Rule-Based Incremental Verification Tools Applied to Railway Designs and Regulations

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

[1] RailComplete AS, Sandvika, Norway
{bjlut,clfey}@railcomplete.no
[2] Department of Informatics, University of Oslo, Oslo, Norway
{cristi,msteffen}@ifi.uio.no

Abstract. When designing railway infrastructure (tracks, signalling systems, etc.), railway engineers need to keep in mind numerous regulations for ensuring safety. Many of these regulations are simple, but demonstrably conforming with them often involves tedious manual work. We have worked on automating the verification of regulations against CAD designs, and integrated a verification tool and methodology into the tool chain of railway engineers. Automatically generating a model from the railway designs and running the verification tool on it is a valuable step forward, compared to manually reviewing the design for compliance and consistency. To seamlessly integrate the consistency checking into the CAD work-flow of the design engineers, however, requires a fast, on-the-fly mechanism, similar to real-time compilation done in standard programming tools.

In consequence, in this paper we turn to *incremental* verification and investigate existing rule-based tools, looking at various aspects relevant for engineering railway designs. We discuss existing state-of-the-art methods for incremental verification in the setting of rule-based modelling. We survey and compare relevant tools (ca. 30) and discuss if/how they could be integrated in a railway design environment, such as CAD software. We examine and compare four promising tools: XSB Prolog, a standard tool in the Datalog community, RDFox from the semantic web community, Dyna from the AI community, and LogicBlox, a proprietary solution.

1 Introduction

Verification of railway systems using formal methods often focuses on interlocking and dynamic safety of the implementation. Often overlooked, however, is the early-stage planning process for railway systems where the design decisions are made. The design process is concerned with producing a specification of the

Part of this research has been supported by the Norwegian Research Council project RailCons (Automated Methods and Tools for Ensuring Consistency of Railway Designs).

J. Fitzgerald et al. (Eds.): FM 2016, LNCS 9995, pp. 772–778, 2016.
DOI: 10.1007/978-3-319-48989-6_49

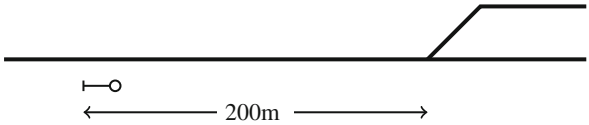

Fig. 1. Home signal layout rule example (Property 1).

railway infrastructure, which we call *the design*, with documented safety and performance requirements. During that phase, it is important to *efficiently handle changes* in track layouts, component capabilities, performance requirements, etc. Tool support for this process is practically unavailable. Such tools would be concerned with verification of the railway infrastructure w.r.t. technical regulations, typically expressing static properties concerned with object properties, topology, geometry, and interlocking specifications.

As an example of a regulation to be verified, we consider the *home signal* rule (Property 1 below, see also Fig. 1). Ensuring that a design is compliant with a large set such regulations could give significant productivity and quality gains, especially if the compliance information could be immediately available after making changes to the design.

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

Section 2 shortly describes the current state of our tool for checking consistency of industrial railway designs, introducing the practical problem of on-the-fly verification. Sect. 3 then describes the *existing techniques for incremental verification* for rule-based modelling. We then survey in Sect. 4 existing tools related to Datalog and focus on those supporting incremental verification. We are particularly interested in industry-ready tools. We end in Sect. 5 by comparing efficiency gains due to incremental evaluation when applied to the industrial case study of the Arna station reconstruction, and suggesting how existing tools could be improved to help make our incremental verification production-ready.

2 Integrating Verification Tools into Railway Engineering Tools

In [8], we presented and demonstrated a verification tool for static infrastructure properties based on evaluation of Datalog rules. The tool is integrated into the RailCOMPLETE® software, a professional railway CAD program for producing and editing *railML* representations of railway infrastructure. The railML format [11] is an international standard for describing railway infrastructure, time tables, and rolling stock information. The railML description is transformed into a logical model for verification.

