Practical Verification of Railway Signalling Programs

Alexei Iliasov, Dominic Taylor, Linas Laibinis, and Alexander Romanovsky

***Abstract*—SafeCap is a modern toolkit for modelling, simulation and formal verification of railway networks. This paper discusses the use of SafeCap for formal analysis and automated scalable safety verification of solid state interlocking (SSI) programs – a technology at the heart of many railway signalling solutions around the world. The main driving force behind SafeCap development was to make it easy for signalling engineers to use the technology and thus to ensure its smooth industrial deployment. The unique qualities and the novelty of SafeCap are in making the use of formal notations and proofs fully transparent for the engineers. In this paper we explain the formal foundations of the proposed method, its tool support, and their successful application by railway companies in developing industrial signalling projects.**

***Index Terms*—Formal modelling and verification, railway signalling, system safety, safety properties, proof conjectures**

I. Introduction

Effective signalling is essential to the safe and efficient operation of a railway network. It enables trains to travel at high speeds, run close together, and serve multiple destinations. Whether by mechanical semaphores, colour lights or electronic messages, signalling allows trains to move only when it is safe for them to do so. Signalling locks moveable infrastructure, such as the points that form railway junctions, before trains travel over it. Furthermore, signalling often actively prevents trains travelling further or faster than is safe and sometimes even drives the trains. At the heart of any signalling system there are one or more *interlockings*. These devices constrain authorisation of train movements as well as movements of the infrastructure to prevent unsafe situations arising. Solid State Interlocking (SSI) technology, developed in the UK, was one of the first computerised interlockings worldwide. The SSI devices use the 2-out-of-3 redundancy to support their safety critical functionality. Programs executed on SSI processors are responsible for safe authorisation of the train and infrastructure movements.

The increasing complexity of modern digital interlockings, both in terms of their geographical coverage and that of their functionality, poses a major challenge to ensuring railway safety. This calls for application of rigorous methods, in particular formal methods, for assurance and verification of safety and other crucial properties of such systems. Even though formal methods have been successfully used in the railway domain (e.g. [1], [2]), their industry application is scarce. In spite of a large body of academic studies addressing issues of formal verification of railway systems, they typically remain an academic exercise due to a prohibitive cost of initial investment for their industrial deployment. The reasons for that are the following ones. First, signalling engineers need to learn mathematical notations to apply them. Second, the tools often cannot be applied for analysing large real stations due to their poor scalability. Third, the companies need to drastically change their existing development processes in order to use them.

The paper's contributions are twofold. First, this paper proposes a formal tool-based approach that addresses the above issues by (i) verifying the signalling programs and layouts developed by signalling engineers in the ways they are developed by industry, (ii) ensuring automated verification of safety properties that uses a customised inference-based symbolic prover and fully eliminates the need for manual proofs, and (iii) providing diagnostics in terms of the notations used by the engineers. All together, this ensures that the developed method and tool can be easily deployed to augment the existing development process in order to provide extra guarantees of railway safety.

Second, the latter part of the paper discusses the successful application of the tool in railway signalling projects in the UK. During this work, conducted in cooperation with major signalling companies in the last 3 years, a number of real datasets have developed for many signalling areas in the UK railway network has been verified and the results of the verification have been passed to the companies to help them to improve the safety of their designs and to meet the country safety regulations. Overall, the paper builds on the results presented in [3] by introducing the full flow of data processing conducted by our verification framework and by providing substantial insights about its practical applications.

The paper is structured as follows. Section 2 presents the problem area and the motivations for verification of railway signalling. Section 3 overviews the state of the art verification methods developed for the railway domain with the focus on signalling, and their limitations. In Section 4 we present the SafeCap framework and discuss in detail its automated process for verifying that railway signalling data satisfy safety properties. Section 5 presents the industrial application of SafeCap for verifying signalling projects and illustrates the

Alexei Iliasov is with The Formal Route Limited, 32A Woodhouse Grove, E12 6SR, London, UK (e-mail: alexei.iliasov@formal-route.com).

Dominic Taylor is with Systra Scott Lister, UK (e-mail: dtaylor@systra.com).

Linas Laibinis (the corresponding author) is with Institute of Computer Science, Vilnius University, Didlaukio 47, LT-08303 Vilnius, Lithuania (e-mail: linas.laibinis@mif.vu.lt).

Alexander Romanovsky is with The Formal Route Limited, 32A Woodhouse Grove, E12 6SR, London, UK, and also with School of Computing, Newcastle University, NE1 7RU, Newcastle upon Tyne, UK (e-mail: alexander.romanovsky@newcastle.ac.uk).

practical advantages of the presented automated methods. Finally, Section 6 concludes the paper by summarising the proposed verification methods and the achieved practical results, and discussing the ongoing and future work on improving SafeCap.

II. Railway Signalling and Safety

A. Railway safety principles and standards

There are two main safety principles shared by all signalling systems. These ensure that the railway systems are designed in such a way that they do not allow train collisions or derailments. From these a large number of lower level signalling principles can be derived, which consequently become verification conditions to check against given signalling data.

A schema must be free from collisions. A collision happens when a train occupies the same physical space as another train or (at a level crossing) a road vehicle. Signalling systems uphold this principle through the use of signalling *routes*, and block sections.

A *route* is a defined path between two geographic locations on a railway. These locations may be marked by physical *signals* with lamps, signs or buffer stops or may be unmarked. A train is only given permission to enter a route when no other vehicles are authorised to travel over any part of that route in a different direction to that train. Permission is also only given when no other vehicles, having previously received such authorisation, may be unable to stop before travelling over part of the route.

For a route to be locked, all the movable equipment such as *points* or level crossings must be set and detected in a position that would let a train safely travel along the route path. They must remain locked in such a state and their position must be positively confirmed before a train enters the route.

In specific circumstances, trains can be authorised to enter routes that are already occupied by another train that is at a standstill or one that is moving in the same direction. Whilst the "*free from collisions*" principle can be upheld by the drivers controlling the train speed to stop before any vehicle ahead, this is only possible at very low speeds ($\leq$ 40 km/h). To enable high speed operation, the signalling system additionally needs to ensure safe separation of trains. It does this through signalling *block sections*: sections of railway track into which only one train is permitted to enter at once. A route can, and often does, consist of a single block section. In more modern signalling systems, routes can be sub divided into multiple block sections or even support moving block sections that follow trains as they move along the route path. Similarly, sections of a track, over which only one direction of travel is possible, can consist of single block sections, multiple block sections, or (in more modern systems) support moving block operation.

A schema must be free from derailments. A derailment may happen when a set of points moves underneath a train. To avoid this, points must be positively confirmed to be locked in position before a train may travel over them and held in that position as a train does so.

Derailments may also happen due to excess speed around a curve. Signalling systems are playing an increasing role in mitigating this risk, although this mostly falls outside the scope of *interlockings*, which is the subject of this paper. The role of interlockings is generally constrained to ensuring that the driver is presented with unambiguous information to determine a safe speed profile for the train. This information may be presented as the route that the train will take (*route signalling*), from which the driver can determine a safe speed profile, or the permitted speed for that route (*speed signalling*).

Any new signalling system or its alterations must demonstrate that the risk associated with collisions and derailments is adequately controlled in accordance with the Common Safety Method for Risk Evaluation and Assessment (CSM RA), described in [4] and [5]. CSM RA permits three principles by which risks can be accepted: *Application of a Code of Practice*, *Similarity with Reference System(s)*, and *Explicit Risk Estimation*.

Application of a Code of Practice is by far the most commonly used risk acceptance principle for interlockings. [6] is an example of a code of practice (standard) normally used for UK mainline interlockings. It also forms the basis for the safety properties analysed by SafeCap. However, there are cases where full compliance with the current standards is not appropriate. For example, where interlockings were developed prior to these standards or for different standards (such as on London Underground or High Speed One). In such cases comparison with a similar reference system may be more appropriate. Also, explicit risk assessment is needed where there is a strong operational or safety need to implement measures different to those described in the established standards. SafeCap identifies where such non-compliances occur, so that signalling design teams can ensure that appropriate alternative risk acceptance principles have been applied.

B. Computerised Signalling and SSI

The first signalling interlockings were mechanical devices that constrained the movement of levers, connected to points and semaphore arms, and were contained in mechanical signalling boxes. During the twentieth century these devices were superseded by electrical relay-based interlockings that switched electrical current to motorised points, colour light signals and other lineside devices. The safety conditions applied by relay interlockings included tests of track circuits or axle counters – the devices that automatically determine whether a route section has a train on it (a task previously achieved through operational procedures and signaller eyesight). The advent of computer technology brought the opportunity to replicate and build on the relay interlocking functionality within a computer based interlocking.

