Static Verification of Railway Schema and Interlocking Design Data

Alexei Iliasov(✉), Paulius Stankaitis, and David Adjepon-Yamoah

Newcastle University, Newcastle upon Tyne, UK
alexei.iliasov@ncl.ac.uk

Abstract. The paper presents an experience of verifying a large scale, real-life dataset describing various aspects of railway station design. We discuss how a number of assorted digital artefacts were pooled together and converted into a set-theoretic model over which a type inference procedure is run. The typed model is then used to confirm or contradict logical conjectures over data elements. We employ a number of state-of-the-art SMT solvers as a verification back-end. The project is ongoing but has already identified a number of issues in topology definition and signalling data that were missed by other automated tests and not revealed by simulation tools.

1 Introduction

The SafeCap project has been working on railway modelling and formal verification for nearly five years. The original view consisted in fitting a railway description into a formal setting through the means of a formal domain specific language [4]. Such a language enables formal and automatic verification of integrity of a schema and its signalling as well as operational safety. True to the well respected practice in computing science, a strict top-down approach was used where a formal model spanning abstraction levels of increasing fidelity covered concepts of safety (e.g., something bad should not happen), principles of safety (e.g., route-based signalling) and implementation of safety principles (ladder logic diagrams for interlocking) [1]. However, the reality turned out to be far more fragmented and fluid to insist on a strict top-down view.

Collaboration with Siemens Rail Automation UK led to the realisation that railway models, i.e., a description of a station, are rarely available in their entirety while the scale of a project and the pace of changes make it unrealistic to undertake an unhurried top-down validation. The variety of ways employed in the industry to capture the same artefact both at conceptual and syntactic levels (say a schema rendered as a track topology or node/edge model and persisted in XML or LDL format) makes it much harder to come up with a universal verification pipeline. And at the signalling implementation level one finds a medley of proprietary technology, notations and tools.

It is increasingly evident that safe and efficient exploitation of a railway network depends on detailed and up to date knowledge of network characteristic

T. Lecomte et al. (Eds.): RSSRail 2016, LNCS 9707, pp. 123–133, 2016.
DOI: 10.1007/978-3-319-33951-1_9

spanning from macroscopic details of station and line capacity to precise characterisation of track side equipment positioning, capabilities and state. Given the scale and importance of such data, a modern railway operation critically depends on data acquisition and storage that are suitably supported by higher level activities of modelling, analysis and planning.

At the purely technological level, storage and distribution of large amount of data is no longer a challenging problem. The issues of scaling, querying, replication and persistence are well researched by the data science community.

A far harder problem, however, is to make sure that collected and stored data makes sense, especially in the presence of incremental updates and non-trivial semantic overlaps in data originating from differing companies. A data collection exercise carried out as merely a dutiful recording and redistribution of incoming data is bound to result in a situation where same information is duplicated and triplicated with slight changes and ever accumulating number of inconsistencies making data interpretation increasingly difficult and the data less valuable overall. What we endeavour to achieve in SafeCap is a way to 'interpret' data without human involvement and through this validate and normalise it to deter 'data rot' while enabling much more semantically involved data querying and processing.

One solution is defining a set of mechanisable validation rules that apply automatically every time a new piece of data is added to a storage. At the simplest level a rule is a piece of code (say, a stored routine for a database engine) which role is to go through entries and check for known signs of mismatch. This will undeniably save a great amount of time. However, given the scale of the challenge - real-life examples contain hundreds of distinct concepts - a new challenge arises almost immediately. Since the responsibility is now delegated to validation code, such code has to be developed, verified and maintained to the highest standard. This is a difficult task and, to start with, the formal verification of a piece of code requires a formal model of its intended behaviour to check against. What SafeCap offers is having just a formal model of data semantics and a technique to match it, automatically, against a piece of data.

