described in [3], and this format is used here with, anticipating that it will become part of the railML standard schema in the future. We give some examples of how XML files with this schema are translated into predicate form:

– Train route with given direction d, start point a, and end point b ($a, b \in$ Atoms, $d \in$ Direction):

$$\begin{aligned}\mathit{trainRoute}(t) &\leftarrow \text{element}_t \text{ is of type } \texttt{route}\\ \mathit{start}(t,a) &\leftarrow (\text{element}_t.\texttt{start} = a)\\ \mathit{end}(t,b) &\leftarrow (\text{element}_t.\texttt{end} = b)\end{aligned}$$

– Conditions on detection section free (a) and switch position (s, p):

$$\begin{aligned}\mathit{detectionSectionCondition}(t,a) &\leftarrow (a \in \text{element}_t.\texttt{sectionConditions}),\\ \mathit{switchPositionCondition}(t,s,p) &\leftarrow ((s,p) \in \text{element}_t.\texttt{switchConditions}).\end{aligned}$$

4.3 Derived Concepts Representation

Derived concepts are properties of the railway model which can be defined independently of the specific station. A library of these predicates is needed to allow concise expression of the rules to be checked.

Object Properties. Properties related to specific object types which are not explicitly represented in the layout description, such as whether a switch is *facing* in a given direction, i.e. if the path will branch when you pass it:

$$\begin{aligned} \mathit{switchFacing}(a,d) \leftarrow \exists c,o : \mathit{switch}(a) \wedge \mathit{switchConnection}(a,c) \wedge \\ \mathit{switchOrientation}(c,o) \wedge \mathit{orientationDirection}(o,d). \end{aligned}$$

Topological and Geometric Layout Properties. Predicates describing the topological configuration of signalling objects and the train travel distance between them are described by predicates for **track connection** (predicate $\mathit{connected}(a,b)$), **directed connection** (predicate $\mathit{following}(a,b,d)$), **distance** (predicate $\mathit{distance}(a,b,d,l)$), etc. The track connection predicate is defined as:

$$\mathit{directlyConnected}(a,b) \leftarrow \exists t : \mathit{track}(t) \wedge \mathit{belongsTo}(a,t) \wedge \mathit{belongsTo}(b,t),$$

$$\begin{aligned} \mathit{connected}(a,b) \leftarrow \mathit{directlyConnected}(a,b) \vee (\exists c_1, c_2 : \mathit{connection}(c_1,c_2) \wedge \\ \mathit{directlyConnected}(a,c_1) \wedge \mathit{connected}(c_2,b)). \end{aligned}$$

Interlocking Properties. Properties such as $\mathit{existsPathWithoutSignal}(a,b)$ for finding elementary routes, and $\mathit{existsPathWithDetector}(a,b)$ for finding adjacent train detectors will be used as building blocks for the interlocking rules.

4.4 Rule Violations Representation

With the input documents represented as facts, and a library of derived concepts, it remains to define the technical rules to be checked. Technical rules are based on [11]. Some examples of technical rules representing conditions of the railway station layout are given below. More details can be found in the technical report [16].

Property 1 (Layout: Home signal [11]). *A* home main signal *shall be placed at least 200 m in front of the first controlled, facing switch in the entry train path.*

See also Fig. 3 for an example. Property 1 may be represented in the following way:

$$\begin{aligned} \mathit{isFirstFacingSwitch}(b,s) \leftarrow \mathit{stationBoundary}(b) \wedge \mathit{facingSwitch}(s) \wedge \\ \neg(\exists x : \mathit{facingSwitch}(x) \wedge \mathit{between}(b,x,s)), \end{aligned}$$

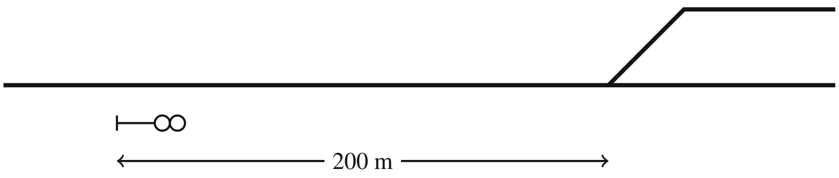


Fig. 3. A home main signal shall be placed at least 200 m in front of the first controlled, facing switch in the entry train path. (Property 1)

$$\begin{aligned} ruleViolation_1(b,s) \leftarrow\ & isFirstFacingSwitch(b,s) \wedge \\ & (\neg(\exists x : signalFunction(x, \text{home}) \wedge between(b,x,s)) \vee \\ & (\exists x,d,l : signalFunction(x,\text{home}) \wedge \\ & \wedge\ distance(x,s,d,l) \wedge l < 200). \end{aligned}$$

Checking for rule violations can be expressed as:

$$\exists b, s : ruleViolation_1(b,s),$$

which in Datalog query format becomes `ruleViolation1(B,S)?`.