One of the earliest forms of computer based interlocking was the Solid State Interlocking (SSI)[7], developed in the UK in the 1980s through an agreement between British Rail (the then nationalised railway operator) and two signalling supply companies, GEC General Signal and Westinghouse. Running on bespoke hardware, SSI software consists of a core application (common to all signalling schemes) and site specific geographic data. SSI GDL (Geographic Data Language) data configures a signalling area by defining site

```
*QR117B(M)                    / route request block for route R117B(M)
  if R117B(M) a               / route R117B(M) is available
            USD-CA f,OSC-BA f,OSV-BA f   / sub-route and sub-overlaps are free
         then if OSL-AC l,      / sub-overlap is OSL-AC locked
                 P223 fr , P224 fr   / points P223, P224 free to move reverse
              then @P223QR \  / call subroutine P223QR
            if OSD-BC f       / sub-overlap is OSD-BC is free
               LTR04 xs       / latch (boolean flag) not set (false)
               P224 crf       / point P224 commanded reverse or free to move reverse
            then R117B(M) s   / set route set flag for R117B(M)
               USD-AC l , USC-AB l , USB-AB l , OSA-AB l  / set sub-routes/overlaps
               P224 cr        / command point P224 reverse
               LARR xs        / clear latch LARR
               S117 clear bpull   / clear signal button pull flag
               if P223 xcr , P223 rf then / check point states
                @P223QR   / point command subroutine
                   EP230 = 0 \ / reset timer EP230
```

Fig. 1. SSI example: route request code for route R117B(M) in the PRR module

specific rules, concerning the signalling equipment as well as internal latches and timers that the interlocking must obey. The original SSI has now been superseded by more powerful, modern hardware platforms running software developed in accordance with modern standards for safety critical software. Nonetheless, the functionalities of the core application and the SSI GDL language remain largely unchanged.

SSI GDL is, despite its name, a form of programming notation. The notation expresses computations transforming inputs, i.e., currently sensed equipment states, into outputs, i.e., commands affecting equipment state. There is also the internal state updated by computations and maintained between cycles. Thus, SSI is a continuously running control system.

There are two main modules defining the signalling behaviour – the route and point request (the PRR module) and formation of output telegrams (the OPT module). An example of route request code (a part of logic that reacts to an external route request) is given in Fig. 1. Notice that the parts between `if` and `then` are atomic predicates combined with implicit conjunction, while everything between `then` and a slash character is a command (made of a sequence of atomic commands). For instance, `USD-CA f` stands for *test that sub route* `USD-CA f` *is free*, while `USD-CA l` commands the sub route to be freed.

SSI systems runs a global loop made out of three stages: polling of inputs (the current states of track circuits, points, signals and so on), computation of necessary responses, as well as formation and transmission of equipment control commands. These stages are strictly sequential and each stage has a fixed time budget that is small enough to react in a timely manner to any state change but big enough to accommodate execution of all the code. The actual time limits are of no particular relevance to modern installations as modern hardware is orders of magnitudes faster that its historic predecessors.

The input and output stages are generic and their safety argument is provided once for a particular underlying hardware implementation. Verification of system safety is thus concerned with the middle stage – the response computation. This stage is unique to every geographic area and explicitly refers to the equipment drawn on the area *scheme plan* – a diagrammatic depiction of a railway. At a high level, the stage comprises a number of code blocks executed sequentially. Separation into blocks is semantically important as block boundaries are defining units of execution plus their naming has an effect on interpretation of the contained code.

One good abstraction for this part of SSI is a box with a number of buttons. A higher level system (e.g., an automated control system or a human signaller) can press one or more buttons to request certain functions to be performed. The system hidden inside the box may choose to block a request or execute it in differing ways in order to maintain safe operation of the overall railway. It is a typical example of a *safety kernel* [8].

Safety is the overriding concern for SSI signalling. Performance, fairness or liveness properties are typically not addressed explicitly although consideration for them does affect the design of interlocking logic.

III. Methods for Verification of Railway Signalling

There has been a long history of applying formal methods and tools in the railway domain [9], with the B method being the most popular industry-strength method [10]. One of the most successful areas of railway has been safety verification of interlocking in the Communication-Based Train Control (CBTC) railway lines, including metro, subway, light and shuttle trains. The mainline railway is a much more complex type of railway, this is why there have been fewer success stories in applying formal methods in this area. A number of research and technology transfer projects have been reported but, unfortunately, they rarely lead to a full take-up or deployment of these methods in industry.

The SSI preparation processes outlined in earlier work by Cribbing and Mitchell [11] fully rely on system testing, manual simulation and redundant human checks to ensure the safety of SSI signalling. These techniques still remain now, 30 years after the paper was written, the dominant approaches used in practice.

There have been a number of research studies focusing on formal verification of SSI programs [12], [13], [14], [15], [16], [17]. The majority of works (e.g., [12], [13], [14]) use various forms of model-checking in an attempt to verify safety of *train run scenarios*, with interlocking rules derived manually or via an automated translation from SSI data. With few exceptions, the proposed techniques actually scale up to only toy examples, or cover a small subset of functionalities, or both. For instance, the approach presented in [14] uses NuSMV to model check a small subset of safety properties for a selected

subset of SSI data based on real-life signalling data. It is not clear whether expanding the technique from the current five state variables to nearly fifty ones necessary to capture the full SSI is something that the underlying verification technology can support. The approach presented in [17] proposes an approach to reducing the states to be explored during the model-checking that relies on adding extra information about the system to be verified. However, the practical applicability of such an approach has not been demonstrated in real settings.

The existing work typically focuses on verifying reachability and liveness properties. We believe that, in the context of SSI verification, this is not the most optimal approach because most safety invariants associated with railway signalling interlockings are concerned with preventing unsafe combinations of states. Where time does feature in safety invariants, it is easily representable as logical tests of SSI timers, which ensure minimum time periods have elapsed between specified events. Interlocking response times are only of interest from a performance perspective and for the specific safety action of stopping trains in an emergency, both of which can be easily calculated from system cycle times without the need for formal verification or simulation.

In the face of sheer number of train run scenarios, one way to avoid the state explosion problem might be statistical simulation of train runs [15]. However, this approach has non-trivial implications on result interpretation. We see a fundamental flaw in all such scenario exploration techniques: by introducing train runs and assuming certain traffic patterns, they cannot find, even if they were to scale up, serious signalling mistakes that do exist in real-life implementations and only manifest themselves when a combination of several rare conditions happen (see, for example [18]). Our approach does not suffer from this limitation as we do not need to consider train runs (and thus limit verification to few assumed possibilities). Instead, we check the worst case safety implications for all possible (and infinite, due to the implicit presence of the temporal domain) train run scenarios.

Another major problem with the solutions described above is their poor diagnostics related to the fact that the feedback on safety violations and the associated counter examples are not given in the terms that signalling engineers could understand (i.e., SSI and the schema language).

In [16], the authors build a model of railway operation constrained by imported signalling data. A model checker automatically explores train movement scenarios (i.e., model states) and reports on violation of safety properties. The technique does not support generic safety properties (which have to be written separately for a specific layout) and the reported results indicate it is unlikely to scale up to the industrial size.

In rare cases a railway model may be decomposed into two independent parts (i.e., two stations connected by auto signalled track) and it is sound to conduct verification of the sub parts separately [19], [20]. Such a technique is not often found in practice as SSI is traditionally limited to 64 or 256 controlled pieces of equipment and it is impractical to wire equipment at a significant distance from a control box.

Verification and validation of a fragment of *safety logic* for European Railways Train Management System (ERTMS), ensuring also interoperability of different signalling solutions, is described in [21]. ERTMS specifications (written in a structured programming language) are automatically translated into formats of the employed external verification tools. The paper [22] presents an ongoing work on automatic model generation and verification of Railway Markup Language (RailML) formatted data, which also include route tables and interlocking information. Interlocking programs are defined in RailML using route scheduling and route automata.

In [20], the authors present verification of railway interlocking (without relying on train run scenarios) based on model checking in NuSMV and NuXMV. To avoid the state explosion problem, [19] introduces a compositional approach, splitting interlocking into smaller components. Finally, in [23], a combination of theorem proving and model checking in the B framework is used for verification of railway data, focusing however on only railway topologies (including the signalling and interlocking equipment).

