Software Model Checking of Interlocking Programs

Phillip James[0000−0002−4307−649X], Faron Moller[0000−0001−9535−8053], and Markus Roggenbach[0000−0002−3819−2787]

Swansea University, UK
{p.d.james, f.g.moller, m.roggenbach}@swansea.ac.uk

Abstract. In this paper, we report and reflect on successful technology transfer from Swansea University to Siemens Mobility over the years 2007–2022. This transfer concerns formal software verification technology for interlocking computers. It spans over Technology Readiness Levels TRL 1-7 and was reported on in two REF Impact Case Studies, in 2014 and 2021 [17,18].

To Jan

Who shows us that
excellence in research, both foundational and applied, and
technology transfer go hand in hand.

1 A signalling problem and our approach to solving it

Interlockings are safety-critical systems which form an essential part of rail control systems. They are often realised as programmable logic controllers programmed in the language Ladder Logic, cf. IEC standard 61131 [8]. In the context of rail signalling systems, they provide a safety layer between a (human or automatic) controller and the physical track which guarantees safety principles such as: before a signal can show proceed, all train detection devices in the route indicate the line is clear. Rail authorities such as the UK Rail Safety and Standards Board as well as rail companies such as Siemens Mobility have defined such safety principles (currently, we work with about 350 principles) that shall guarantee safe rail operation. This poses the research question of how one can verify that a given program written in Ladder Logic fulfils a safety property, i.e., a logical representation of a safety principle.

Our journey to answer this question went, up to now, through three phases, cf. Fig. 1. A number of themes stayed invariant in all of them, though each phase shed its specific light on them: how to ensure faithful models of the software and its desired properties? what program size can be treated? who can use the produced artefacts? how can different components interact with each other?

The first phase concerns *Theoretical Foundations*, cf. Section 2. In terms of artefacts involved, all are paper-based documents, speaking about faithfully

Artefacts	Documents	1^{st} Prototype	2^{nd} Prototype
Theme	Theoretical Foundations	Academic Experiments	Technology Transfer
Faithful modelling	LL-programs as transition systems	Capturing rail-specific safety properties	'All' safety properties from standards
Scalability	'Small' theory only	Slicing	Optimised encoding
Usability	Mathematical reader	Tool programmer	Industry standard
Interoperability	Semantic preservation	'Whatever goes'	Bespoke Siemens formats

Fig. 1. The three phases of our journey.

representing Ladder Logic programs and their properties in propositional logic and first order logic. Only programs of a few lines length can be treated manually. Academics can work out examples. It is key to preserve semantics between the different artefacts (logics and automata), cf. *Example* 1 below.

The next phase was on building a 1^{st} Prototype for carrying out *Academic Experiments*. As a proof of concept, selected rail-specific safety principles were modelled in first order logic and transformed into propositional formulae. Abstraction through program slicing turned out to be a necessity in order to verify interlocking programs of small railway stations with the SAT solvers and computing power available in 2010. The 1^{st} Prototype could be operated mainly by the original tool developers. Its components were written in different languages with data exchange through text files.

The performance of the 1^{st} Prototype was promising enough to support development of a 2^{nd} Prototype, starting a process of *Technology Transfer*, i.e., for performing verification in the industrial setting of Siemens Mobility. Safety properties were systematically encoded in first order logic, their transformation to propositional logic was proven to be semantics-preserving using a temporal first order logic. In terms of scalability, optimised encodings of the safety properties were developed. The 2^{nd} Prototype was developed with Rail Engineers as users in mind. All software was written in C♯ and bespoke Siemens Mobility interfaces were used to guarantee interoperability with existing Siemens Mobility tools.

Utilizing the notion of Technology Readiness Levels (TRLs)[1], the rest of the paper describes our journey through these three phases. The paper concludes with a perspective on the final phase: establishing an improved 2^{nd} Prototype as a standard tool in interlocking design.

This paper can be read in different ways. The domain expert on Ladder Logic will find out about concrete, state-of-the-art steps of how to verify Ladder Logic programs of industrial size. The formal methods expert or industrial research manager will find a report on a successful technology transfer together with reflections on lessons learnt.