The technique comprises two stages: container agnostic extraction of a formal model describing source data and model validation. To avoid a dependency on certain syntax and container, the information necessary for validation comes in the form of typed relations. We use a certain mathematical framework for representing and classifying relations and their typing constraints. Any structured data may be rendered as a collection of relations and, if it meets minimal consistency requirements, relation types may be inferred. This process is completely automatic and does not require any knowledge about structure or purpose of data source. It applies equally to structured textual formats and relational databases.

2 Formal Model

We build a simple conceptual model of a railway operation. The basic premise is that railway track is a contended resource and there is a number of actors that

affect each other by consuming and freeing track resources. The basic unit of consumption is a block. We do not define how small or large a block is in terms of physical track; the block concept may be used for route-based and moving block (ERTMS) signalling.

A pre-existing model of data semantics would speak about mathematical relations as well; specifically the kind of relations that are permitted and considered well-formed. But these would generally be different from extracted relations and with differing types. However, in such a formal setting it is not necessary to translate source data into a new format: a potentially dangerous exercise that may alter source data semantics. In its stead, one defines a formal link model that semantically links extracted data model with the verification model. The link does not need to be total: some elements of source may be uninterpreted while some concepts of semantics might have no direct counterpart. Unlike a software translator tool, such link model is generally not executable as we don't need actual translation of data to perform verification. At the same time it is terse and white-box and can be easily evolved with the changes in source data formats.

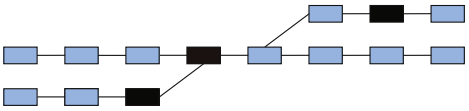
Once these three models - source data, link and semantics - are put together we have a model that is consistent, when all model parts are in an agreement, or not. The check is performed via automated theorem proving using a range of state-of-the-art automatic verification tools. If the check fails, the reason can often be narrowed down to a specific source data structure and semantic model constraint.

Railway Track Topology. The first step is to define constraints on track topology, that is a graph of blocks. For instance, track graph must have nodes of only degree one (boundary), two (normal), three (point) and four (diamond crossing). There should be no cycles, self-loops and disjointed sections. Points and diamond crossings should not appear as boundary nodes.

Definition 1 (Track topology assumptions).

$$finite(BLOCK) \tag{1}$$
$$next \subseteq BLOCK \times BLOCK \tag{2}$$
$$BLOCK = ran(next) \cup dom(next) \tag{3}$$
$$next \cap (BLOCK \lhd id) = \varnothing \tag{4}$$
$$next\_closure \subseteq BLOCK \times BLOCK \tag{5}$$
$$next\_closure; next = next\_closure \tag{6}$$
$$next\_closure; (BLOCK \lhd id) = next\_closure \tag{7}$$
$$first = dom(next) \setminus ran(next) \tag{8}$$
$$last = ran(next) \setminus dom(next) \tag{9}$$
$$points \subseteq BLOCK \tag{10}$$
$$\ldots \tag{11}$$

Signalling Model. At the most abstract level we observe blocks being consumed and freed. This is a high-level metaphor for train movement and point/route locking. The following diagram shows three blocks consumed at some point of time.

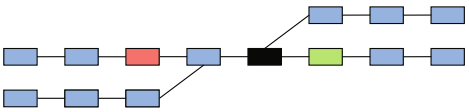


The model of behavior is given in a formal notation called Event-B. The first model is extremely simple and defines one variable $consumed \subseteq \mathrm{BLOCK}$ (the set of consumed blocks) and two events (actions) to consume and free blocks.

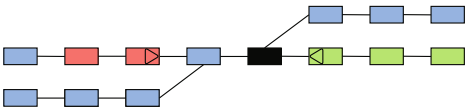
Definition 2 (Abstract model).

$consume \triangleq$
any b **where**
$b \in \mathrm{BLOCK} \setminus consumed$
then
$consumed := consumed \cup \{b\}$
end
$free \triangleq$ **any** b **where** $b \in consumed$ **then** $consumed := consumed \setminus \{b\}$ **end**

Refinements. The abstract consume/free model is gradually refined to capture route-based signalling based on a control table. We introduce the notion of actors and keep track of which actor consumes which block. There are two main actor kinds: the control actor that consumes only points and diamond crossing; and a train actor that may consume any block kind. The following depicts a situation where three blocks are consumed by three different actors (red, green and black - colours differentiate into block reservation and train occupation).