None of these approaches [19], [20], [21], [22], [20], [23] however could be directly applied for safety verification of SSI programs. Moreover, as opposed to our work, they all heavily rely on model checking techniques and tools for formal verification of railway safety properties.

To summarise, model checking is known not to scale to projects of the complexity of the typical SSI projects in industry. One of the smaller SSI interlocking we checked has more than 2000 variables and 6000 lines of code. Exploring even a tiny fraction of such a system is well beyond the capabilities of existing model checking techniques. In fact, we have not seen any successful application of such techniques for verification of real-life signalling projects. Moreover, it is difficult to expect signalling engineers to be able to apply complex decomposition, structuring or layering techniques to reduce the complexity of verification as a possible solution proposed in some work on SSI model-checking.

To conclude, the main limitations of the existing work in SSI verification are (i) the high cost of deployment (staff retraining, the disruptive nature of development changes), (ii) poor verification scalability insufficient for real signalling projects, (iii) the focus only on subsets of SSI, and (iv) the diagnostics given in terms of formal methods rather than in the standard notations (SSI and the layout schema) used by the signalling engineers.

IV. The SafeCap Verification Process

This section starts with introducing the general SafeCap framework under development since 2011 that served as an experimental prototype for the SSI SafeCap tool described in Sections IV.B-IV.C, which has been developed to target specifically industrial verification of the SSI interlockings. Section IV.B overviews the SSI SafeCap verification process and Section IV.C discusses the stages of this process in detail.

A. The SafeCap framework

The SafeCap platform is a toolkit for modelling railway capacity and verifying railway network safety [24]. It allows

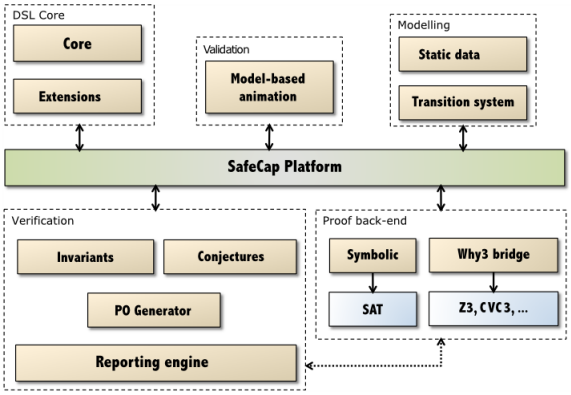

Fig. 2. SafeCap architecture

signalling engineers to design stations and junctions relying on the provided domain specific language (SafeCap DSL) and to check their safety properties, simulate train runs as well as evaluate potential improvements of railway capacity by using a combination of theorem proving, SMT solving and model checking [25]. The platform has been substantially extended by adding new simulators, solvers and provers, as well as the support for importing the existing designs in a wide range of the signalling frameworks supported by industry [26], [27]. The overall SafeCap architecture is presented in Fig. 2.

The SafeCap verification and proof back-ends enable automated reasoning about static and dynamic properties of railways or their signalling data. Our principal verification routes are the built-in symbolic prover backed by a SAT solver, accompanied by a range of external provers provided via the Why3 framework [28] and the ProB model checker [29] (used just as a constraint solver). A SAT solver (integrated with the built-in symbolic prover) is required to prove or disprove statements involving arithmetic and inequalities as these are usually too hard for rewrite-based techniques. The Why3 provers and ProB are used independently, in parallel to the built-in symbolic prover, to gain higher confidence in the verification results. In the work on general SafeCap framework we used a number of manually created, experimental or machine-generated representations of medium-size signallings (but without attempting to work with real industrial interlockings). This extensive experimentation demonstrated that the SafeCap approach works: it supports fully automated safety verification avoiding the bottlenecks of model-checking [26], [27].

The SafeCap DSL allows the designers to rigorously and unambiguously define a model of the given railway network (e.g., stations or junctions). In other words, the SafeCap DSL provides a formal, graph oriented way to capturing railway schemas and some aspects of signalling [30]. In the SafeCap DSL, a railway schema is a mathematical object consisting of data structure definitions (namely, datatypes and constants) as well as required logical constraints on the defined data (axioms and lemmata). We can distinguish two main parts of the SafeCap DSL – the Core and its various extensions.

The Core allows us to mathematically describe the physical topology of a railway schema (or, in fact, any graph-based structures). Its first-class concepts are graphs and subgraphs. These typically represent track topology, track circuits, routes or axle counters.

Once a railway schema model is created (or imported from an external format) in the Core, it is checked for its validity or well-definedness. For that, a number of graph theoretical statements are automatically generated and verified, including isomorphism properties between constituent subgraphs, path validity within a given graph, connectivity, acyclicity, node degree and so on.

Various concepts of a railway schema such as signals and signalling solutions, speed limits, stopping points and so on can be incorporated into via DSL extension plug-ins. Such plug-ins introduce new data (as custom annotations) and the supporting logic (as additional logical constraints or relationships). Such a tool architecture allows us not to commit to any regional technology and thus to offer a broadly similar approach for a range of legacy and current technologies.

The SafeCap framework was initially used as a prototype experimental proof-of-concept toolset for SSI safety verification. This allowed us to evaluate its applicability and at the same time to understand better the bottlenecks of this verification. In the last five years we have developed a dedicated industry-strength SafeCap toolset targeting SSI verification. During this transition we removed all the unnecessary functionalities (including animation and simulation) from SafeCap, streamlined its modelling/verification flows, and replaced all the solvers and provers with a customised inference-based symbolic prover that currently outperforms all state-of-the-art provers and solvers when applied for the specific SSI verification in the industrial project (our experiments showed, for example, that it takes Why3 20s and up to 12GB RAM to discharge one conjecture from a typical SSI project, whereas our customised prover needs 1 mins 40s and up to 70GB RAM to discharge 47230 proof obligations in the largest SSI industrial projects we completed by now). In addition to these modifications we have developed a general input notation for symbolic execution called Generic Verification Framework and an SSI plugin that parses the SSI inputs into this notation. The resulting SSI SafeCap toolset is discussed in the remaining part of the paper.

B. Overview of the SSI Verification Process

As a programming notation, SSI GDL is a typed language with declared variables storing equipment properties or states. Each variable carries its data type signature as a part of its name. For production deployment, SSI code is compiled into machine instructions. We only ever analyse systems that have at least successfully passed the compilation stage and hence their syntactic correctness is not a concern. Moreover, we typically verify SSI code that has passed through some testing and review and, in many cases, is close to final deployment.

The central question in the verification of signalling correctness is what constitutes a safe signalling design. Certain basic principles are universally accepted, for instance, the absence of train collisions and derailment. However, it is almost hopeless to verify the absence of such hazards in the strictest possible

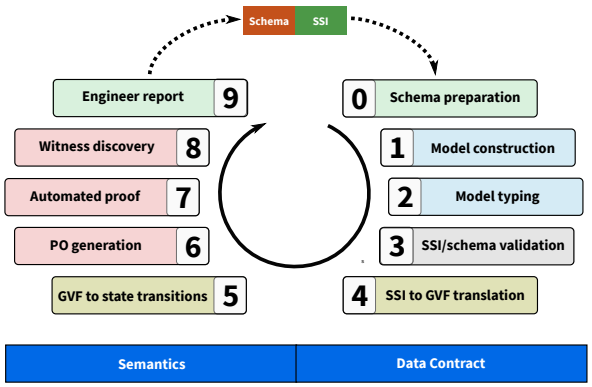

Fig. 3. SafeCap verification process

sense. First, in any operational railway there is a non-zero probability of something going wrong either due equipment failure or driver mistakes. Interlocking protects against a signaller error (or an error by an automatic route setting system) and, to a limited degree, mitigates driving errors and equipment failures. In normal operation, interlockings are pretty much comprehensive in protecting against such events, i.e., it is nearly impossible for the signaller to create an unsafe state through normal route setting commands. However, unsafe states can occur in degraded scenarios where the interlocking is overridden either through procedural instructions to drivers or (in more modern data) specific interlocking requests that bypass certain areas of functionality, such as proceed on sight routes or emergency point and route release.

Interlockings, by themselves, provide only a modicum of protection against driver errors through the provision of overlaps and control of signal aspects. This is an area where there is significant variation in practice across different geographic regions and times. Train warning and protection technologies ranging from AWS (Automatic Warning System), TPWS (Train Protection & Warning System) to ATP (Automatic Train Protection) or ETCS (European Train Control System) play a much greater role in mitigating the risk of driver errors.

Interlockings also do little to protect against equipment failures, but are themselves designed to be resilient to such failures, e.g., through the use of timers in sequential release of sub routes.