[1] Technology Readiness Levels, HORIZON 2020, Annex G

An early version of this paper appeared as an extended abstract in the proceeding of Isola'21 [3].

2 Theoretical foundations

We discuss the theoretical foundations of our first prototype. This includes basic principles (TRL1) and the formulation of a technology concept (TRL2).

2.1 Textbook knowledge on verifying finite transition systems

The following definition is standard:

Definition 1. *Let $\bar{x} = (x_1, \ldots, x_n)$, $\bar{x}' = (x'_1, \ldots, x'_n)$, and $\bar{i} = (i_1, \ldots, i_m)$ be vectors of Boolean variables, for some $m, n \geq 0$. Given propositional formulae $I(\bar{x})$ (the initialisation condition) and $T(\bar{x}, \bar{i}, \bar{x}')$ (the transition condition), we define a* **labelled transition system** *$\mathcal{S} = (S, \longrightarrow, Init)$ as follows:*

- *The set of all Boolean vectors $S = \{0,1\}^n$ is the set of states;*
- *$\longrightarrow \subseteq S \times \{0,1\}^m \times S$ is the transition relation given by*

$$s \xrightarrow{i} s' :\Longleftrightarrow T(s, i, s') \text{ evaluates to } 1;$$

- *$Init = \{s \in S \mid I(s) \text{ evaluates to } 1\}$.*

We say that a state s is **reachable** *in $\mathcal{S}$, if there exists a (possibly empty) sequence of transitions from a state $init \in Init$ to s. We write $Reachable(\mathcal{S})$ for the set of reachable states of $\mathcal{S}$.*

Given a propositional formula $P(\bar{x})$, we say the transition system $\mathcal{S}$ **has safety property** *P if $P(s)$ evaluates to 1 for all $s \in Reachable(\mathcal{S})$.*

Example 1. Fig. 2 shows the transition system defined by the initialisation con-

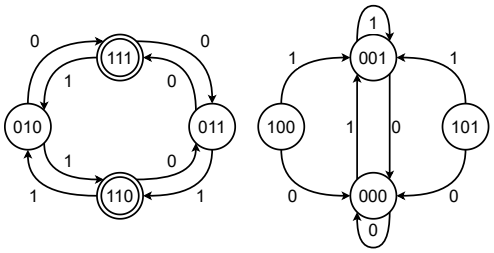

Fig. 2. A finite transition system

dition $I = x \wedge y$ and the transition condition

$$T \;=\; \big(x' \longleftrightarrow (\neg x \wedge y)\big) \;\wedge\; \big(y' \longleftrightarrow y\big) \;\wedge\; \big(z' \longleftrightarrow a \oplus y'\big)$$

over the vectors (x, y, z), (x', y', z') and (a) of Boolean variables. The initial states are marked by double circles. Considering the property $P = x \vee y$, we can see that it holds for all reachable states; however, the property $Q = y \wedge z$ does not hold, as, e.g., the state 010 is reachable.

The following Theorem attests to the fact that various verification methods can be applied to such transition systems.

Theorem 1. *Let $\mathcal{S}$ be a transition system as in Definition 1, and let $P(\bar{x})$ be a propositional formula representing a safety property.*

Inductive Verification: *Provided*

- $I(\bar{x}) \longrightarrow P(\bar{x})$ *and*
- $P(\bar{x}) \;\wedge\; T(\bar{x}, \bar{i}, \bar{x}') \longrightarrow P(\bar{x}')$

hold, then $\mathcal{S}$ has safety property P.

Inductive Strengthening: *Let $Inv(\bar{x})$ be a propositional formula such that $Inv(s)$ evaluates to 1 for all $s \in Reachable(\mathcal{S})$. Provided*

- $I(\bar{x}) \longrightarrow P(\bar{x})$ *and*
- $P(\bar{x}) \;\wedge\; Inv(\bar{x}) \;\wedge\; T(\bar{x}, \bar{i}, \bar{x}') \longrightarrow P(\bar{x}')$

hold, then $\mathcal{S}$ has safety property P.

