This paper is part of the Proceedings of the 15th International Conference on Railway Engineering Design and Operation (CR 2016)
www.witconferences.com

Automated verification of rules and regulations compliance in CAD models of railway signalling and interlocking

B. Luteberget & C. Feyling
RailComplete AS, Oslo, Norway

Abstract

The design of signalling and interlocking for railway stations can require large efforts in checking consistency with rules, regulations and expert knowledge. This obstacle can slow down projects and hinder design iterations. Much of this manual effort can be automated. We suggest a software architecture based on familiar CAD practices where railML documents and other machine-readable railway representations are embedded in CAD database elements, and an automated rule-based verification software verifies the consistency of the design with respect to the relevant rules. The theory and tools of logic programming and knowledge-base systems are used for this purpose. The verification is performed directly on a CAD model from within the CAD program user interface, and inconsistencies (rule violations) are presented to the user by highlighting the relevant objects in the CAD model. This CAD integration can greatly increase the usage of automated verification during the design process because it is performed behind the scenes and requires very little of the user. This approach also facilitates data exchange with other types of analysis tools with much less effort than the traditional process, by exporting machine-readable data formats from a CAD model. The method has been tested on a real-world design project in Norway. The performance of the implementation is promising, and the feasibility for on-the-fly rule verification on larger railway stations is discussed together with several possible ways to improve the running times. The implementation is integrated with the RailCOMPLETE® software.

Keywords: railway infrastructure, signalling, interlocking, CAD, verification, automation, logic programming, Datalog.

WIT Transactions on The Built Environment, Vol 162, © 2016 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)
doi:10.2495/CR160151

1 Introduction

Railway engineering, and especially the signalling discipline, is characterised by heavy national regulatory oversight, high and long-standing safety and engineering standards, a need for interoperability and (national and international) standardisation. 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. [1]).

In the signalling design phase, however, there has been less effort on automation and verification of safety, regulation conformance, and quality. Tool support has traditionally been focused on the civil engineering and high-level planning and capacity aspects. There has also been extensive research on formal verification of interlockings (see e.g. [2–4]). However, these methods are computationally heavy and also depend on low-level implementation details, and are as such not currently suited for tight integration with interactive design tools.

During the signalling design, even small changes in station layout and interlocking may require thorough (re-)investigation to prove that the designs remain compliant with relevant rules and regulations. Many of these manual checks are simple enough to be automated with little computational effort, so that the verification can be performed inside existing engineering software, while the model is being created. We describe here a method for automatic verification of infrastructure manager rules for signalling and interlocking, implemented in a logic programming environment and tightly integrated with currently-used CAD tools.

Efficiency of designing railway construction projects, in all stages of planning, can be increased by working with machine-readable models and getting constant verification feedback with administration-specific configuration. Export from a CAD model to other software, such as capacity and timetable analysis software (OpenTrack, LUKS, etc.) and interlocking code generation and verification software (Prover Technology), can also increase the efficiency and usage of these external tools. Our architecture would also complement these tools and simplify their use by exporting machine-readable data which can be used directly.

This work is integrated in the RailCOMPLETE® software and the ongoing design of the *Arna-Fløen* upgrade project, a major infrastructure activity of the Norwegian railway system, with planned completion in 2020. The Arna train station is located on Northern Europe's busiest single-track connection (between Arna and Bergen), which is being extended to a double-track connection. 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.

Some research effort has gone into verification with similar scope as described in this paper. Lodemann *et al*. [5] use semantic technologies, specifically Semantic Constraints, as a means to automate railway infrastructure verification. Their scope is 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 interactive

graphical user interfaces where the user is specifically "performing verification", rather than integrating verification with other design tools.

2 Overall tool-chain

First, we describe the tool chain that we propose for automating the current manual tasks involved in the design of railway infrastructures. In particular, we are focused on integrating and automating the simple, yet tedious, rules and conditions that are used to check that a railway design is internally consistent, and that it adheres to regulations. 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 when working with CAD tools (running on standard computers).