Property 2 (Layout: Exit main signal [11]). *An exit main signal shall be used to signal movement exiting a station.*

This property can be elaborated into the following rules:

– No path should have more than one exit signal:

$$\begin{aligned} ruleViolation_2(s) \leftarrow & \exists d : signalType(s,\text{exit}) \wedge following(s,s_o,d) \wedge \\ & \neg signalType(s_0,\text{exit}). \end{aligned}$$

– Station boundaries should be preceded by an exit signal:

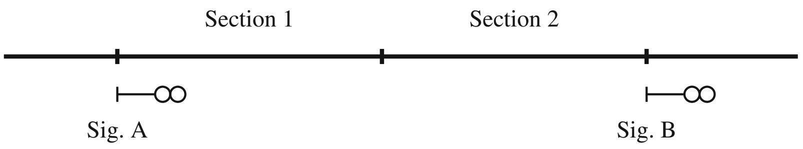
$$\begin{aligned} exitSignalBefore(x,d) \leftarrow & \exists s : signalType(s,\text{exit}) \wedge following(s,x,d) \\ ruleViolation_2(b) \leftarrow & \exists d : stationBoundary(b) \wedge \neg exitSignalBefore(b,d). \end{aligned}$$

Property 3 (Interlocking: Track clear on route). *Each pair of adjacent train detectors defines a track detection section. For any track detection sections overlapping the route path, there shall exist a corresponding condition on the activation of the route.*

See Fig. 4 for an example. Property 3 can be represented as follows:

$$\begin{aligned} adjacentDetectors(a,b) \leftarrow & trainDetector(a) \wedge trainDetector(b) \wedge \\ & \neg existsPathWithDetector(a,b), \end{aligned}$$

$$\begin{aligned} detectionSectionOverlapsRoute(r,d_a,d_b) \leftarrow\ & trainRoute(r) \wedge \\ & start(r,s_a) \wedge end(r,s_b) \wedge \\ & adjacentDetectors(d_a,d_b) \wedge overlap(s_a,s_b,d_a,d_b), \end{aligned}$$



Tabular interlocking:

Route	Start	End	Sections must be clear
AB	A	B	1, 2

Fig. 4. Track sections which overlap a route must have a corresponding condition in the interlocking. (Property 3)

$$\begin{aligned} \mathit{ruleViolation}_3(r, d_a, d_b) \leftarrow\ & \mathit{detectionSectionOverlapsRoute}(r, d_a, d_b) \wedge \\ & \neg \mathit{detectionSectionCondition}(r, d_a, d_b). \end{aligned}$$

Property 4 (Interlocking: Flank protection [11]). *A train route shall have flank protection.*

For each switch in the route path and its associated position, the paths starting in the opposite switch position defines the *flank*. Each flank path is terminated by the first flank protection object encountered along the path. An example situation is shown in Fig. 5. While the indicated route is active (A to B), switch X needs flank protection for its left track. Flank protection is given by setting switch Y in right position and setting signal C to *stop*. Property 4 can be elaborated into the following rules:

– All flank protection objects should be eligible flank protection objects, i.e. they should be in the list of possible flank protection objects, and have the correct orientation (the *flankElement* predicate contains the interlocking facts):