Bounded Model Checking: *If $\mathcal{S}$ has safety property P, then for all $k \geq 0$:*

$$I(\bar{x}) \;\wedge\; T(\bar{x}, \bar{i}, \bar{x}') \;\wedge\; T(\bar{x}', \bar{i}', \bar{x}'') \;\wedge \cdots \wedge\; T(\bar{x}^{(k-1)}, \bar{i}^{(k-1)}, \bar{x}^{(k)}) \longrightarrow P(\bar{x}^{(k)}).$$

(In the above, given a vector of boolean variables z, we denote by $\bar{z}^{(m)}$ a vector in which each variable has m prime symbols.)

Applying Theorem 1 to Fig. 2, we observe the following.

- *Inductive Verification* cannot be used to show that $\mathcal{S}$ has property P, as it considers all states rather than only reachable states; this over-approximation provides a false positive resulting from the unsafe state 001 being reachable from the safe but unreachable state 101.
- *Inductive Strengthening* can be used to show that $\mathcal{S}$ has property P, using the invariant $Inv = \neg((x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge \neg z))$.
- *Bounded Model Checking* can be used to show that $\mathcal{S}$ does not have the property Q: $I(\bar{x}) \wedge T(\bar{x}, \bar{i}, \bar{x}') \rightarrow Q(\bar{x}')$ does not hold as $I(110)$ evaluates to 1, $T(110, 0, 010)$ evaluates to 1, but $Q(010)$ evaluates to 0.

The three verification methods listed in Theorem 1 can be decided utilising SAT solving. All conditions listed are of the form $\models \varphi$ for some propositional formula φ, i.e., we need to determine if a formula φ is valid. This is equivalent to determining if $\neg\varphi$ is satisfiable.

Definition 1 and Theorem 1 can be extended to cater for safety properties P that speak about several consecutive states and also take inputs into account.

2.2 Verifying propositional safety properties of Ladder Logic programs

The operation of an interlocking (IXL) can be described in terms of the following imperative program:

Algorithm 1: PLC Operation

input : Sequence of values
output: Sequence of values

```
initialisation
while (true) do
    read (Input)                                        %% read
    State′ ← LadderLogicProgram(Input, State)           %% process
    write (Output′) & State ← State′                    %% update
```

After initialisation of the system's state, the IXL runs in a non terminating loop. This loop consists of three steps: First, the IXL reads Input, a set of values; based on this Input and the IXL's current State, utilizing a Ladder Logic program, the IXL computes its next state State′ which also includes some Output′ values; finally, the PLC writes Output′ and updates its state.

Ladder Logic, defined in the IEC standard 61131 [8], is a graphical programming language for PLCs. It gets its name from its ladder-like appearance for programs. We consider a sublanguage of Ladder Logic, which from a mathematical point of view is a subset of propositional logic.

Definition 2. *Let I and C be finite, disjoint sets of Boolean variables, where I represents input variables and $C = \{c_1, \ldots, c_n\}$ represents n distinct state/output variables, from which we define a set of* update *state/output variables $C' = \{c'_1, \ldots, c'_n\}$. A* **Ladder Logic formula** *ψ is a propositional formula of the form*

$$\psi \;\equiv\; (c'_1 \leftrightarrow \psi_1) \;\wedge\; (c'_2 \leftrightarrow \psi_2) \;\wedge \cdots \wedge\; (c'_n \leftrightarrow \psi_n)$$

in which, for each $i \in \{1, \ldots, n\}$, $vars(\psi_i) \subseteq I \cup \{c'_1, \ldots, c'_{i-1}\} \cup \{c_i, \ldots, c_n\}$. The conjuncts of ψ are referred to as **rungs***; and the restriction on variables ensures that the update value c'_i of each rung depends only on input variables along with update values (c'_j with $j < i$) from earlier rungs and non-update values (c_k with $k \geq i$) from the previous cycle.*

Formula T from *Example* 1 is a Ladder Logic formula. Processing program T with Algorithm 1 runs it through the states as shown in Fig. 2, provided the initialisation step sets the initial states according to formula I.

This connection allows to apply the verification methods listed in Theorem 1 to prove that a propositional safety property holds for a Ladder Logic program.