Generally, analysis and verification tools for railway signalling designs can have complex inputs, must account for a large variety of situations, and usually

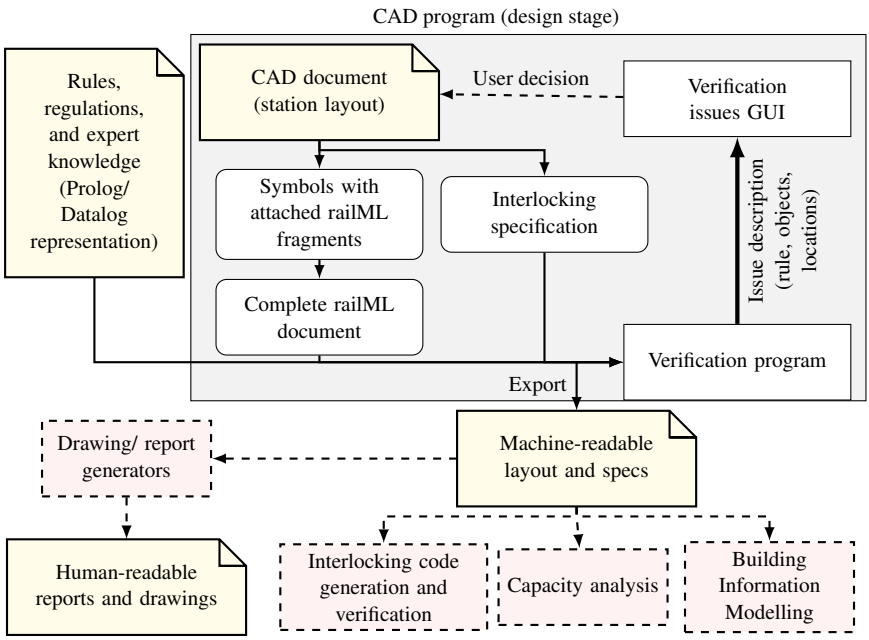

Figure 1: Railway design tool chain. The CAD program box shows verification features which are directly accessible at design time inside the CAD program, while the export creates machine-readable (or human-readable) documents which may be further analysed and verified by external software (shown in dashed boxes).

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 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 considerations. 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 types of analysis, which are less automated or require heavier computation, such as:

- **Code generation and verification for interlockings** is possible e.g. through the formal verification framework of Prover Technology.
- **Capacity analysis and timetabling** can be performed using e.g. OpenTrack, LUKS, or Treno.
- **Building information modelling (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.

3 Semantic CAD

Civil engineering construction projects, such as railway projects, make heavy use of computer-aided design (CAD) tools to model the *geometric* aspects of the construction project and its product. The origins of CAD tools are in the computerising of traditional *drafting*, which produces human-readable technical drawings that are used as plans and documentation for the construction. Mainstream CAD tools are mainly concerned with manipulating databases of geometrical objects constituting 2D or 3D representations of spatial properties, and the production of human-readable drawings which depict these geometrical structures.

Certain extensions of CAD are widely used, such as *computer-aided engineering* (CAE) and *building information modelling* (BIM). CAE extends CAD by adding engineering analyses with physical properties, such as finite element analysis, computational fluid dynamics, etc. using the CAD geometry as input.

3.1 Blocks and abstractions

Grouping together several geometrical features into a single unit is in CAD terminology called "making a *block*". This allows the CAD user to create models more efficiently, by reusing commonly used components. The blocks, which may represent such things as chairs, doors, or railway signals, also create the opportunity to store higher-level information in a CAD model, other than the pure geometrical description. For example, if one uses a railway signal block to model signal in a railway station, a program can count the number of signals in a model.

This idea can be extended by adding any number of attributes to a block. For a railway signal, we can add attributes that describe e.g. to which track it signals, along with its type, function, and direction. The CAD object database does then not only contain the geometrical objects, such as lines, curves, triangles, cubes, etc., but groups these primitives into higher-level concepts which are closer to the representation that one uses to reason about the actual working of the railway infrastructure.

With a good library of blocks (called a *symbol library*), the engineer can more efficiently build the geometric CAD models which lead to human-readable drawings, but they are also building a machine-readable model of high-level railway concepts. We call this *semantic CAD*. While this concept is also a part of building information modelling (BIM), BIM also includes many other concepts such as 3D visualisation, time ("4D"), and cost ("5D").

3.2 Object oriented modelling

The verification of signalling and interlocking rules requires information about 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 hierarchical object model *railML* [6]. 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 stores fragments of railML in this way, and can thus compile a complete railML infrastructure description from the CAD model.

It is necessary to decide which objects in the CAD model should be associated with which data types, i.e. what attributes should be stored in the symbols. This is comparable to specifying an object's *class* in an object-oriented programming language. To do this, we create an *object type description* which augments the symbol library with class information. Whenever the user adds a symbol, its data editor is determined by the assigned class, and vice versa: when e.g. a railML object is imported into CAD, its corresponding symbol is inserted in the graphical model. RailCOMPLETE® contains a comprehensive symbol library for railway signalling CAD design, and a complementing object type description.