For these reasons, correctness is established not against the basic principles but rather against the lower level signalling principles derived from foundational safety principles and designed to enforce railway operation with an acceptable level of risk and failures. Such principles are carefully designed by domain experts but can vary between regions and do change over time.

For the purposes of verification, such a principle is rendered as an *inductive safety invariant* – a system property that must hold when a system boots up and must be maintained (or, equivalently, reestablished) after any state update. Verification is then understood as the problem of checking that any safety invariant is respected by every state update. Technically this is done be generating conjectures of the form "*if an invariant holds in a previous state and an certain state update happens, is it true that the invariant holds for the new state?*". Formally, a conjecture (also called a *proof obligation* (PO)) is represented as a logical sequent consisting of a number of hypotheses (H) and a goal (G), denoted as $H \vdash G$.

The number of such conjectures is $m * n$, where m is the number of safety invariants (we have defined 67 so far) and n is the number of possible state updates (for the industrial projects we have carried out this value varies between 4000 and 140000 with the mean at 17641). This is a small number when contrasted against the number of potentially reachable states (more than 2^{2000}: there are approximately 2000 boolean variables for each SSI interlocking and 50-200 one-byte unsigned integers; larger specimen use more than 5000 boolean variables). As the number of safety invariants is fixed for all projects, the complexity measured in the number of conjectures grows linearly with the interlocking complexity. The downside of safety invariant verification is the reliance on theorem proving. In our case, a conjecture is based on the first order logic, and, in a general setting, establishing its truth or falsity is an undecidable problem. In practice, however, it is possible to automatically prove or disprove vast majority of conjectures with only a tiny number (less than 1 in 10000 for recent results) of provable conjectures failing to prove and resulting in a false positive. Achieving such a level of proof automation is not easy and requires a number of bespoke tools and techniques.

The theoretical foundations of the SafeCap approach to verifying SSI are close to the earlier work by Ingleby and Mitchell [31], which outlines a general method for proving SSI program safety. Our approach differs in the formalisation style and the ability to automatically handle real-life interlockings, thus supporting fully automated scalable verification of real signalling projects.

C. The Verification Process in Detail

The major steps of our verification process are depicted in the diagram in Fig. 3.

There are four major stages: preparation of input (steps 0-3), translation and symbolic execution of the signalling logic (steps 4-5), formal verification (steps 6-8), and report generation (step 9). All but the very first stage are automated. Input preparation turns an electronic image of a scheme plan into a mathematical model; symbolic execution yields a large number of individually simple state transitions; verification checks that these state transitions are safe; and the generated report presents findings of safety violations.

The associated diagram in Fig. 4 depicts the overall data flow of the approach. The corresponding data transformation steps (arrows) on this figure are annotated by numbers referring to the respectively numbered steps of Fig. 3. Dashed arrows show manual steps, whereas non-dashed ones represent automatic steps executed by SafeCap. Intuitively, the left branch focuses more on the system statics (railway scheme, its various constituent elements and their relationships), while the right one deals with the system dynamics related to SSI signalling. The middle branch covers industrial standards, formulating safety principles relevant to the scope and operation level of SSI and then deriving formal safety invariants.

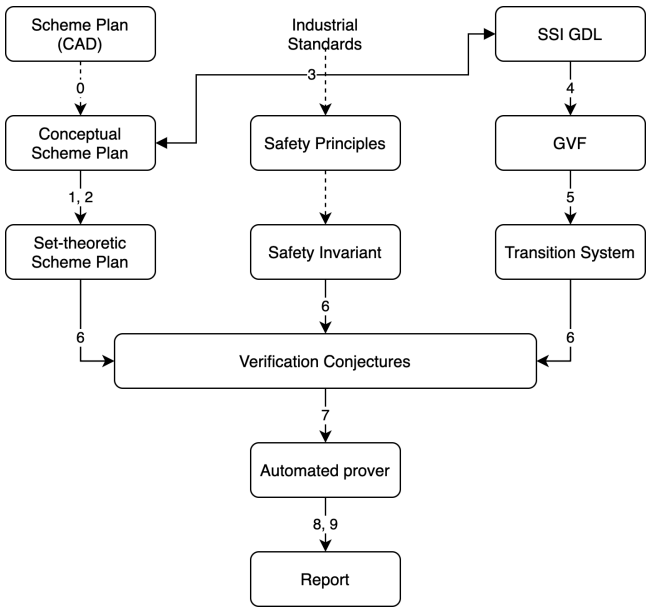

Fig. 4. Data flow diagram

In general, each subsequent layer downwards adds more rigour and formality. The data representations of the third layer (i.e., set theoretical scheme plan, safety invariant, and transition system) are based on the same underlying mathematical language and can be considered as parts of the overall formal system model ready to be verified.

At the top level, the inputs defining a particular area are a relevant scheme plan and the associated SSI GDL source code. They both are translated into the corresponding mathematical models in two steps. The intermediate representations, a conceptual scheme plan and GVF (Generic Verification Framework), are needed for practicality reasons, in particular, to shield against idiosyncrasies of a particular input notation and provide a generic verification approach. The two formal models derived from the scheme plan and SSI GDL are interrelated using the derived safety invariants, which leads to generation of necessary verification conjectures. Any violations of the safety invariants are reported as findings in the final report. Note that solid arrows in Fig. 4 depict automated actions by a tool, while dashed ones are manual activities.

a) **The conceptual and set theoretical scheme plans (steps 0 - 3):** A railway is typically described by a scheme plan or a signalling plan, which are two forms of a diagrammatic representation of a railway emphasising details pertaining to safety control. A standard engineering approach is to prepare such plans as drawings in a CAD tool. To enable formal reasoning about a certain railway area, we construct a semantic representation of a railway based on annotated graph [30], called *a conceptual scheme plan*. The SafeCap framework provides a dedicated graphical tool for reconstructing railway layout in such a format. A conceptual scheme plan captures the information already present in a scheme plan drawing and intelligible to a signalling engineer.

Control tables are widely used in railway for converting specifications of the signalling requirements to a form which can be applied during the design [32]. So their development precedes and feeds into the design of SSI signalling. We do not use these in the approach presented as these tables contain little additional information to that of a conceptual scheme plan and are also CAD drawings. In principle, however, one might choose to verify the given signalling data against the corresponding control tables. One critical downside of that is the complete absence in control tables of visual intuition and hints provided by a scheme plan. Thus it would be very hard to decipher and act upon reported violations without an explicit reference to the original scheme plan. Another drawback is that a control table is produced manually from a scheme plan and might itself introduce difficult to detect errors.

One apparent weakness of relying on a manually constructed conceptual scheme plan is the possibility of introducing mistakes (as compared to the original, CAD-based plan) that would mask errors in the verified SSI data. We use SSI data itself to do an automated early health check of a conceptual scheme plan (step 3 in Fig. 3). For instance, all variables declared in SSI that correspond to the scheme plan equipment (such as routes, points and signals) must be present in the conceptual scheme plan as well. There are a number of other checks and all of them help to rule out certain kinds of mistakes in a conceptual scheme plan.

We employ manual cross-review of the constructed conceptual to ensure its correctness. A further layer of assurance is the verification process itself as most of mistakes in preparation of a conceptual plan manifest themselves as serious safety errors. Hence, while it is not impossible that a conceptual scheme plan mistake remains uncovered and masks a real error in SSI data, we believe the process we have in place makes this unlikely. Note that we rely on the input scheme plan to faithfully depict reality and generally do not attempt to find and correct any errors in it.

A conceptual scheme plan is an intermediate representation for the construction of a *set theoretic railway model*. The resulting mathematical model consists of a number of constant sets, functions and relations formally representing different elements and aspects from a railway scheme plan. For instance, a route (i.e., a path between some two signals), is qualified by its entrance and exit signals, a list of sub routes, and a number of other properties. In a set theoretic railway model, the entrance signals of all routes are represented as a constant function *route.entrance* with the type $\mathrm{Route} \nrightarrow \mathrm{Signal}$ ($\nrightarrow$ is a symbol used to denote a partial function) and some constant set value $\{R_1 \mapsto S_1, R_2 \mapsto S_2, \dots\}$ defining specific mappings between routes and signals.