2.3 Translating generic safety principles to track plan specific ones

Safety principles, as stated by the UK Rail Safety and Standards Board or rail companies such as Siemens Mobility, are often formulated in tables, with one

column providing preconditions for an effect such as "movement authority can be given", and further columns indicating the kind of route (main or shunting) to which a rule applies. The table entries are written in natural language. As such, the first step towards verification is to formalise such safety principles in, e.g., many-sorted first order logic (FOL) with variables ranging over entities such as points, signals, routes and track segments, resulting in predicates describing track layout and system state.

Example 2. The safety principle

> *For all pairs of distinct routes that share a track segment, at most one route is set to 'proceed'.*

can be formalised as

$$\forall rt, rt' \in \text{Route} \,.\, \forall ts \in \text{Segment} \,.\, rt \neq rt' \;\longrightarrow$$
$$\big((\text{part\_of}(ts, rt) \wedge \text{part\_of}(ts, rt')) \;\longrightarrow\; \neg\big(\text{route\_set}(rt) \wedge \text{route\_set}(rt')\big)$$

The following Theorem attests to the fact that, given a concrete track plan, formulae expressing safety principles in FOL can be translated to a logically equivalent formula in propositional logic (PL).

Theorem 2.

1. *Every formula in FOL is logically equivalent to a formula in prenex normal form (i.e., a formula is written as a string of quantifiers and bound variables, followed by a quantifier-free part).*
2. *Assuming that the carrier of a sort symbol s is freely generated by finitely many constant symbols $c_1, \ldots, c_k : s$,*
   - $\forall x \in s \,.\, \varphi(x) \;\leftrightarrow\; \varphi(c_1) \;\wedge \cdots \wedge\; \varphi(c_k)$ *and*
   - $\exists x \in s \,.\, \varphi(x) \;\leftrightarrow\; \varphi(c_1) \;\vee \cdots \vee\; \varphi(c_k)$.

The formula in *Example* 2 is in prenex normal form. Using quantifier replacement as formulated in Theorem 2, a typical resulting subformula looks like:

$$\big(\text{part\_of}(\mathit{ts54}, \mathit{rt26}) \wedge \text{part\_of}(\mathit{ts54}, \mathit{rt27})\big)$$
$$\longrightarrow\; \neg\big(\text{route\_set}(\mathit{rt26}) \wedge \text{route\_set}(\mathit{rt27})\big)$$

This subformula contains two different kinds of predicates. The first kind concerns the topology of a track plan: $\text{part\_of}(\mathit{ts54}, \mathit{rt26})$. The property, if track segment 54 is part of route 26, can automatically be evaluated by analysing the track plan under discussion. The second kind concerns the state of the interlocking: $\text{route\_set}(\mathit{rt26})$. This predicate, including its application to a constant, corresponds to a variable in the Ladder Logic program under discussion.

Outcomes:

General safety principles can be formalised within in FOL.
Then they are translated into track plan specific safety properties in PL.
Such properties in PL can be verified for Ladder Logic programs.

3 Technology prototype

Having developed a sound formal basis, our work moved towards developing a verification process for Siemens Mobility. The main goal here was to provide an academically built, prototypical tool chain that allowed for experimental proof of concept (TRL3) and a technology that was validated in the controlled setting of an academic environment (TRL4). Many groups have worked on similar projects. As an example, we mention here Groote et al. who, as early as 1995, applied software verification to a real world interlocking [9].

3.1 Automatising translations

The first aspect of our tooling concerns transformation between data formats. For our setting, the following data translations were required:

- Ladder Logic Programs L represented in Siemens Westrace format needed parsing and automatic translations to our defined transition system $\psi(L)$ (in a suitable formal format).
- Our first order logic safety principles needed translating into propositional logic instances specific to the given track plan T.
- A process for translating counterexamples from failed model checking attempts into a insightful format was needed.

We briefly explore initial tools that were developed, to overcome these challenges. For detailed reading we refer to [10,11,12,13].