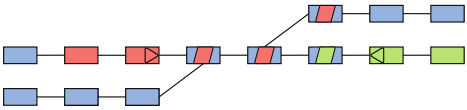
We make model realistic by requiring that train actors consume contiguous blocks and also keep track of train orientation. A train actor may only appear and disappear (that is, consume its first block) on a boundary block; it may also reverse its direction when its head is on a buffer stop block. At this stage, trains travel through a point or crossing in any direction (even when point topology would not allow this). The following diagram shows occupation and reservation for directed trains (a triangle in block depicts train head):



The subsequent refinements introduce the notion of train path through a schema; point and diamond crossing states, and the concept of block locking. Now a block may only be consumed once it is locked for a given actor (a slanted

stripe in the diagram below). For a control, the consumed state corresponds to the switching time of points and crossings. A train may only lock a point or crossing block if the block is in the right state for the train path. Hence, we might observe the following sequence of actions of train B to travel through some point block X after train A travelling over a conflicting route.

- free block X for train A
- lock block X for control
- consume block X for control
- free block X for control
- lock block X for train B
- consume block X for train B



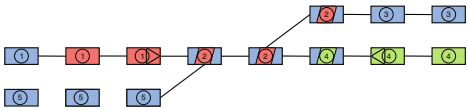
The final stage is to introduce the notion of a route as a sequence of blocks. Once the individual blocks of a route are locked, an actor may lock the route made of the locked blocks. Train head movement has a dedicated action for switching between two routes.

While a train actor is forced to inspect a route locking state, it is also directly inspecting the state of blocks in front. To make the behaviour localised we introduce conditions defining when a locked route may show one of proceed aspects. It is then formally proven that inspecting route state alone is sufficient to ensure train safety.

Definition 3 (Move train head onto a next route). *Localised version.*

$move\_head\_new\_route \triangleq$
any h, i, j, nr, t **where**
$t \in \mathrm{dom}\,(train\_seq)$
$h = train\_seq\,(t)\,(train\_seq\_head\,(t))$
$j = \mathsf{line}\,(train\_line\,(t))^{-1}\,(h)$
$i = \mathsf{line\_routes}\,(train\_line\,(t))^{-1}\,(train\_head\_route\,(t))$
$i + 1 \in \mathrm{dom}\,(\mathsf{line\_routes}\,(train\_line\,(t)))$
$nr = \mathsf{line\_routes}\,(train\_line\,(t))\,(i + 1)$
$route\_aspect\,(nr) \geq \mathsf{PROCEED}$
then
$train\_seq\_head\,(t) := train\_seq\_head\,(t) + 1$
$train\_seq\,(t) := train\_seq\,(t) \cup \{(train\_seq\_head\,(t) + 1) \mapsto \mathsf{line}\,(train\_line\,(t))\,(j + 1)\}$
$locking := \{\mathsf{line}\,(train\_line\,(t))\,(j)\} \lhd locking$
$train\_head\_route\,(t) := nr$
$route\_locking := \{train\_head\_route\,(t)\} \lhd route\_locking$
$route\_aspect\,(nr) := \mathsf{STOP}$
end

The diagram below shows blocks numbered with route indices. In reality, the same block may be attributed to several routes.



The model is still fairly abstract but covers all the essential aspects of safety principles: we prove freedom of collision and derailment. We have not considered many advanced properties such as absence of deadlocks, provision and treatment of overlaps, flank protection and etc. These may be introduced in next refinements of increasing fidelity.

3 Reading Station Dataset

The testing ground for the technique is a simulation data set provided by Siemens Rail Automation UK. The data is made of roughly 12MB of XML and structured text files describing topology and signalling of Reading station with signalling split into three overlapping interlocking areas. The diagram in Fig. 1 provides an indication of the scale of the studied data. The diagram was rendered directly from a subset of the data which include the visual layout for tracks.