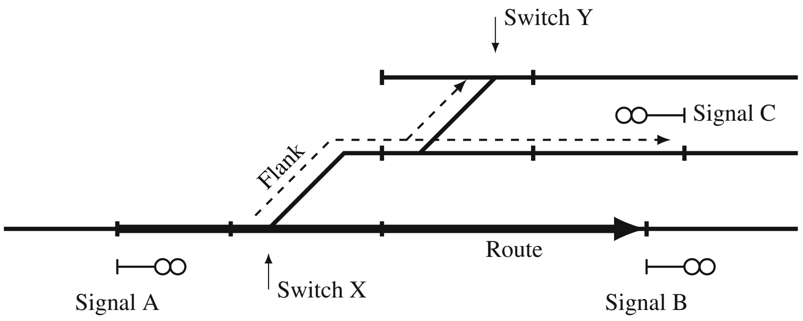


Fig. 5. The dashed path starting in switch X must be terminated in all branches by a valid flank protection object, in this case switch Y and signal C. (Property 4)

$$\begin{aligned}\mathit{flankProtectionObject}(a,b,d) \leftarrow & ((\mathit{signalType}(a,\text{main}) \wedge \mathit{dir}(a,d))\vee \\ & (\mathit{signalType}(a,\text{shunting}) \wedge \mathit{dir}(a,d))\vee \\ & \mathit{switchFacing}(a,d)\vee \\ & \mathit{derailer}(a)) \wedge \mathit{following}(a,b,d).\end{aligned}$$

$$\begin{aligned}\mathit{flankProtectionRequired}(r,x,d) \leftarrow\ & \mathit{trainRoute}(r) \wedge \mathit{start}(r,s_a)\wedge \\ & \mathit{end}(r,s_b) \wedge \mathit{switchOrientation}(x,o) \wedge \mathit{between}(s_a,x,s_b)\wedge \\ & \mathit{orientationDirection}(o,o_d) \wedge \mathit{oppositeDirection}(o_d,d).\end{aligned}$$

$$\begin{aligned}\mathit{flankProtection}(r,e) \leftarrow & \mathit{flankProtectionRequired}(r,x,d)\wedge \\ & \mathit{flankProtectionObject}(e,x,d).\end{aligned}$$

$$\begin{aligned}\mathit{ruleViolation}_4(r,e) \leftarrow & \mathit{flankElement}(r,e)\wedge \\ & \neg \mathit{flankProtection}(r,e).\end{aligned}$$

– There should be no path from a model/station boundary to the given switch, in the given direction, that does not pass a flank protection object for the route:

$$\begin{aligned}\mathit{ruleViolation}_4(r,b,x) \leftarrow\ & \mathit{stationBoundary}(b)\wedge \\ & \mathit{flankProtectionRequired}(r,x,d) \wedge \mathit{following}(b,x,d)\wedge \\ & \mathit{existsPathWithoutFlankProtection}(r,b,x,d).\end{aligned}$$

5 Tool Implementation

The XSB Prolog interpreter was used as a back-end for the implementation as it offers tabled predicates which have the same characteristics as Datalog programs [20], while still allowing general Prolog expressions such as arithmetic operations.

5.1 Counterexample Presentation

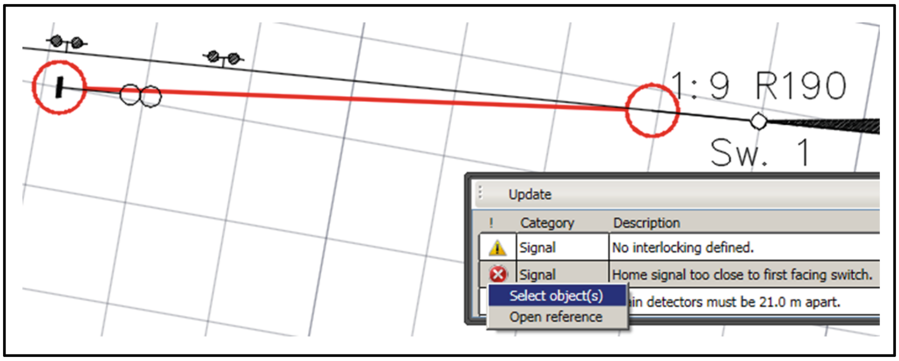
When rule violations are found, the railway engineer will benefit from information about the following:

– Which rule was violated (textual message containing a reference to the source of the rule or a justification in the case of expert knowledge rules).
– Where the rule was violated (identity of objects involved).