The verification tool created by Kanso [12] and James [10] consisted of two underlying programs, one concerned with verification and the other concerned with safety properties. The general outline of these tools is shown in Fig. 3 and Fig. 4. The verification component of the tool was predominantly pro-

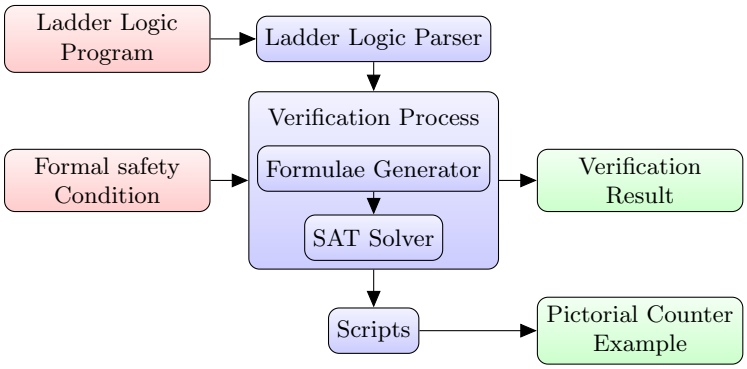

Fig. 3. Architecture of the verification tool.

grammed using Haskell. As input, it takes a Ladder Logic program and a formal safety condition obtained from the safety condition generator. This program

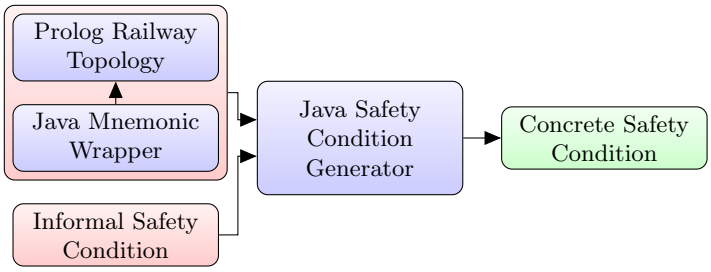

Fig. 4. Architecture of the Safety Condition Generator.

firstly uses a Ladder Logic parser to parse the Ladder Logic program into an internal abstract syntax for propositional formula. Using this propositional formula, and the safety condition (given by the process below), the program constructs either a pair of inductive formulae to be verified, or a bounded model checking problem. These formulae are then passed to a SAT solver to be verified. Depending on the result from the SAT solver, the program would then either report that the system was safe, or provide a counterexample showing the system to be unsafe; the counterexample would be run through a series of scripts to present it in pictorial form for the engineers at Siemens Mobility to study.

From Fig. 4, we can see that there are two inputs to the program for generating safety conditions to be used in the verification phase. One is an informal first order safety condition, given in a language defined by Kanso [12]. The other is a railway plan constructed out of two further parts: a railway topology describing the layout of the railway via an encoding in Prolog, and a Java mnemonic wrapper. The name space of the railway topology used for signals, points and other entities differs from the name space of the concrete Ladder Logic program. For this reason, a specific name space mnemonic wrapper was created in Java. This wrapper is responsible for converting names used in the railway topology into concrete names used within the Ladder Logic program.

Given these inputs, the safety condition generation program transforms the informal safety condition into a series of propositional formulae to be verified. The propositional formulae would contain concrete names instead of the abstract names that were given in the informal safety condition. For example the word "point" could be replaced by the actual point "TP101". These names are gained, as explained above, from the Prolog encoding and Java wrapper.

3.2 First academic experiments

Initial experiments conducted using the produced tool chain were based on two small interlocking programs (each around 300 rungs in size) and around five safety principles. The results [12,10] highlighted the following.

- Inductive verification can be successfully applied to verify properties of 'small' interlockings, although some properties could not be verified due to false-positives (see Theorem 1). Strengthening the approach to include

k-induction [10] (also know as temporal induction [5]), which aims to avoid such false positives, was not feasible due to the size of the problems.

- The bounds that were practically explored using bounded model checking were fairly limited in size. However, applying bounded model checking allowed successful error detection (with run times stretching over hours).
- Presenting counterexamples that only included information on the final violating state were less useful in identifying underlying issues than studying (in a somewhat laborious manner) counterexample traces produced by bounded model checking.