All elements of a mathematical railway model must be well-typed where a type is a *maximal set* of possible values, such that it does not intersect with any other type. Types of railway elements such as signals and routes are constant sets of listing all defined signals and routes. The typing process then ensures that various entities are used in a type-safe manner. A further step to the typing process is demonstrating that totality, functionality and injectivity constraints of various relations are satisfied by a given concrete mathematical model. For instance, a route entrance is constrained to be a partial function – a relation that maps some routes to one signal. Hence, one

can use the functional application operator to find entrance signal of a route although albeit with a side condition (a well-formedness conjecture) that application value is in the domain.

Set theory gives a great deal of expressiveness in formulating various statements about railways. Writing, for instance, *route.entrance*(R_1) would give us the entrance signal of route R_1 (equal to S_1). We can also reference all the routes with some entrance signal as dom(*route.entrance*) (i.e., the domain of a function or a relation or a relation), all the entrance signals for some route as ran(*route.entrance*) (i.e., the range of a function or a relation), all the routes from a given signal s as *route.entrance*$^{-1}[\{s\}]$ (i.e., the relational image of a concrete set for an inverse function) and so on.

b) **Generic Verification Framework, a state transition system, and safety invariants (steps 4 - 6):** The notation, called Generic Verification Framework (GVF) language, provides a simple imperative representation of a program with familiar control flow constructs such as subroutines, conditional statements, and variable assignment. A loop statement is also supported but must be accompanied by a loop invariant expression. Fortunately, there are no loops in SSI. By using GVF as an intermediary notation, we can support the same translation back end for a wide range of input notations.

SSI is a simple imperative language and its translation into GVF presents a few challenges. There are a number of intricate points related to the order of execution, the global loop, and the usage of timer variables, however addressing these issues requires a dedicated paper. Also, as it is common for symbolic execution, there is always a danger of exponential explosion in the number of potential execution paths. However, knowing a safety invariant to be preserved gives us an ability to combine or drop some paths at the translation stage without introducing potential false positives. As one example, a block of code containing 103 `if` statements is translated into only 203 state transitions, while the overall number of potential execution paths is 2^{103}.

To verify properties of the GVF representation of SSI, we further translate GVF into a form suitable for verification. This step is a form of symbolic execution, translating a GVF model (program) into a formal representation of a state transition system.

The translation step is performed using one of two well-known formal semantics – the strongest post condition semantics (spc) or the weakest precondition semantics (wpc) [33], [34]. For terminating program constructs, those semantics are logically equivalent. The default choice is spc as the translation is independent of a safety invariant and can be combined with any safety invariant goal.

In case of wpc, there is necessarily one unique state transition system for each safety invariant. It means calculation of a transition system must be repeated many times but each one is normally more compact as it is tailored for a specific invariant.

One typically sees between 8 and 20 thousands of state transitions after translating input SSI programs into a state transition system. To check their validity, we need to generate and prove a collection of logical conjectures (POs).

To properly describe such a conjecture, we must introduce some preliminary definitions. A state transition is characterised by a pre-condition predicate $P(c, v)$ expressed over constants c (elements of the conceptual scheme plan) and model variables v, representing the system state; and a post-condition $Q(c, v, v')$ relating a next state v' from some current state v and constants c. The whole transition system is an indexed set (P^J, Q^J) of such pre- and post-condition pairs, where J represents a set of all state transitions.

The following is an example of a state transition derived from the SSI code excerpt in Fig. 1. This transition captures one of eight possible execution paths for the source route request.

$$
\begin{array}{l}
\textbf{pre} \\
\quad \mathsf{QR117B(M)} \in \mathit{request} \wedge \\
\quad \mathsf{R117B(M)} \in \mathit{route\_a} \wedge \\
\quad \mathsf{USD\text{-}CA} \in \mathit{subroute\_l} \wedge \\
\quad \mathsf{OSC\text{-}BA} \in \mathit{suboverlap\_l} \wedge \\
\quad \mathsf{OSV\text{-}BA} \in \mathit{suboverlap\_l} \wedge \\
\quad \mathsf{OSL\text{-}AC} \notin \mathit{suboverlap\_l} \wedge \\
\quad \ldots \\
\textbf{post} \\
\quad \mathit{route\_s}' = \mathit{route\_a} \cup \{\mathsf{R117B(M)}\} \\
\quad \mathit{subroute\_s}' = \mathit{subroute\_s} \cup \{\mathsf{USD\text{-}AC} \wedge \mathsf{USC\text{-}AB}, \mathsf{USB\text{-}AB}\} \\
\quad \ldots
\end{array}
$$

Let us denote the set theoretic scheme plan as $M(c)$ and the axioms constraining the constants c and model state v as $A(c, v)$.

In our formalisation, a safety invariant is expressed over the previous state $'v$, current state v and constants c: $I(c,'v, v)$. Explicit referencing of the previous and current states in an invariant helps with clarity and terseness of safety conditions and offers extra expressive power. An alternative, not available to us as formal models are automatically constructed from the given GVF representation, is to use auxiliary model variables explicitly capturing the previous state.

An example of a safety principle encoded in a safety invariant is the absence of derailment due to moving a point under a train: *whenever points are commanded into a new lie all train detection sections over those points are proven clear*. In our formal notation, this statement takes the following form:

$$
\begin{array}{l}
\forall\, p \in \mathsf{Point} \cdot \\
\quad \mathit{point\_c}'(p) \neq \mathit{point\_c}(p) \\
\quad \Rightarrow \\
\quad \mathit{point.sections}[\{p\}] \cap \mathit{track\_o} = \varnothing
\end{array}
$$

Here Point is the set of all points, $\mathit{point.sections}$ is a constant relation defined by the set theoretic scheme plan and relating the track sections with railway points, while $\mathit{track\_o}$ and $\mathit{point\_c}$ are model variables modelling track section occupation and point states respectively.

c) **Proving verification conjectures (steps 7 - 9):** Combining all these above definitions, a schematic conjecture for the preservation of a safety invariant (for a state transition $j \in J$) takes the following form:

$$
\begin{array}{l}
M(c) \wedge A(c, v) \wedge I(c,'v, v) \wedge P_j(c, v) \wedge Q_j(c, v, v') \\
\Rightarrow \\
I(c, v, v')
\end{array}
\tag{1}
$$

In practice, it makes sense to consider a safety invariant as a collection (conjunction) of several verification properties

(simpler safety invariants) $I = I_1 \wedge I_2 \wedge \ldots I_n$, which allows us to reformulate or split (1) into:

$$\begin{aligned} & M(c) \wedge A(c,v) \wedge I(c,'v,v) \wedge P_j(c,v) \wedge Q_j(c,v,v') \\ \Rightarrow & \qquad (2) \\ & I_k(c,v,v'), \end{aligned}$$

where $1 \leq k \ \wedge \ k \leq n$.

One of advantages of the conjecture scheme (2) is that it permits immediate filtering of the vast majority of potential conjectures on the basis of $Q_j(c,v,v')$, constraining only a subset of new states v' that might be of relevance to the part in $I_k(c,v,v')$.

Typically, after such a simple filtering, we tend to get 2 to 3 conjectures for every state transition. Obviously, there is no even remote possibility of applying interactive proof at such a scale. All these conjectures must be dealt in an automatic manner with only limited degree of project-level configuration and fine tuning. The latter is achieved via the use of *proof tactics* – scripts (heuristics) describing specific preferable application of particular proof steps (inference rules).

Let us look closer at (2) to see how it can be approached by a theorem prover. As mentioned above, $M(c)$ stands for a definition of the complete theoretic scheme plan. As such, it is very large, consisting of hundreds of constants, some of which are defined with hundreds of mappings. Just pretty printing $M(c)$ into a notation used by Why3 or B results in a very large model that can take minutes to parse. Moreover, Why3 typically gets so overwhelmed by large constant sets that it cannot really progress any further.

Our solution is to hide the complexity of $M(c)$ behind a black-box simplification rule, which automatically simplifies constant expressions occurring in the goal and hypotheses by substituting them according to the corresponding definitions from M. For instance, for some constant r, the expression *route.entrance*$[\{r\}]$ would be simplified to a singleton set containing the route r entry signal.

The axioms $A(c,v)$, however, do not present much difficulty. There are only a few dozens of them and most of them are written in a set theoretic rather than a predicate form, thus avoiding use of quantifiers. We purposely avoid a predicate form in axioms as it normally requires using quantifiers. Quantifiers are expensive for an automatic prover as they can be used to generate a huge number of new hypotheses, thus delaying any meaningful progress.

The predicate $I(c,'v,v)$ stands for an overall safety invariant. It is typically a conjunct consisting of some 50 predicates, with many of those using nested quantifiers. It presents an inexhaustible source of rewrites for a prover, in most cases preventing any useful progress. Hence $I(c,'v,v)$ is another part that needs to be changed in order to make proof automation feasible.