Also, classification of rules based on e.g. *discipline* and *severity* may be useful in many cases. In the rule databases, this may be accomplished through the use of *structured comments*, similar to the common practice of including structured documentation in computer programs, such as JavaDoc (see Fig. 6 for an example). A program parses the structured comments and forwards corresponding queries to the logic programming solver. Any violations returned are associated with the information in the comments, so that the combination can be used to present a helpful message to the user. A prototype CAD add-on program for Autodesk AutoCAD was implemented, see Fig. 7.

```
%| rule: Home signal too close to first facing switch.
%| type: technical
%| severity: error
homeSignalBeforeFacingSwitchError(S,SW) :-
    firstFacingSwitch(B,SW,DIR),
    homeSignalBetween(S,B,SW),
    distance(S,SW,DIR,L), L < 200.
```

Fig. 6. Structured comments on rule violation expression



Update

!	Category	Description
⚠	Signal	No interlocking defined.
⊗	Signal	Home signal too close to first facing switch.
		in detectors must be 21.0 m apart.

Fig. 7. Counterexample presentation within an interactive CAD environment.

5.2 Case Study Results

The rules concerning signalling layout and interlocking from *Jernbaneverket* [11] described above were checked in the railML representation of the Arna-Fløen project, which is an ongoing design project in Anacon AS (now merged with Norconsult). Each object was associated with one or more construction phases, which we call phase A and phase B, which also correspond to two operational phases. The model that was used for the work with the Arna station (phase A and B combined) included 25 switches, 55 connections, 74 train detectors, and 74 signals. The interlocking consisted of 23 and 42 elementary routes in operational phase A and B respectively.

Table 1. Case study size and running times on a standard laptop.

	Testing station	Arna phase A	Arna phase B
Relevant components	15	152	231
Interlocking routes	2	23	42
Datalog facts	85	8283	9159
Running time (s)	0.1	4.4	9.4

The Arna station design project and the corresponding CAD model has been in progress since 2013, and the method of integrating railML fragments into the CAD database, as described in Sect. 3, has been in use for about one year. Engineers working on this model are now routinely adding the required railML properties to the signalling components as part of their CAD modelling process. The rule collection consisted of 37 derived concepts, 5 consistency predicates, and 8 technical predicates. Running times for the verification procedure can be found in Table 1.

6 Conclusions, Related and Further Work

We have demonstrated a logical formalism in which railway layout and interlocking constraints and technical rules may be expressed, and which can be decided by logic programming proof methods with polynomial time complexity.

Related Work. Railway control systems and signalling designs are a fertile ground for formal methods. See [1,7] for an overview of various approaches and pointers to the literature, applying formal methods in railway design. In particular, safety of interlockings has been intensively formalized and studied, using for instance VDM [9] and the B-method, resp. Event-B [12]. Model checking has proved particularly attractive for tackling the safety of interlocking, and various model checkers and temporal logics have been used, cf. e.g. [5,6,22]. Critically evaluating practicality, [8] investigated applicability of model checking for interlocking tables using NuSMVresp. Spin. The research shows that interlocking systems of realistic size are currently out of reach for both flavors of model checkers. [10] uses *bounded* model checking for interlockings. An influential technology is the verified code generation for interlockings from Prover AB Sweden [2]. Prover is an automated theorem prover, using Stålmarck's method.

The mentioned works generally include *dynamic* aspects of the railway in their checking, like train positions and the interlocking state. This is in contrast to our work, which focuses on checking against static aspects. Lodemann et al. [14] use semantic technologies to automate railway infrastructure verification. Their scope is still wider than this paper in the computational sense, with the full expressive power of OWL ontologies, running times on the order of hours, and the use of separate interactive graphical user interfaces rather than integration with design tools.

Future Work. In the future work with RailComplete AS, we will focus on extending the rule base to contain more relevant signalling and interlocking rules from [11], evaluating the performance of our verification on a larger scale. Design information and rules about other railway control systems, such as geographical interlockings and train protection systems could also be included. The current work is assuming Norwegian regulations, but the European Rail Traffic Management System is expected to dominate in the future.

Finally, we plan to extend from consistency checking to optimization of designs. Optimization requires significantly larger computational effort, and the relation between Datalog and more expressive logical programming frameworks could become relevant.

Acknowledgments. We thank Anacon AS and RailComplete AS, especially senior engineer Claus Feyling, for guidance on railway methodology and philosophy. We acknowledge the support of the Norwegian Research Council through the project RailCons – Automated Methods and Tools for Ensuring Consistency of Railway Designs.