3.3 Improving verification through slicing

During the development of these results, it was clear that any potential abstractions that could be formulated to reduce the size of the state space for verification would greatly improve the applicability of the tool. The proposed approaches for the verification of Ladder Logic programs quickly give rise to large formulae to be verified. As the formula size increases, both the space and time requirements increase. This increase leads to a rather small bound (approximately 2 000) on the number of iterations of a Ladder Logic program that could be verified. Hence, in a somewhat hand-in-hand nature whilst running initial experiments, a program slicing abstraction was developed [10,11] following ideas presented in [9,6].

The intuition behind slicing is that the variables occurring in a safety condition often depend only on some part of the Ladder Logic program, and hence parts that have no effect on the safety condition can be removed. At a high level, the approach takes the following steps.

Step 1: Extract variables from safety conditions. Given a safety condition φ, we extract its variables $U = vars(\varphi)$.

Step 2: Calculate dependant variables. Calculate all the variables of the Ladder Logic formula that affect the variables in U. To do this, we begin at the last rung $R_i \equiv c_i' \leftrightarrow \psi_i$ of the Ladder Logic formula and compare its variable c_i' with the set of variables U. If $c_i' \in U$, then we add all the variables occurring in ψ_i to the set U. This step is repeated for each rung until a fixed point $\overline{U}$ is reached.

Step 3: Extract dependant rungs. Using the variable set $\overline{U}$, we remove all rungs that do not affect the safety condition. To do this, we construct the set

$$index = \{\ i \in \{1, \ldots, n\} \mid c_i \in \overline{U} \ \text{ or } \ c_i' \in \overline{U}\ \}.$$

We then remove from the original program all rungs R_i whose indices do not appear in $index$. The result ψ_φ is the sliced version of program ψ.

Example 3. Considering the finite transition system in *Example* 1 and the safety property $P = x \vee y$, we can construct a sliced version of the propositional formulae. Here we can compute the variables in the formulae that affect the set $U = \{x, y\}$ of variables from our safety property. In particular, the formulae

defining the values of x and y are both required, but the formula defining z is not. We can thus remove z from our transition system as it does not affect the values in U. This results in the transition system given in Fig. 5.

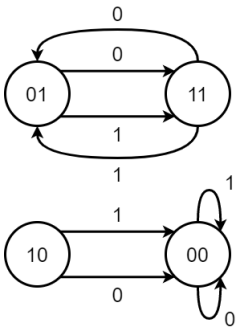

Fig. 5. The transition system from *Example* 1 after slicing

In [10] we proved the following theorem:

Theorem 3. *Let φ be a safety condition over a Ladder Logic formula ψ. The transition system induced by ψ has safety property φ iff the transition system induced by ψ_φ has safety property φ.*

Slicing proved effective in the two interlockings with which we experimented. For the first interlocking, the number of rungs was reduced, on average, from 331 rungs to 60 rungs. For the second interlocking, the number of rungs was reduced, on average, from 238 rungs to 25 rungs [11].

The novelty of our approach was that we gave the first proof that slicing is correct with respect to reachable states [10,11]. We also demonstrated that the above approach gave a significant reduction – a full order of magnitude on average – in the size of the state space required for verification of the five initial properties considered. For the first of the two interlockings with which we experimented. the number of rungs was reduced, on average, from 331 rungs to 60 rungs. For the second interlocking, the number of rungs was reduced, on average, from 238 rungs to 25 rungs. For full details and results we refer to [12,10].

Outcomes:

The 1^{st} prototype fully automatically verifies Ladder Logic programs.
With slicing, it can effectively handle small IXLs ($\sim$ 300 rungs).
General safety properties are translated into location specific ones.

4 Technology transfer

In the latest phase of development, a substantial effort has been put towards re-developing the technology stack. This has seen new formats developed for data along with a complete re-writing of the verification tool. This has allowed the demonstration of the verification process within the operational environment of signalling system design at Siemens Mobility (TRL 7). Here we highlight the main scientific and technological challenges.