Our solution is to remove $I(c,'v,v)$ completely and manually approximate the effect of $I(c,'v,v)$ with a number of pre-defined inference rules (i.e., rewrite rules acting upon a whole conjecture). The point of doing this manually is to have the smallest possible set of inference rules providing sufficient proof power. Formulation of such inference rules is done once as a part of prover configuration.

The pre-condition $P_j(c,v)$ is typically a conjunct of 5 to 40 predicates. One potential factor limiting proof progress is the possibility of having several large (10 to 40 or event more terms) disjunctions in $P_j(c,v)$, unfortunately not too uncommon in applications to railway signalling. These could lead to an explosion in the number of proof branches once a prover starts to consider various combinations of control flow chains encoded in such a pre-condition. Thus we include $P_j(c,v)$ as another obstacle to a fully automated proof. Our solution here is to use heuristics to suppress certain "unwanted" (i.e., irrelevant to proof outcome of the current conjecture) hypotheses at various proof stages.

The post-condition $Q_j(c,v,v')$ is very similar to described pre-conditions, with a conjunct in it representing a number of possible state updates. However, every member of this conjunct has, in the context of SSI, the following form of an imperative (i.e., deterministic) state update on individual variables:

$$Q_j \ = \ \bigwedge_j Q_{j,l} \ = \ \bigwedge_l \left(v'_{j,l} = E_{j,l}(c,v)\right)$$

With this in mind, we can further break down definition (2) into

$$\begin{aligned} & M(c) \wedge A(c,v) \wedge I(c,'v,v) \wedge P_j(c,v) \wedge \\ & v'_{j,l} = E_{j,l}(c,v) \\ \Rightarrow & \qquad (3) \\ & I_k(c,v,v'), \end{aligned}$$

generated for every part $Q_{j,l}$ of Q_j. This increases the overall number of conjectures approximately five-fold but is a fair price to pay for proof decomposition (and thus significant simplification) outside of an automatic prover.

Finally, the right hand side of implication $I_k(c,v,v')$ can be a complex predicate but there is really no opportunity to simplify anything more here.

Conjectures of the form (3) are processed by our custom-designed theorem prover. The prover not only attempts to prove or disprove a given conjecture but also, when noticing that a successful proof is unlikely, it would backtrack in the proof, if necessary, for the purpose of a better explanatory failed goal. Backtracking is needed for the steps that create auxiliary variables arising from mapping between different representations thus a making a failed goal unintelligible.

For every failed conjecture, the integrated witness discovery tool harvests "stuck" goals and attempts to interpret them using the notions from the problem domain. For instance, a goal of a form $TAA \notin track\_o$ is interpreted as *section* `TAA` *must be free*. With some further processing, such harvested witnesses are compiled into a verification report presented to a railway engineer. The report details the nature of a violation, the affected SSI and scheme parts, and the likely reason of a found violation. An example of such a report could be found in [3].

V. Practical experience

SSI GDL is the predominant language used for computer-based interlockings on UK mainline railways. It also has applications overseas, including in India, Australia, New Zealand, France and Belgium. Production of SSI GDL data is a labour

intensive process, involving multiple stages of design, independent checking and testing by specialist staff. The process is becoming increasingly challenging as data gets progressively more complex due to the successive addition of functions to address new safety and performance needs: axle counter based train detection controls, emergency releases, European Train Control System (ETCS), etc.

SafeCap has automatically verified compliance of real-world SSI GDL data with safety properties for 19 different UK interlockings. Six of these interlockings were analysed as trial applications of SafeCap. The remaining 13 were commercial applications of SafeCap in live signalling projects. These interlockings varied significantly:

- some were original SSIs, others were more modern technologies that use SSI GDL;
- complexity ranged from 10s of signalling routes to over 200;
- data originated from different signalling design offices and dates (1990 to present);
- some data was entirely new, other was updates of pre-existing older data.

Safety properties were derived from signalling principles specified in industry standards, notably [6]. The argument for the correctness and completeness of these principles stems from the extensive expert review and operational experience over more than two centuries of railway history that has led to them in their current form. Nonetheless, expressing them in the rigorous formal notation needed for SafeCap presented some challenges.

Signalling principles are generally expressed as plain text with a certain level of assumed knowledge about how railways operate. They are also written in a manner that is agnostic to the technology and data constructs through which they are implemented. Translation of signalling principles into useful safety properties required translation of plain text into mathematical expressions in terms of the variables used by the interlocking code (in this case SSI GDL) being analysed. Furthermore, to avoid false positives, the properties had to be written in a manner that mirrored the state transitions instigated by the code.

For example, section C6.3 of [6] contains the requirement that "Points shall only be permitted to move if they are free of....track locking (including dead...locking)." Domain knowledge is required to understand that 'dead locking' means locking by an occupied train detection section. To translate this into a safety property that is meaningful for SSI GDL, some features of the language also needed to be taken into account:

- physical points are represented, and controlled by, software points objects with a commanded lie property that can be either 'reverse', 'normal' or neither;
- train detection sections are represented by track objects that can be in either the 'clear' or 'occupied' state.

The original signalling principle thus translates to "whenever points are commanded into a new lie all train detection sections over those points are proven clear", from which the formal safety property presented earlier can be derived:

$$\begin{array}{l} \forall\, p \in \mathsf{Point} \cdot \\ \quad point\_c'(p) \neq point\_c(p) \\ \quad \Rightarrow \\ \quad point.sections[\{p\}] \cap track\_o = \varnothing \end{array}$$

SafeCap currently verifies 49 distinct safety properties, which represents of the order of 20% of all properties that may need to be verified for UK mainline interlockings. Whilst some of these properties have a direct role in ensuring the two main safety principles of "free from collisions" and "free from derailments", others do so indirectly by mitigating hazards that can occur in real-world applications, but not in the idealised case. For example: in an idealised world, a train would never exceed the limit of its authority to move, the end of which is typically marked by a red signal. However, driver error and/or poor adhesion conditions mean that such scenarios do occasionally occur in the real-world. It is therefore necessary to ensure that, so far as reasonably practicable, railways remain free from collisions even in such scenarios. To this end signalling principles, and hence safety properties, require a section of track known as an 'overlap' to be locked beyond the end of a train's authorised movement just in case it exceeds that authorisation.

Safety property violations, reported by SafeCap in the practical applications, fell into four categories:

- errors – straightforward errors that needed to be corrected;
- vulnerabilities – where the combination of states under which a reported violation occurs can, through other properties, be shown impossible in practice though could unintentionally be made possible through modifications to seemingly unrelated data;
- intentional violations – violations of specific properties in specific locations for operational reasons, for example to allow a train to enter a route occupied by another train, where the "free from collisions" principle is achieved through driving trains on-sight at low speed rather than through interlocking logic;
- false positives – reported violations where it can be shown that none exists.

Practical experience of SafeCap with real-world data has led to refinements of safety properties to reduce false positives by reflecting the manner in which SSI GDL data is constructed. To make efficient use of once scarce processing power, signalling interlockings do not explicitly test safety principles in every instance they apply. Instead, they test specific instances and infer compliance in other instances. For example, where multiple set of points need to be locked in a signalling route, the interlocking may only test that the last section of route (known as a "sub route") locking any of the points is free. The fact that other sub routes locking the points are also free is inferred, because they are locked at the same time as, and freed sequentially before, the sub route being tested.

The report produced by the tool describes in a stylised English all violations of the safety properties and the the nature of the failed conditions, points to the problem location in the SSI source code, and shows a part of the schema diagram with the key elements related to the failed proof. Our earlier paper

provides a snippet of such a report presented as part of a case study that demonstrates how the tool verifies signalling of a real medium-size station [3].

Formal verification is very robust in terms of finding errors for all possible train run scenarios. However, a potential vulnerability was identified due to the phenomenon of hidden errors. Where two or more violations of the same safety property occur in the same section of SSI GDL data, there is a risk of them appearing and being reported as a single violation: one violation becomes hidden by the other(s). If the reported error is an intentional violation or false positive, there is risk that it is hiding a genuine error or vulnerability. To address this risk, having found a violation, SafeCap repeats its analysis with the reported violation states excluded until no more violations are found.

The automated SafeCap verification is accompanied by some manual efforts from the formal method expert and the experienced signalling engineer. The former often needs to adjust the proof tactics for a specific project and the properties, the latter is involved in adjusting the properties and in the interpretation of the automated verification results. The railway expert leads the production of the final report including the categorisation of the violations found and the selection of the counter examples. Some manual efforts are often required to deal with differences in the way the SSI programs and the schemas are produced by different engineering teams and with the mismatches between the SSI program and the schema (typically, naming or naming convention mismatches) in a given project.