3.3 Interlocking and train protection systems

Besides the CAD model layout, the design of a railway station's signalling consists also of specifications for the interlocking and train protection (speed control) systems. These specifications are used to build the interlocking controllers and speed controllers, and they model the behaviour of the signalling equipment and its interaction with trains. These systems are tightly linked to the station layout.

A formal representation of the interlocking and train protection specifications 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, geographical 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 and train protection systems. This allows analysis of the operational aspects of the train station directly in a familiar editable CAD model. See Figure 2 for an overview of this architecture.

4 Logic programming and knowledge-base systems

We have shown how to associate semantic information with CAD symbols, but in order to automatically verify rules and regulations on this railway infrastructure model, we need a computer program which can check each rule for violations with the given model as input.

A straight-forward approach to making such a program is to create some search function, e.g. a depth-first or breadth-first search on the graph implicit in the track network. This procedure should allow, for example, to find the nearest object of a given type, or to find all paths between two points. With this framework in place, we could describe a checking procedure for each rule. As example, we consider the *home signal* rule.

Property 1 (Layout: Home signal [7]) *A* home main signal *shall be placed at least 200 m in front of the first controlled, facing switch in the entry train path.*

Checking such a rule can be done by iterating over tracks, locating station boundaries, starting a search function to locate the relevant facing switches, starting another search backwards to check that there is a home signal, and so on.

The amount of code required to do this in a mainstream programming language can become large, and this code is often very specific to a given railway administration. To manage the large amounts of code required for a large amount of rules, we turn to *logic programming*, which allows rule descriptions that are much closer to the original specifications than in a mainstream programming language.

4.1 Logic programming

Logic programming is a family of programming languages based on formal logic. Logic programs are *declarative*, i.e. they describe properties of the solution of a problem rather than a calculation procedure for finding the solution. This separates the concerns of expressing rules about railway systems from the algorithms required to do automatic analysis. This separation allows one to systematically maintain a large set of rules, and decouple the tool implementation from the set of concepts, rules and expert knowledge that is specific to a railway administration.

We have successfully used the Datalog language, a subset of the more well-known Prolog language, for verifying many properties given as technical rules and expert knowledge. It allows concise formulations of railway concepts, and queries can be efficiently calculated. For an introduction to Datalog and its use in railway signalling rules, see [8, Sec. 4].

Ideally, we would like the railway engineers themselves, without much programming education, to be able to create and maintain the set of rules which