One immediate issue was that a part of the data is not XML but a proprietary text-based format called LDL (originating at Invensys Rail). The SafeCap Platform has an import facility for LDL files but this silently ignores unknown data fields. We thus developed a new, more basic import tool that treats XML-based and LDL-based data on the same footing of an abstract relation-based data representation.

There is a considerable overlap between various parts of this data set. Many of them are not trivial to spot and for historic reason same elements are sometimes known under differing names. In addition, no provision for distinguishing between sets of elements and a sequence of elements. A strict interpretation would require regarding any multiplicity as a sequence or a tuple rather than a set. This is inefficient from the verification viewpoint. To counter this, we allow a user to manually demote sequence and tuples types into set types. For the case of tuples, a unified type (which might not exist for incompatible types) is used.

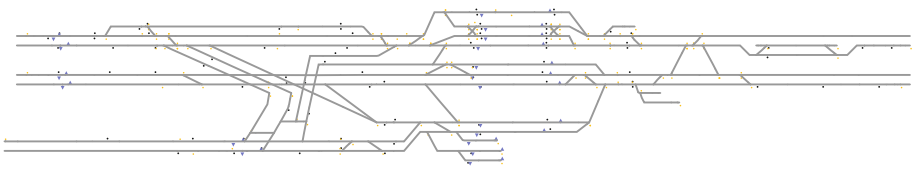


Fig. 1. Rendering of Reading Station *track layout data* as a track schema from a subset of the dataset (produced by SafeCap Platform). Black and orange (light gray) circles are signal and fixed speed limit positions; triangles denote train stopping points.

3.1 Reading Station Verification

The formal model presented above is used to validate Reading St. data. There is no simple correspondence between the data structures used in our formal abstraction and the real-life data characterizing Reading St. Yet some correspondence is bound to exist since both define, in their own way, a railway schema and route-based signalling.

Our verification technique consists in matching a data set against the assumptions of formal behaviour model (such as, for instance, given in Definition 1. This means there is no danger of state explosion and verification is comprehensive and conclusion, when it can be reached, is definite.

In the source dataset, there is a significant amount of duplication. Some cases are not trivial to spot and for various reasons same elements may, at times, be known under differing names. At the structural level there is no indication whether a collection of elements is a sequence or a set. A strict interpretation necessitates sequences at all times but this make it harder to write and check verification conditions. When data is imported, all such cases are treated as sequences and user can do one of two things: manually demote type to set, or request that there should be a separate, set based view of the same data. Thus, for instance, relation $r \in A \to \text{seq}(B)$ can be replaced by some $r' \in A \to \text{pow}(B)$ or r and r' may be present at the same time together with an axiom statement $r' = r; (\lambda t.\ \text{ran}(t))$.

The process of verification consists in positing a conjecture and checking it by combing with the data model to see if a contradiction arises. For instance, to check no two train detection circuits overlap we can state the following:

$$
\begin{aligned}
P_1 := [\forall\ & t, s, a, x, y, b, i, j\ . \\
& t : \text{TrackSection} \text{ and } s : \text{TrackSection} \text{ and } t \neq s \text{ and} \\
& a \mapsto x \mapsto y : \text{ran}(\text{“TRACK\_CIRCUIT.M\_SECTION”}(t)) \text{ and} \\
& b \mapsto i \mapsto j : \text{ran}(\text{“TRACK\_CIRCUIT.M\_SECTION”}(s)) \text{ and} \\
& a = b \\
& \quad \Rightarrow x > j \text{ or } y < i]
\end{aligned}
$$

Here “INTERLOCKING.M_SECTION” is a function name defined in a data source (detailed station topology). It is taken into quotes to escape characters clashing with operator syntax. Expression ran(“TRACK_CIRCUIT.M_SECTION”(t)) defines all pieces of a graph defining the sub-graph of a train detection circuit. This gives a set of track names. The condition checks that any two distinct train detection circuit t and s do not physically overlap.