Each project constitutes a substantial geographical area controlled by one SSI program located on one interlocking controller unit. The development of the railway network, including signalling is naturally structured into such projects. For example, (re-)development of a large railway station could consist of 2-3 such projects. The full automated verification of one project in SafeCap, after the manual adjustments and corrections, typically takes 2-8 minutes on a standard desktop PC.

Table I shows the statistics about the verification effort conducted in the last three recently completed projects run on a professional PC with 16 cores. The Z project was one of the projects we completed with the longest verification run. To the best of our knowledge, the station Z is one of the largest interlocking in the UK with the SSI code of high complexity. We note here that the scheme elements are not a good predictor of verification complexity. The complexity, especially its upper boundary, is better explained by the maximum cyclomatic code complexity of the verified SSI program, as reflected in the number of computed distinct state transitions (column 6). For project Z, the complexity is due to the presence cascading swinging overlaps that require highly branching logic and deeply nested subroutine calls. The theoretical state space of these SSI projects is extremely large as they have in average 1.5-2.5K Boolean variables and 100-200 Integer variables, but this is irrelevant to the type of verification we conduct.

Initial commercial applications of SafeCap have been in response to a new UK industry requirement for automated verification, which provides additional mitigation of the risk of error in safety-critical SSI GDL data. However, SafeCap has the potential to offer much greater benefit if used earlier in the design process. Doing so would not only meet the industry requirement for automated verification, but could also enable earlier identification of errors in the data production process thereby avoiding expensive and time-consuming rework cycles. The authors are currently working with industry partners on determining the optimal phases for SafeCap analysis within the data production process.

The role of SafeCap in improving the efficiency of interlocking data production, as well as providing an additional mitigation against error, has also been the subject of several railway industry articles and publications [35], [36], [37], [38]. Case study examples of the application of SafeCap to railway systems, and the analysis of results, are included in two of these articles: [35] provides a worked example of formal verification of the property described earlier '*whenever points are commanded into a new lie all train detection sections over those points are proven clear*' for a simple railway junction; [37] provides worked examples of other properties and analyses the processing time needed to verify multiple properties on layouts of varying complexity.

VI. Conclusions

In this paper we present the SafeCap approach to verifying railway signalling, in particular, the signalling data expressed in a program-like representation called SSI. The SafeCap toolset has been successfully deployed in the UK railway industry. As our substantial experience in using SafeCap in real industrial projects has demonstrated, the approach scales extremely well. Moreover, although only a subset of safety principles has been encoded so far, we are confident that the approach is capable to effectively capture and formalise different formats of signalling data as well as required safety properties.

Even though SSI is a fairly simple notation, it is still liable to state explosion. With all possible modules defining controllers for equipment, such as signals and points, in place, the state space can grow to about 10^{1204} states. Also, it should be noted that in industry safety principles are not designed or discussed in terms of train movements – something we commonly see in the research papers applying simulation or state exploration techniques – but rather as constraints on the signalling rules.

Apart from formalising safety properties, the most time-consuming parts of verification are creating (and repeatedly re-checking) a digital representation of a railway schema from a printout or a PDF diagram, and interpreting the findings, which may require tweaks to the safety properties or proving tactics as well as the layout itself (for example, when a signal is physically on the approach to a set of points, but logically beyond them). The verification itself takes seconds and is completely automatic.

A combination of set theory and first order logic as the underlying mathematical language is the result of experiments over the course of several years. It appears to deliver the optimal combination of a terse, efficient notation for expressing

Name	Routes	Points	Signals	Safety invariants	State Transitions	Time, seconds	RAM, peak, GB
X	140	48	83	55	11078	37	42
Y	64	29	92	55	4462	3	<4
Z	190	64	84	55	112560	265	67

TABLE I
VERIFICATION STATISTICS

conjectures and safety invariants, while, at the same time, also enabling effective symbolic automated proofs. Two other alternatives we have also explored are pure predicate logic and first order logic with functions and equality.

A custom made symbolic prover might seem a dangerous direction to take for an industry-oriented tool. Indeed, the prover we have developed is not anywhere as powerful or comprehensive as many state-of-the-art provers we tried in our earlier experiments. However, it has a decisive advantage of being highly customisable via per-invariant tactic scripts. At such a level of fine-tuning it has shown to be able to outrun any competition. The prover is also carefully designed to backtrack and terminate in a state facilitating helpful end user feedback.

To evaluate SafeCap, we initially applied it together with our industrial partners for verifying several existing signalling datasets. This allowed us to not only improve the tool, its scalability and the quality of reporting, but also to identify several previously-unknown areas of risk in the data.

In the last year and a half SafeCap has been successfully used to automatically verify the real-world SSI GDL data for 19 different UK interlockings. This positive experience has demonstrated both its efficiency and efficacy in improving the signalling production processes.

The presented approach offers immediate industry benefits as it can be used within the existing SSI GDL production processes. The rapid, automated verification that it offers enables errors to be identified earlier in these processes, thereby reducing time consuming and expensive re-work. Furthermore, the SafeCap formal approach to verification provides additional assurance over the scenario based testing that is traditionally used in railway signalling. As the safety case underpinning SafeCap develops, and the range of safety properties that it verifies expands, further industry benefits become possible as the manual testing and checking activities are replaced by automated verification by SafeCap.

ACKNOWLEDGMENTS

This work was partially supported by EPSRC/UK STRATA and TRAMS-2 platform grants (EP/N023641/1 and EP/J008133/1). We are grateful to the anonymous reviewers for their suggestions on improving the earlier version of the paper.

REFERENCES

[1] P. Behm, P. Benoit, A. Faivre, J.-M. Meynadier, Météor: A successful application of B in a large project, in: Proceedings of FM'99 – World Congress on Formal Methods, Vol. 1708 of LNCS, Springer, 1999, pp. 369–387.

[2] F. Badeau, A. Amelot, Using B as a High Level Programming Language in an Industrial Project: Roissy VAL, in: Proc. of ZB 2005: Formal Specification and Development in Z and B, Vol. 3455 of LNCS, Springer, 2005, pp. 334–354.

[3] A. Iliasov, D. Taylor, L. Laibinis, A. Romanovsky, Formal Verification of Signalling Programs with SafeCap, in: Proceedings of 37th International Conference, SAFECOMP 2018, Västerös, Sweden, September 19-21, 2018, pp. 91–106. doi:10.1007/978-3-319-99130-6_7.

[4] Commission Implementing Regulation (EU) No 402/2013 of 30 April 2013 on the common safety method for risk evaluation and assessment and repealing, Regulation (EC) No 352/2009, Official Journal of the European Union.
URL https://eur-lex.europa.eu/LexUriServ/\LexUriServ.do?uri=OJ:L:2013:121:0008:0025:EN:PDF

[5] Office of Rail and Road, Common Safety Method for Risk Evaluation and Assessment, Guidance on the application of Commission Regulation (EU) 402/2013, September 2018.
URL https://www.orr.gov.uk/sites/default/files/\om/common-safety-method-guidance.pdf

[6] Interlocking Principles (Former Railway Group Standard GK/RT0060), Network Rail Company Standard NR/L2/SIG/30009/GKRT0060, Issue 2, 07/03/2015.

[7] D H Stratton, Solid State Interlocking. First edition, IRSE Booklet, 28. Institution of Railway Signal Engineers (IRSE). 20 pages. 1988.

[8] J. Rushby, Kernels for Safety?, in: T. Anderson (Ed.), Safe and Secure Computing Systems, Blackwell Scientific Publications, 1989, Ch. 13, pp. 210–220.

[9] A. Ferrari, M. H. ter Beek, F. Mazzanti, D. Basile, A. Fantechi, S. Gnesi, A. Piattino, D. Trentini, Survey on Formal Methods and Tools in Railways: The ASTRail Approach, in: Proceedings of Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - Third International Conference, RSSRail 2019, Vol. 11495 of Lecture Notes in Computer Science, Springer, 2019, pp. 226–241. doi:10.1007/978-3-030-18744-6_15.

[10] M. J. Butler, P. Körner, S. Krings, T. Lecomte, M. Leuschel, L. Mejia, L. Voisin, The First Twenty-Five Years of Industrial Use of the B-Method, in: Proceedings of Formal Methods for Industrial Critical Systems - 25th International Conference, FMICS 2020, Vienna, Austria, September 2-3, 2020, Vol. 12327 of Lecture Notes in Computer Science, Springer, 2020, pp. 189–209. doi:10.1007/978-3-030-58298-2_8.