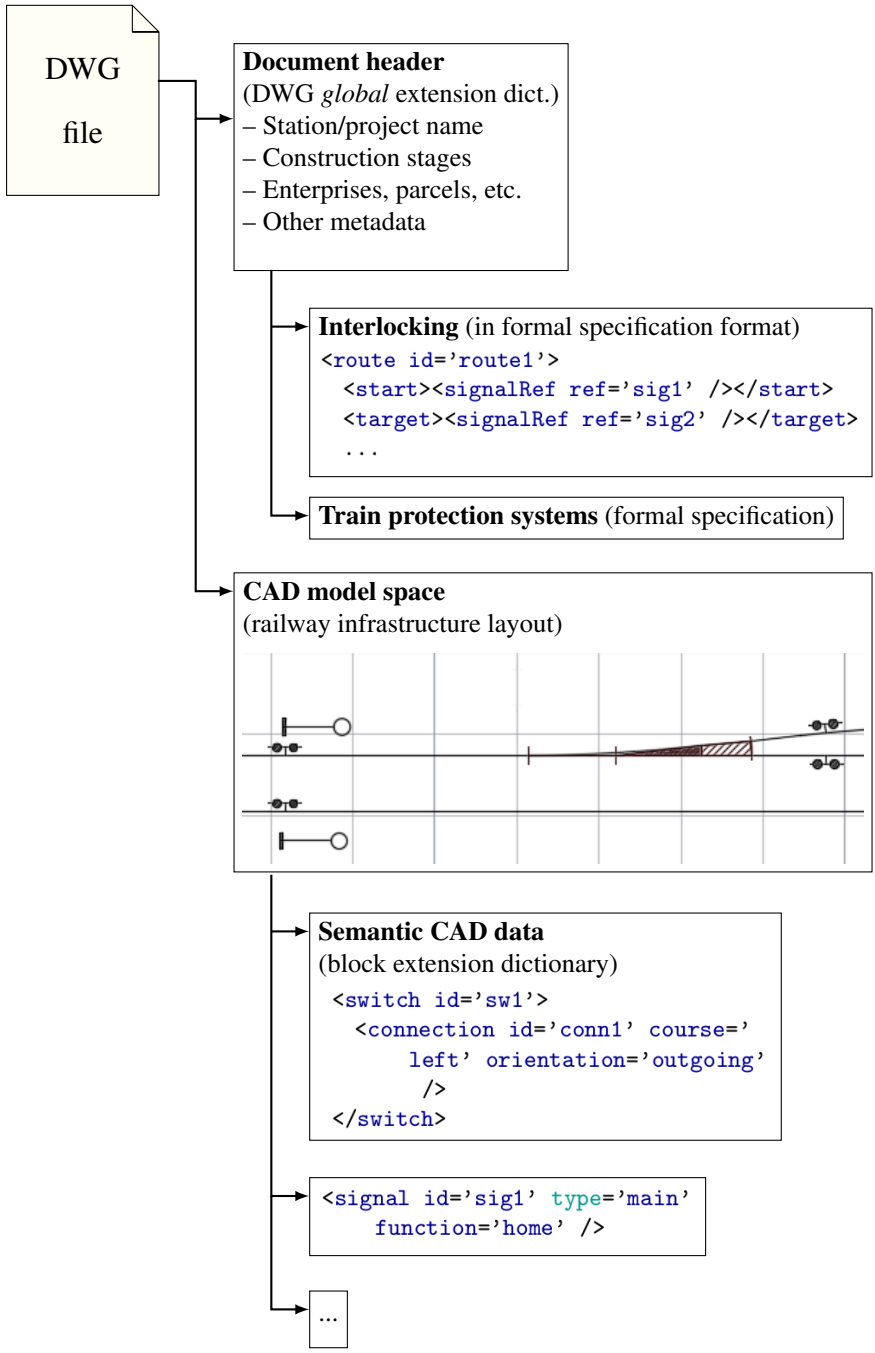

Figure 2: Semantic CAD document organisation including interlocking specification.

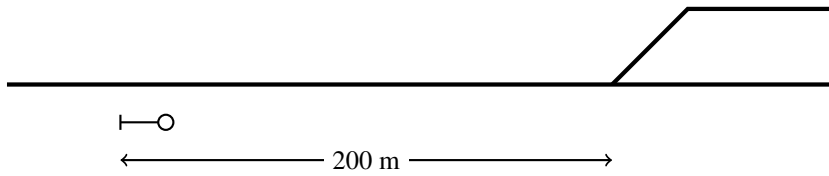

Figure 3: Home signal layout rule example (Property 1).

is used for the verification. This separation of logic and algorithm is a step in this direction, because non-IT experts can work on the rules without considering how the calculations are implemented. However, the strict formalism and subtle semantics of logic programming are still a challenge for any inexperienced programmer. Further work on more user-friendly specification languages for railway properties could improve the situation.

We envision that a common rule base would be exchanged between all engineers working with a railway administration, and that the rule base would be worked out partly by software experts, partly by railway experts. Also, the rule base should be fairly constant, like the regulations, requiring an update frequency of perhaps once per year.

4.2 Knowledge-base system

With Datalog as specification language, we build a *knowledge-base system* to perform the verification. A knowledge-base system consists of a set of facts and rules, along with an inference engine which answers queries by applying logical inference rules. For an introduction to knowledge-base systems in general, see [9, Ch. 3] or [10, Chs 8 and 12]. We give here an overview of how we encode railway signalling properties as Datalog predicates, which in turn may be automatically checked for consistency. In our verification tool, we organise our knowledge base in the following manner:

1. **Input documents:** Predicate representation of input document, i.e. track layout and interlocking, are represented as *facts* which are converted from the railML representation stored and maintained in the CAD database by a CAD plug-in program.
2. **Derived concepts:** Predicate representation of derived concept *rules*, such as object properties, topological properties, and calculation of distances. A library of general railway concepts and administration-specific concepts and definitions are kept in a rule base which is re-used between projects.
3. **Technical rules and expert knowledge:** Predicate representation of technical rules or expert knowledge as logic programming *rules*, which encode the administration-specific rules and expert knowledge that is checked and reported to the user of the verification tool.
4. **Inference engine:** A Datalog evaluation engine is used as inference engine, in our case the XSB Prolog tabled logic programming system.

A short description of each of these aspects are given in the sections below. For a more thorough look on this translation and formalisation procedure, see [8].