The modelling and verification has the following characteristics: it (1) uses Datalog (many properties depend on graph reachability encoded as transitive

```
%| 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. 2. Structured comments attached to a rule expressing violation of a regulation.

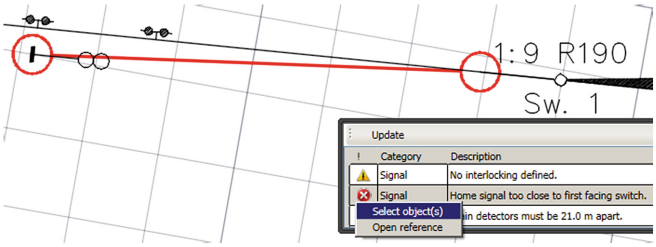

Fig. 3. Counter-example presentation within the RailCOMPLETE® CAD tool.

closures), and uses (2) negation with negation-as-failure semantics (stratified negation). Finally, and going beyond pure Datalog, it uses (3) arithmetic, to model aspects such as distances.

Our prototype implementation uses *XSB Prolog* which does conventional top-down Prolog search, combined with tabling of recursive predicates, ensuring the Datalog properties of termination and polynomial running time. Figure 2 shows an example rule input corresponding to a railway property, whereas Fig. 3 shows the graphical representation indicating to the engineer which regulation is violated. The tight integration into the CAD program and, as such, into the engineer's design process, creates the demand for fast re-evaluation of all conclusions upon small changes to the railway designs. The performance studies of [8] show that the current implementation is well acceptable for "one-shot" validation even for realistic designs with running times in the range of seconds (the tool is applied to a real train station currently under construction). However, it is not fast enough to *smoothly* and transparently be integrated such that it can automatically rerun the complete verification for each small change.

3 Incremental Verification for On-the-Fly Performance

An alternative approach that promises to be more efficient is *incremental verification*: instead of solving logic programs from scratch for each verification run, it tries to materialize all consequences of the base facts and then maintains this view under fact updates. The existing literature on incremental materialization of Datalog programs gives various strategies for doing this efficiently. We briefly

survey methods for incremental evaluation of Datalog programs, also known in the deductive database literature as the *view maintenance* problem [5] [1, Chap. 22]. We also survey relevant tools and compare their features (e.g., availability, industry-quality, performance) in the context of our verification tool. A more thorough evaluation appears in a long version of this work [9].

Datalog systems use rules to derive a set of consequences (*intensional* facts), from a given set of base facts (*extensional* facts). Typically, Datalog systems use a *bottom-up* (or *forward-chaining*) evaluation strategy, where all possible consequences are materialized [15, Chap. 3] [1, Chap. 13]. This simplifies query answering to simply looking up values in the materialization tables. Any change to the base facts, however, will invalidate the materialization. Several approaches have been suggested to reduce the work required to find a new materialization after changing the base facts.

First, if considering only addition of facts to positive Datalog programs, i.e. without negation, then the standard *semi-naive* algorithm [15, Chap. 3] [1, Chap. 13] is already an efficient approach. The real challenge are non-monotonic changes, i.e., removing facts appearing positively in rules or adding facts appearing negatively in rules. Non-monotonicity is essential in our railway infrastructure verification rules. Graph reachability is prominent in many of the regulations for railway signalling, so efficiently maintaining rules involving transitivity is also essential. Some algorithms, such as truth maintenance systems [3], work by storing more information (in addition to the logical consequences) about the *supporting facts* for derived facts, so that removal of supporting facts may or may not remove a derived fact. This allows efficient removal of facts, at the cost of requiring more time and memory for normal derivations. Another class of algorithms, working *without* additional "bookkeeping", can be more efficient if the re-evaluation of sets of facts is relatively easy compared to re-materializing all facts. The Propagation-Filtering algorithm [7] works on each removed fact separately, propagating it through to all rules which depend on it. In contrast, the