4.1 Logic rework

From a methodological point of view, in the logical approach developed in Section 2 one could see a number of weaknesses. The first concerns the nature of the safety properties under discussion. By their very nature, they are temporal properties relating consecutive states. However, we formulated them in FOL without modelling the temporal aspect. The second weakness is that, when translating a generic safety property in FOL to a track plan specific one in PL, for each track plan we are building a specialised logic. For instance, the truth of the formula part_of$(ts54, rt26)$ depends on the track plan under discussion. Diaconescu discusses how to build logics (institutions) with predefined types [4]. The final weakness concerns the application of a hybrid specification. Verification is carried out in PL, whilst the translation of generic safety properties to track plan specific properties is carried out in FOL with predefined types. These two logics are combined only by the sharing of signature elements rather than by a semantic integration.

Addressing the first and the last of these points, we sketch here a temporal logic based on ideas published in Gruner et al. [7]. *Signatures* are many sorted, first-order logic signatures. A *model* at a point of time is a pair (T, I) where T is a track plan and I is a propositional model for all propositional variables associated with T, e.g. $I(S106.G) \in \{true, false\}$. Sorts and functions are given a fixed interpretation according to the track plan, e.g.,

- $Signal_T = \{S100, ...\}$: iff T *has signals* $S100, ...$
- $routesOf_T(s) = \{r_1, ..., r_n\}$: iff *in* T, *signal* s *has routes* $r_1, ..., r_n$

Predicates obtain their interpretations usually from a combination of looking up information from both the track plan T and the propositional model I: p $isInCorrectPositionFor_{T,I}$ r *holds* iff

- **case 1:** in T, p needs to be in reverse for r and $I(p.RL)$ is true
- **case 2:** in T, p needs to be in normal for r and $I(p.NL)$ is true

Here, p is a point, r is a route, and $p.NL$ and $p.RL$ are variables in the Ladder Logic program representing point p being set to normal or reverse position respectively. The models of a signature are sequences of the form $(T, I_0), (T, I_1), ...$, i.e., the track plan in the first component stays constant, only the state of the propositional variables is changing.

Formulae are standard first order logic formulae, where predicate symbols can also appear with up to k primes, $k \geq 0$. The prime indicates that a predicate shall be evaluated in the k-th successor state.

Given $k+1$ models $(T, I_0), \ldots, (T, I_k)$, *satisfaction of a formula* is satisfaction as in first order logic, where l primed predicates are evaluated over (T, I_l). A formula φ holds in a sequence $\langle (T, I_0), (T, I_1), \ldots \rangle$, iff for all $i \geq 0$ the formula φ holds over $(T, I_i), \ldots, (T, I_{i+k})$.

4.2 Data formats, interoperability and efficiency

A large focus of the redevelopment was on both expansion of the number of interlocking safety principles to cover the full standard (approximately 350 properties) and interoperability of the tool with existing Siemens Mobility formats. Here, the following core technology changes were made:

Systematic documentation of Ladder Logic variable naming schemes. Different interlockings use various naming schemes for variables. Typically these are dependant on the type of signalling scheme being deployed and the geographic location of the interlocking. Here, to allow for generalisability of the verification process, an XML schema was defined that allows for parameterisation of the verification process by a naming scheme.

Rewrite of the verification engine with a focus on efficiency. The underlying Haskell verification engine was rewritten in C#. Here, the main focus was on improvements in efficiency and ensuring Siemens Mobility development processes for developing safety critical software we followed.

Representation of Safety Properties. In order to provide a standardised language for Safety Properties, a new XML schema was defined that captured first order logic with predicates describing objects that occur within the railway domain. Rather than having an ad-hoc definition, these predicates were designed to align with an existing Siemens Object Model used internally to capture railway components. However, the language was also influenced by the earlier Prolog formats and included predicates to describe states of variables (such as in the next step). All 350 safety principles were then modelled within this language. To ease readability of the XML, an XSLT transformation was defined to produce HTML-viewable formulae. This also allowed for independent validation of the modelling. Finally, a C# module was developed to translate the XML properties down into the concrete condition format used by the verification engine which we developed. This translation included a mapping for variable names as defined by the given XML mapping presented above.