4.2.1 Input documents

Each of the XML elements and attributes is translated into a corresponding predicate. An example of translating a railML *switch* element into predicate representation is given below.

```
<switch id='sw1'>                    switch(sw1).
  <connection id='conn1'             connection(conn1).
      course='left'                  belongsTo(sw1,conn1).
      orientation='outgoing'    →    course(conn1,left).
      />                             orientation(conn1,
</switch>                                outgoing).
```

4.2.2 Derived concepts

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:

- Switch facing or trailing ($a \in \text{Atoms}$, $d \in \text{Direction}$):

$$switchFacing(a, d) \leftarrow \exists c, o : switch(a) \wedge switchConnection(a, c) \wedge switchOrientation(c, o) \wedge orientationDirection(o, d).$$

$$switchTrailing(a, d) \leftarrow \neg switchFacing(a, d)$$

Predicates describing the topological configuration of signalling objects, and the train travelling distance along the tracks connecting them, are described by predicates for **track connection** (predicate $connected(a, b)$), **directed connection** (predicate $following(a, b, d)$), **distance** (predicate $distance(a, b, d, l)$), etc. For example, the track connection predicate is defined as:

- There is a **track connection** between object a and b ($a, b \in \text{Atoms}$):

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

$$connected(a, b) \leftarrow directlyConnected(a, b) \vee (\exists c_1, c_2 : connection(c_1, c_2) \wedge directlyConnected(a, c_1) \wedge connected(c_2, b)).$$

4.2.3 Technical rules and expert knowledge

A technical rule from Norwegian infrastructure manager Jernbaneverket [7] is used here as an example of a layout rule using geometric information. We have tested our tool with rules of different kinds, verifying topological, geometrical and interlocking properties. See [8] and [11] for details. The *home signal* property

(Property 1 above) may be represented in the following way:

$$isFirstFacingSwitch(b, s) \leftarrow stationBoundary(b) \wedge facingSwitch(s) \wedge \neg(\exists x : facingSwitch(x) \wedge between(b, x, s)),$$

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

Checking for rule violations can be expressed as:

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

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

5 Verification procedure as CAD plug-in program

A prototype CAD add-on program for Autodesk AutoCAD was implemented, see Figure 4. The XSB Prolog interpreter was used as a back-end for the implementation of a verification procedure, as it offers tabled predicates which have the same characteristics as Datalog programs [12], while still allowing general Prolog expressions such as arithmetic operations.

The translation from railML to Datalog facts assumes that the document is valid railML, which may be checked with general XML schema validators, or a specialised railML validator.

When rule violations are found, the railway engineer will benefit from information about (a) *which rule* was violated (with a textual message referencing the source of the rule) and (b) *where* the rule was violated (identity of the objects that are involved). Also, rules are classified based on e.g. *discipline* and *severity*. In the rule databases, this is accomplished through the use of *structured comments*, similar to the common practice of including structured documentation in computer

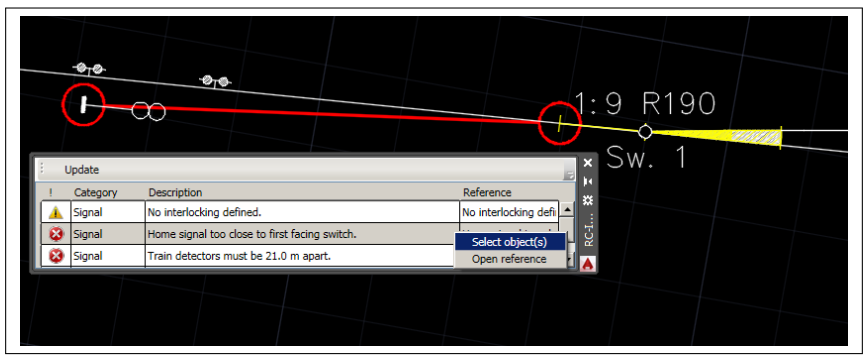

Figure 4: Counterexample presentation within an interactive CAD environment.

programs, such as JavaDoc. 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.

5.1 Usage and performance

A set of signalling layout and interlocking technical rules from *Jernbaneverket* [7] were checked using the railML representation of the Arna-Fløen project which is an ongoing design project in Anacon AS (now merged with Norconsult). The rule collection consisted of 37 derived concepts, 5 consistency predicates, and 8 technical predicates. The station CAD model that was used for the work with the Arna station included 20 tracks, 15 switches, 43 signals, 37 train detectors. The total number of Datalog facts for this model is 3180.

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 Section 2, 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.