Delete-Rederive (DRed) algorithm [6] is rule-oriented and works on sets of facts, first over-approximating all possible deletions that may result from a change in base facts, then re-deriving any still-supported facts from the over-deleted state before finally continuing semi-naive materialization on newly added facts. Recently, the Forward/Backward/Forward (FBF) algorithm [10] used in RDFox improved the DRed algorithm in most cases by searching for alternative support (and caching the results) for each potentially deleted fact before proceeding to the next fact. Notably, this method performs better on rules involving *transitivity,* as deletions do not propagate further than necessary.

4 Datalog Tools for Incremental Verification

Our procedure uses rule-based modelling and verification techniques in the style of Datalog. In consequence, we perform a survey of Datalog-based and related tools. The logic programs for our verification make use of recursive predicates, stratified negation, and arithmetic. Therefore, we pay particular attention to tools that at least satisfy these needs. In addition, we are looking for high performance on relatively small (in-memory) data sets, so light-weight library-style

logic engines are preferred. High-performance distributed "big data" type of tools have less value in this context.

XSB Prolog continuously developed since 1990, has constantly been pushing the state of the art in high-performance Prolog. XSB is especially known for its tabling support [14], which allows fast Datalog-like evaluation of logic programs without restricting ISO Prolog. The tabling support was extended to allow incremental evaluation [12], and these features have been under continued development and seem to have reached a mature state [13]. For some applications, however, the additional memory usage for incremental tabling can lead to a significant increase in the total memory needed.

RDFox is a multicore-scalable in-memory RDF triple store with Datalog reasoning. It reads semantic web formats (RDF/OWL) and stores RDF triples, but also includes a Datalog-like input language which can describe SWRL rules. This rule language has been extended to include stratified negation and arithmetic. The RDFox system also implements a new algorithm called FBF for incremental evaluation [10].

RDFox stores internally only triples as in RDF, which, in Datalog, corresponds to only using unary and binary predicates. A method of reifying the rules for higher-arity Datalog predicates into binary predicates allows RDFox to calculate any-arity Datalog programs. However, this requires separate rules for each component of the predicate, and when doing incremental evaluation, the FBF algorithm's backward chaining step then examines all combinations of components potentially involved. Because of this problem, using RDFox incrementally did not improve running times in our case study.

LogicBlox is a programming platform [2] for combining transactions with analytics in enterprise application areas including web-based retail planning and insurance. It uses a typed, Datalog-based custom language LogiQL and has a comprehensive development framework. It claims support for incremental verification, but we could not evaluate it on our railway example due to absence of freely downloadable distributions.

Dyna is a promising new Datalog-like language for modern statistical AI systems [4]. It has currently not matured sufficiently for our application, but its techniques are promising, and we hope to see it more fully developed in the future.

Many other Datalog tools are available (around 30), few of them supporting incremental evaluation. An overview and our brief evaluation of them can be found in the technical report [9]. We hope to include these findings also in the Wikipedia page for Datalog.[1]

5 Efficiency Gains, Shortcomings, and Possible Ways Forward

Table 1 compares the running time and memory usage for the verification on Arna station used as a reference station in RailCOMPLETE. The railway

[1] https://en.wikipedia.org/wiki/Datalog#Systems_implementing_Datalog.

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 input facts			85	8283	9159
XSB:					
Non-incrementalverif.:	Running time	(*s*)	0.015	2.31	4.59
	Memory	(MB)	20	104	190
Incremental verif. baseline:	Running time	(*s*)	0.016	5.87	12.25