[11] A. H. Cribbing, I. H. Mitchell, The application of advanced computing techniques to the generation and checking of SSI data, IRSE Proceedings 1991/92 (1992) 54–64.

[12] M. J. Morley, Safety Assurance in Interlocking Design. PhD thesis, University of Edinburgh (1996).

[13] P. James, A. Lawrence, F. Moller, M. Roggenbach, M. Seisenberger, A. Setzer, K. Kanso, S. Chadwick, Verification of Solid State Interlocking Programs, in: SEFM 2013 Workshops, Vol. 8368 of LNCS, Springer, 2014, pp. 253–268.

[14] M. Huber, S. King, Towards an Integrated Model Checker for Railway Signalling Data, in: Proceedings of FME 2002: Formal Methods Europe, Vol. 2391 of LNCS, Springer, 2002, pp. 204–223.

[15] Q. Cappart, C. Limbrée, P. Schaus, J. Quilbeuf, L.-M. Traonouez, A. Legay, Verification of Interlocking Systems Using Statistical Model Checking, in: Proceedings of HASE – High Assurance Systems Engineering, 2017, pp. 61–68.

[16] S. Busard, Q. Cappart, C. Limbrée, C. Pecheur, P. Schaus, Verification of railway interlocking systems, in: Proceedings of ESSS 2015, 2015, pp. 19–31.

[17] Q. Cappart, P. Schaus, A Dedicated Algorithm for Verification of Interlocking Systems, in: Proceedings of Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP 2016, Vol. 9922 of Lecture Notes in Computer Science, Springer, 2016, pp. 76–87. doi:10.1007/978-3-319-45477-1_7.

[18] Department for Transport, RAIB review of the railway industry's investigation of an irregular signal sequence at Milton Keynes, available at https://www.gov.uk/raib-reports/review-of-the-railway-industry-s-formal-investigation-of-an-irregular-signal-sequence-at-milton-keynes (2008).

[19] C. Limbrée, Q. Cappart, C. Pecheur, S. Tonetta, Verification of Railway Interlocking - Compositional Approach with OCRA, in: Proc. of RSS-

Rail – Reliability, Safety, and Security of Railway Systems, Springer, 2016, pp. 134–149.
[20] H. D. Macedo, A. Fantechi, A. E. Haxthausen, Compositional Verification of Multi-station Interlocking Systems, in: Leveraging Applications of Formal Methods, Verification and Validation, Springer, 2016, pp. 279–293.
[21] A. Cimatti, R. Corvino, A. Lazzaro, I. Narasamdya, T. Rizzo, M. Roveri, A. Sanseviero, A. Tchaltsev, Formal Verification and Validation of ERTMS Industrial Railway Train Spacing System, in: CAV, Springer, 2012, pp. 378–393.
[22] T. Gonschorek, L. Bedau, F. Ortmeier, Automatic Model-based Verification of Railway Interlocking Systems using Model Checking, in: Proceedings of ESREL, 2018, pp. 741–748.
[23] J. Falampin, H. Le-Dang, M. Leuschel, M. Mokrani, D. Plagge, Improving Railway Data Validation with ProB, in: Industrial Deployment of System Engineering Methods, Springer, 2013, pp. 27–43. doi:10.1007/978-3-642-33170-1_4.
[24] A. Iliasov, I. Lopatkin, A. Romanovsky, The SafeCap Platform for Modelling Railway Safety and Capacity, in: Proceedings of SAFECOMP - Computer Safety, Reliability and Security. LNCS 8135, Springer, 2013, pp. 130–137.
[25] A. Iliasov, I. Lopatkin, A. Romanovsky, Practical Formal Methods in Railways – The SafeCap Approach, in: Proceedings of Reliable Software Technologies (Ada-Europe), LNCS 8454, Springer, 2014, pp. 177–192.
[26] A. Iliasov, A. B. Romanovsky, Formal Analysis of Railway Signalling Data, in: Proceedings of HASE – High Assurance Systems Engineering, 2016, pp. 70–77.
[27] A. Iliasov, P. Stankaitis, D. Adjepon-Yamoah, Static Verification of Railway Schema and Interlocking Design Data, in: Proceedings of RSSRail: Reliability, Safety, and Security of Railway Systems, 2016, pp. 123–133.
[28] F. Bobot, J.-C. Filliâtre, C. Marché, A. Paskevich, Why3: Shepherd your herd of provers, in: Proccedings of Boogie 2011, 2011, pp. 53–64.
[29] M. Leuschel, M. Butler, ProB: A Model Checker for B, in: A. Keijiro, S. Gnesi, M. Dino (Eds.), Formal Methods Europe 2003, Vol. 2805, Springer-Verlag, LNCS, 2003, pp. 855–874.
[30] A. Iliasov, A. Romanovsky, SafeCap Domain Language for Reasoning about Safety and Capacity, in: Proceedings of PRDC - Pacific-Rim Dependable Computing, IEEE, 2012, pp. 1–10.
[31] M. Ingleby, I. Mitchell, Proving Safety of a Railway Signalling System Incorporating Geographic Data, in: H. H. Frey (Ed.), Safety of Computer Control Systems (Safecomp'92), IFAC Symposia Series, Pergamon, 1992, pp. 129–134. doi:https://doi.org/10.1016/B978-0-08-041893-3.50026-4.
[32] Signalling Design Control Tables. GKRT0202 Issue 1. Railway Group Standard. Issued Oct 1, 1996., UK Railway Safety and Standards Board.
[33] E. W. Dijkstra, Guarded Commands, Nondeterminacy and Formal Derivation of Programs, Communications of ACM 18 (8) (1975) 453–457. doi:10.1145/360933.360975.
[34] R.-J. Back, J. von Wright, Refinement Calculus: A Systematic Introduction, Springer, 1998. doi:10.1007/978-1-4612-1674-2.
[35] A. Iliasov, D. Taylor, A. Romanovsky, Automated testing of SSI data. IRSE (Institution of Railway Signal Engineers) News 241, February 2018 (2018).
URL https://www.irse.org/Publications-Resources/IRSE-News/Archived-Issues
[36] SafeCap: Automated Verification of Railway Signalling. Rail Engineer 168. October 2018 (2018).
[37] D. Taylor, A. Iliasov, A. Romanovsky, K. King, Driving Efficiency & Resilience to Human Error: SafeCap Automated Verification of Signalling Data, in: In Proceedings of ASPECT 2019, Delft, Netherlands, October 21-24, IRSE, 2019.
URL https://webinfo.uk/webdocssl/irse-kbase/ref-viewer.aspx?refno=740881177
[38] D. Taylor, A. Iliasov, K. King, O. Jarratt, S. Benson, W. Dearman, Command, control and signalling design in the digital age. IRSE News 271. November 2020 (2020).

Alexei Iliasov received his PhD degree in computer science from Newcastle University, Newcastle, UK, in 2008 in the area of modelling artefacts reuse in formal developments. Until 2019 he was a Senior Research Associate with the School of Computing, Newcastle University. Since 2012 he is the Technical Director of The Formal Route Ltd., a SME working in the area of signalling safety. His research interests include formal methods and tools for software engineering and verification.

Dominic Taylor is the Technical Head of Systems and Signalling at SYSTRA Scott Lister. He has fifteen years' railway signalling experience with a focus on the introduction of new technologies into the sector including formal verification, ETCS, traffic management, video surveying and novel data management techniques.

Linas Laibinis received his PhD degree in 2000 from Åbo Akademi University (Finland) on the topic of Mechanised Formal Reasoning About Modular Programs. His research interests include formal development of software systems, using automated formal methods and tools for verification and validation of software models, program correctness, automated theorem proving, as well as formal techniques for probabilistic or numerical assessment of quantitative system characteristics. He is the author of more than 70 scientific articles. From 2017, Laibinis is a Professor in Institute of Computer Science, Vilnius University, Lithuania.

Alexander Romanovsky received the MSc degree from Moscow State University and the PhD degree from Saint Petersburg State Technical University (Russia). He is a Professor with the School of Computing, Newcastle University, UK. His main research interests include system dependability, fault tolerance, software architectures, system verification for safety and system structuring. In 2004-2012 he coordinated two major European Projects, RODIN and DEPLOY, that developed and deployed in industry several advanced formal modelling techniques. He is a Director of The Formal Route Ltd., an SME working in the area of signalling safety.