References

1. Bjørner, D.: New results and trends in formal techniques for the development of software in transportation systems. In: Proceedings of the Symposium on Formal Methods for Railway Operation and Control Systems (FORMS 2003). L'Harmattan Hongrie (2003)
2. Borälv, A., Stålmarck, G.: Formal verification in railways. In: Hinchey, M.G., Bowen, J.P. (eds.) Industrial-Strength Formal Methods in Practice. FACIT, pp. 329–350. Springer, London (1999)
3. Bosschaart, M., Quaglietta, E., Janssen, B., Goverde, R.M.P.: Efficient formalization of railway interlocking data in RailML. Inf. Syst. **49**, 126–141 (2015)
4. Boulanger, J.-L.: CENELEC 50128 and IEC 62279 Standards. Wiley-ISTE, New Jersey (2015)
5. Busard, S., Cappart, Q., Limbrée, C., Pecheur, C., Schaus, P.: Verification of railway interlocking systems. In: 4th Internationl Workshop on Engineering Safety and Security Systems (ESSS), vol. 184 of EPTCS, pp. 19–31 (2015)
6. Eisner, C.: Using symbolic model checking to verify the railway stations of hoorn-kersenboogerd and heerhugowaard. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 97–109. Springer, Heidelberg (1999)
7. Fantechi, A., Fokkink, W., Morzenti, A.: Some trends in formal methods applications to railway signalling. In: Gnesi, S., Margaria, T. (eds.) Formal Methods for Industrial Critical Systems, pp. 61–84. Wiley, New Jersey (2012)
8. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model checking interlocking control tables. In: Schnieder, E., Tarnai, G. (eds.) FORMS/FORMAT 2010, pp. 107–115. Springer, Heidelberg (2010)
9. Fukuda, M., Hirao, Y., Ogino, T.: VDM specification of an interlocking system and a simulator for its validation. In: 9th IFAC Symposium Control in Transportation Systems Proceedings, vol. 1, pp. 218–223, Braunschweig. IFAC (2000)
10. Haxthausen, A.E., Peleska, J., Pinger, R.: Applied bounded model checking for interlocking system designs. In: Counsell, S., Núñez, M. (eds.) SEFM 2013. LNCS, vol. 8368, pp. 205–220. Springer, Heidelberg (2014)
11. Jernbaneverket. Teknisk regelverk (2015). http://trv.jbv.no/
12. Lecomte, T., Burdy, L., Leuschel, M.: Formally checking large data sets in the railways. In: Proceedings of DS-Event-B 2012: Advances in Developing Dependable Systems in Event-B. In conjunction with ICFEM, 2012, vol. 3(1), pp. 35–43 (2012)
13. Libkin, L.: Elements of Finite Model Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2004)

14. Lodemann, M., Luttenberger, N., Schulz, E.: Semantic computing for railway infrastructure verification. In: IEEE Seventh International Conference on Semantic, Computing, pp. 371–376 (2013)
15. Luteberget, B., Feyling, C.: Automated verification of rules and regulations compliance in CAD models of railway signalling and interlocking. In: Computers in Railways XV. WIT Press (2016) (to appear)
16. Luteberget, B., Johansen, C., Steffen, M.: Rule-based consistency checking of railway infrastructure designs. Technical report 450, January 2016
17. Nash, A., Huerlimann, D., Schütte, J., Krauss, V.P.: RailML - a standard data interface for railroad applications. In: Allan, J., Hill, R.J., Brebbia, C.A., Sciutto, G., Sone, S. (eds.) Computers in Railways IX, pp. 233–240. WIT Press, Southampton (2004)
18. Pachl, J.: Railway Operation and Control. VTD Rail Publishing, Mountlake Terrace (2015)
19. RailML. The XML interface for railway applications (2016). http://www.railml.org
20. Swift, T., Warren, D.S.: XSB: extending prolog with tabled logic programming. Theor. Pract. Log. Program. **12**(1–2), 157–187 (2012)
21. Ullman, J.D.: Principles of Database and Knowledge-Base Systems. CSPP, New York (1988)
22. Winter, K., Johnston, W., Robinson, P., Strooper, P., van den Berg, L.: Tool support for checking railway interlocking designs. In: Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software, pp. 101–107 (2006)