The statement is conjoined with the mathematical model of source data $\mathbf{H}$ to form conjecture $\mathbf{H} \vdash P_1$. The conjecture undergoes a conservative filtering to remove parts of data model $\mathbf{H}$ irrelevant to P_1 and form a less constrained model $\mathbf{H}_f$ where $\mathbf{H} \subseteq \mathbf{H}_f$. The typing information is removed and all the literal values are encoded as integers. The un-typing and coding process has fairly modest impact on proof success per se but without it some tools cannot ingest and parse otherwise typically a very large input file. Every condition is checked twice - once in the positive (i.e., as given) form and once in the negative form.

Both cases must have a definite answer (that is, *unknown* result for either case renders the whole condition *false*) so that a conjecture is assumed to hold only when its positive is truth and the negative is false. The double check addresses potential well-definedness problems such as applying a relation outside of its domain or having self-contradictory data model.

When a conjecture of the form $\forall x.P(x)$ is found to be false it is, at times, possible to obtain a witness for $\exists x.\neg P(x)$ from the verification back-end. And for certain types of expressions (sequences and sets and single elements of signals, routes, tracks, points, etc.) a counter-example may be visualised on a track diagram.

3.2 Verification Results

We went through all the conditions (47 total) of track topology assumptions from Definition 1. In the process we have found that one condition does not hold:

> Condition (18) of Definition 1 states that edge (sink) blocks may not be points or diamond crossings. However, we found a counter-example: track $UpReaWestC$.

The majority of verification load is concerned with routes and signalling rules. The data set does not define possible train paths but defines routes. The analysis revealed a fair number of broken conditions but nearly all of these turned out to be due to cutting of signalling data across interlocking area boundaries. A simple aggregation of data leads to basic well-formedness problems, i.e., same entity is defined twice. But throwing out overlapping data seems to produce a number of validation errors. For instance

> Mappings between track circuit and a sub-route must agree in both directions:
>
> $$\forall r.r \in \text{“ILTrackSectionControlTables.SubRoutes”} \Rightarrow r \in \text{“ILSubRouteControlTables.TrackSection”}^{-1}$$
>
> There is a number of counter-examples.

and also

> In a control table, track circuits locked that must be locked for a point must be among the required track sections of a route. It is an essential safety conditions and is rendered as the following property.
>
> $$\text{id}(\text{dom}(\text{“ILRouteControlTables.NormalPoints”})) \subseteq (\text{“ILRouteControlTables.NormalPoints”}; \text{“ILPointsControlTables.NormalLockingTrackSections”}; \text{“ILRouteControlTables.TrackSections”}^{-1})$$

It does not hold for a number of cases when a route goes across interlocking boundary.

There are six more issues that seems to arise due to basic data completeness and consistency issues. This will need further investigation.

All in all, we have checked the data against 72 verification conditions and the vast majority of the conditions were discharged. All the conditions were handled completely automatically by a collection of theorem provers and constraint solvers. It takes less than 2 min to go through all the conditions for the whole station.

The majority of unsatisfied conditions exhibit the same pattern of incomplete definitions and seem to be stemming from the issue of splitting and then reassembling signalling data for the three interlocking areas.

4 Discussion

Perhaps the most prevalent validation technique in the railway industry is simulation. Simulation engines range from coarse-grained time stepping of a national railway network to a detailed model of various aspects of mechanical performance of specific rolling stock over certain track. Validation concerns span from the analysis of digital communication protocols connecting trains and regional control to stressing of tunnels and bridges by passing trains. Simulation is widely applied for time table optimisation and interactive 3D simulation is sometimes used for driver training. RailSys [12] and OpenTrack [10] are two of the more well-known simulation suites applied in time table optimisation and general analysis of signalling performance.