For the interactive use of this verification tool, productivity can be increased if the calculations are performed "on-the-fly", i.e. quickly and continuously while the user is modifying the railway model. This moderately sized station could be checked in about 0.1 second. See [11] for details. This is promising for performance on larger stations and rule sets. Datalog engines will re-use facts from calculating one rule in another rule by storing the results of evaluating the derived concepts in memory, and the re-using them in other rules. For example, the $distance$ predicate might be populated with facts while calculating a rule, which may then be re-used when calculating other rules.

For situations with many rules which are run on large railway models, e.g. while modelling large stations or combining many stations into a single model, the running time for verifying the whole station eventually becomes too long for on-the-fly updates. For example, moving a single signal to a new position would change the facts of the knowledge base, requiring a new verification.

An approach which keeps the intermediate results from the last verification and recalculates only those predicates affected by the changes being made in the model would be able to handle on-the-fly verification of much larger models. Such an approach has been researched for bottom-up Datalog evaluation [13, 14]. Also, parallel processing may be applied to Datalog queries [15].

6 Conclusions

We have shown the results of a prototype implementation of an automatic verification procedure for static properties of railway signalling and interlocking designs, and how this procedure can be tightly integrated with existing CAD practices in railway engineering.

The logical formalism used to express the rule set for a specific infrastructure manager may also be used to maintain and exchange a body of expert knowledge between engineers. Both the continuous verification during interactive CAD work and the formalised knowledge transfer that results from this method may contribute to increasing the quality and efficiency of railway design work.

6.1 Future work

We have ongoing work to study the feasibility of implementing relevant rules and evaluating the performance of our verification on a larger scale. Verification of geographical interlockings and Automatic Train Control (ATC) systems could also be included. The current work is assuming Norwegian regulations, but the impact of European Rail Traffic Management System (ERTMS) on verification should be investigated. Finally, we are considering the use of incremental updates on the Datalog queries to see if a truly on-the-fly verification can be feasible for large model sizes and joint models across several projects.

Acknowledgement

We thank our collaborators from Oslo University, Christian Johansen and Martin Steffen, for helping with automation and logical aspects.

References

[1] J.-L. Boulanger, *CENELEC 50128 and IEC 62279 Standards*. Wiley-ISTE, Mar. 2015.
[2] A. Ferrari, G. Magnani, D. Grasso, and A. Fantechi, "Model checking interlocking control tables," in *FORMS/FORMAT 2010*, pp. 107–115.
[3] S. Busard, Q. Cappart, C. Limbrée, C. Pecheur, and P. Schaus, "Verification of Railway Interlocking Systems," *Electronic Proceedings in Theoretical Computer Science, (Workshop at FM'15, Oslo)*, June 2015.
[4] A. Haxthausen, J. Peleska, and R. Pinger, "Applied bounded model checking for interlocking system designs," in *Workshops on SW Eng. and Formal M.*, Springer, 2014.
[5] M. Lodemann, N. Luttenberger, and E. Schulz, "Semantic computing for railway infrastructure verification," in *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pp. 371–376, Sept. 2013.
[6] A. Nash, D. Huerlimann, J. Schütte, and V. P. Krauss, "RailML — a standard data interface for railroad applications," *WIT Press*, 2004.
[7] Jernbaneverket, "Teknisk regelverk." `http://trv.jbv.no/`, 2015.
[8] B. Luteberget, C. Johansen, and M. Steffen, "Rule-based consistency checking of railway infrastructure designs," Technical report 450, Univ. of Oslo, Jan. 2016.

[9] J. D. Ullman, *Principles of Database and Knowledge-Base Systems*. CSP, 1988.
[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 ed., 2003.
[11] B. Luteberget and C. Johansen, "Verification of rules and regulations in CAD models of railway signalling," technical white paper, RailComplete, Feb. 2016. http://www.mn.uio.no/ifi/english/research/projects/railcons/.
[12] T. Swift and D. S. Warren, "XSB: Extending Prolog with tabled logic programming," *Theory Pract. Log. Program.*, vol. 12, pp. 157–187, Jan. 2012.
[13] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining views incrementally," SIGMOD '93, (New York, NY, USA), pp. 157–166, ACM, 1993.
[14] B. Motik, Y. Nenov, R. E. F. Piro, and I. Horrocks, "Incremental update of Datalog materialisation," in *Proceedings of the AAAI Conf. on Artificial Intelligence*, 2015.
[15] B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu, "Parallel materialisation of Datalog programs in centralised, main-memory RDF systems," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.