	Memory	(MB)	21	1110	2195
Incr. single object update:	Running time	(*s*)	0.014	0.54	**0.61**
	Memory	(MB)	22	1165	2267

signalling design project for this station is currently in progress by Norconsult AS. The extra bookkeeping required in XSB to prepare for incremental evaluation requires more time and memory than non-incremental evaluation, so we include both non-incremental and from-scratch incremental evaluation in the table for comparison. We show how updates can be calculated faster than from-scratch evaluation by moving a single object (an axle counter) in and out of a disallowed area near another object (regulations require at least 21.0 m separation between train detectors). Without using abstraction methods, the case study verification uses over 2 GB of memory. So, for any hope of handling larger stations on a standard laptop or workstation, this must be reduced. We were not able to reduce memory usage in this case study using the abstraction methods in XSB (version 3.6.0).

While currently none of the tools seem to satisfy all conditions we hoped for in our integration, notably efficiency, but also maturity and stability, it should also be noted that the need for incremental evaluation has been identified by the community not only as theoretically interesting, but also as of practical importance. The RDFox developers aim to support incremental updates of higher-arity predicates in a later version. The XSB project has made efforts to improve its abstraction mechanisms, so future versions might become feasible for our use. If reducing the memory usage would require adapting a Datalog algorithm (such as DRed), then XSB's unrestricted Prolog might be a challenge. A different approach would be to extend another efficient Datalog tool, such as Soufflé, to do incremental evaluation, which could require a significant effort.

References

1. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases, 1st edn. Addison-Wesley Longman Publishing Co., Boston (1995)
2. Aref, M., ten Cate, B., Green, T.J., Kimelfeld, B., Olteanu, D., Pasalic, E., Veldhuizen, T.L., Washburn, G.: Design and implementation of the LogicBlox system. In: SIGMOD International Conference on Management of Data, pp. 1371–1382. ACM (2015)
3. Doyle, J.: A truth maintenance system. Artif. Intell. **12**(3), 231–272 (1979)
4. Eisner, J., Filardo, N.W.: Dyna: extending datalog for modern AI. In: Moor, O., Gottlob, G., Furche, T., Sellers, A. (eds.) Datalog 2.0 2010. LNCS, vol. 6702, pp. 181–220. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24206-9_11
5. Gupta, A., Mumick, I.S., et al.: Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng. Bull. **18**(2), 3–18 (1995)
6. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. In: SIGMOD International Conference on Management of Data, pp. 157–166. ACM (1993)
7. Harrison, J.V., Dietrich, S.W.: Maintenance of materialized views in a deductive database: an update propagation approach. In: Workshop on Deductive Databases, pp. 56–65 (1992)
8. Luteberget, B., Johansen, C., Steffen, M.: Rule-based consistency checking of railway infrastructure designs. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 491–507. Springer, Heidelberg (2016). doi:10.1007/978-3-319-33693-0_31
9. Luteberget, B., Johansen, C., Steffen, M.: Rule-based consistency checking of railway infrastructure designs (long version). Technical report 450, University of Oslo (IFI) (2016)
10. Motik, B., Nenov, Y., Piro, R.E.F., Horrocks, I.: Incremental update of datalog materialisation: the backward/forward algorithm. In: Proceedings of AAAI 2015. AAAI Press (2015)
11. Nash, A., Huerlimann, D., Schütte, J., Krauss, V.P.: RailML – a standard data interface for railroad applications, pp. 233–240. WIT Press (2004)
12. Saha, D., Ramakrishnan, C.R.: Incremental evaluation of tabled logic programs. In: Palamidessi, C. (ed.) ICLP 2003. LNCS, vol. 2916, pp. 392–406. Springer, Heidelberg (2003). doi:10.1007/978-3-540-24599-5_27
13. Swift, T.: Incremental tabling in support of knowledge representation and reasoning. Theory Pract. Log. Program. **14**(4–5), 553–567 (2014)
14. Swift, T., Warren, D.S.: XSB: extending Prolog with tabled logic programming. Theory Pract. Log. Program. **12**(1–2), 157–187 (2012)
15. Ullman, J.D.: Principles of Database and Knowledge-base systems, vol. I & II. Computer Society Press (1988)