The main attraction of simulation is that it does not require deep understanding of railway functioning. Simulation tools present many aspects of railway performance in an intuitive, visual manner helping to quickly obtain the big picture of overall layout and signalling performance. There is, however, no guarantee of safety as simulation can only ever consider a tiny proportion of all scenarios.

The safety challenge of railways and the fact that collision and derailment properties may be dealt with within the setting of discrete, inertia-less train movement makes railway safety verification especially appealing for formal method practitioners. The principal idea of railway model checking is quite simple: a model of train movement laws is combined with the definitions of track topology and signalling rules. A model checking tool attempts to go through all or many execution scenarios to confirm that unsafe scenarios are ruled out. The list of modelling notations used in this setting is practically endless. Notable examples include Coloured Petri nets [2], process algebra CSP [5], a continuation work based on the model-based notation ASM [6], an algebraic language Maude [3] and the B Method together with ProB model checking tool [9]. The latter can also be used in the capacity of a property verifier for assertions written against B or Event-B contexts. In this form ProB has been used for the validation of

railway datasets [8] and this led to the development of a commercial toolset [11]. Our approach differs by the kind of properties we try to prove (safety principles of signalling) and the provenance of verification constraints (an Event-B model of signalling).

Model checking imposes limitations on the model size and performs best with a relatively limited logical language. Theorem proving overcomes these limitations and offers potentially unlimited opportunities for verifying safety with the utmost level of rigour. Theorem proving is not necessarily an all-manual process: there is a large and successful community developing automated theorem provers [13]. At the moment, automated prove support is best in the domain of first order logic and set theory; an attempt at reasoning about continuous train dynamics is likely to require an intervention by a highly skilled verification expert - the kind of people mostly found in academia.

Theorem proving, even with excellent tool support, requires a high level of expertise in formal verification and mathematical modelling. The semantic gap between logic and railway concepts is formidable. This leads to generally low productivity (but we should notice efforts like the BART tool for automatic refinement of B models [7]), difficulties in interpreting tool feedback, and posing verification statements in a manner convincing to a non-expert reviewer.

References

1. Iliasov, A., Lopatkin, I., Romanovsky, A.: Practical formal methods in railways - the SafeCap approach. In: George, L., Vardanega, T. (eds.) Ada-Europe 2014. LNCS, vol. 8454, pp. 177–192. Springer, Heidelberg (2014)
2. Janczura, C.W.: Modelling and Analysis of Railway Network Control Logic using Coloured Petri Nets. PhD thesis, School of Mathematics and Institute for Telecommunications Research, University of South Australia (1998)
3. Hagalisletto, A.M., Bjørk, J., Chieh Yu, I., Enger, P.: Constructing and refining large-scale railway models represented by Petri Nets. IEEE Trans. Syst. Man Cybern. Part C **37**, 444–460 (2007)
4. Iliasov, A., Romanovsky, A.: SafeCap domain language for reasoning about safety and capacity. In: Pacific-Rim Dependable Computing Conference (PRDC 2012), Niigata, Japan. IEEE CS, November 2012
5. Winter, K.: Model checking railway interlocking systems. In: Proceeding of the 25th Australian Computer Science Conference (ACSC 2002) (2002)
6. Winter, K., Robinson, N.: Modelling large railway interlockings and model checking small ones. In: Proceeding of the Australian Cumputer Science Conference (ACSC 2003) (2003)
7. Burdy, L.: Automatic refinement. In: Proceedings of BUGM at FM 1999 (1999)
8. Lecomte, T., Burdy, L., Leuschel, M.: Formally checking large data sets in the railways. CoRR, abs/1210.6815 (2012)
9. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003)
10. OpenTrack simulator. http://www.opentrack.ch/

11. Abo, R., Voisin, L.: Formal implementation of data validation for railway safety-related systems with OVADO. In: Counsell, S., Núñez, M. (eds.) SEFM 2013. LNCS, vol. 8368, pp. 221–236. Springer, Heidelberg (2014)
12. RailSys simulation platform. http://www.rmcon.de
13. TPTP. Thousands of Problems for Theorem Provers. www.tptp.org/