Professional UI supporting verification work-flow. An extensive user interface was developed in C# allowing signalling engineers to interact with the verification process. Here, not only were obvious features implemented such as file interactions, but also strategies for verification were introduced that involved imposing order on proof attempts (for example inductive, then bounded model checking) along with parallelisation of proof attempts. Another feature that was heavily explored was dealing with particularly large and lengthy counterexamples. This led to a parser for counterexamples being developed along with features

for filtering and the dynamic selection of variables based on the failing property. Here, earlier work on counterexample visualisation was used as a motivation [16].

4.3 Technicalities of real world constraints

During the redevelopment, the verification process was actively tested against a number of more complex interlockings (see Section 4.4). During this, several unexpected phenomena were observed that challenged the underlying theoretical foundations of the tool. There were also improvements made to the efficiency of the verification engine thanks to insights from domain knowledge.

Fleeting outputs. Fleeting outputs are regarded as outputs that "flip" their value for a single cycle of the interlocking. Such outputs can cause counterexamples when model checking; however, in practice, this flip happens at a rate that is quicker than can be observed on the railway, and thus is not too concerning for signalling engineers. Here, a strategy was devised to allow checks that can ignore fleeting outputs if requested by the engineer during verification. Logically, 'Fleeting outputs' require more than two successive states to be encoded into a safety property, i.e., changing a formula from $\bar{x}$ and $\bar{x}'$ to $\bar{x}^1, \bar{x}^2, \ldots, \bar{x}^k$, $k = 3$ appears to be enough. This is, however, a check that should only be enabled after careful consideration of the original counterexample.

Boundary based properties. A number of interlocking safety principles concern the operation of the interlocking with respect to the boundary regions of the railway it controls. Here, verification of these particular principles became a challenge as it became apparent that the principles relied on assumptions made about a bordering interlocking controlling an adjacent region. Here, the decision was made that such assumptions would be documented, and an option provided within the use interface to verify with or without this assumption being included as an additional constraining formula when verifying.

Optimised encoding of safety properties. During tests, it was observed that the underlying translation of safety properties was simpler/more efficient if properties were expressed in a particular manner. For example,

$$\forall x \,.\, \varphi(x) \rightarrow \big(\forall y \,.\, \psi(x, y) \rightarrow \xi(x, y)\big)$$

leads to faster verification than a property of the form

$$\forall x, y \,.\, \big(\varphi(x) \wedge \psi(x, y)\big) \rightarrow \xi(x, y)\big)$$

We believe this is due to the falsified cases of the precondition being large when a condition is expressed for a concrete interlocking.

Invariants based on design decisions. Finally, optimisations to the verification procedure were also devised thanks to extended discussions around variables within the Ladder Logic program. Inevitably, some variables exhibit inherent relationships thanks to design decisions. A typical example would be a classic SR latch where, if the reset bit is set, we know the output will be reset. This can allow for constraints on these variables to be added as invariant formulae to the model checking procedure, in turn constraining the state space. Here we have yet to systematically analyse the effect of this on verification.

4.4 Fully functional prototype at Siemens Mobility

As documented in our REF 2021 impact case study [18], there is now a fully functioning Ladder Logic Verifier running at Siemens Mobility. It has roughly 350 safety principles implemented, taken from various standards and developed from Siemens test objectives. This 2^{nd} prototype is fully integrated into the Siemens Mobility ecosystem of IXL development tools and has been demonstrated in Siemens Mobility operational environment through the verification of about 10 interlockings with up to 12 000 rungs and 75 000 variables. Safety checking of one IXL takes about two hours. The verification approach has uncovered mistakes in IXLs that Siemens Mobility deems non-detectable by testing.

One unresolved challenge is that there are safety properties with can neither be decided by inductive verification nor by bounded model checking: inductive verification fails; and bounded model checking does not find a counterexample within reasonable bounds, meaning the property may or might not hold. In our verification practice, depending on the IXL under discussion, about 35–40% of all properties fall into this category.

Outcomes:

The 2^{nd} prototype is fully integrated in the Siemens Mobility ecosystem. Large IXLs can now be verified against 350 properties within two hours. But 35–40% of an IXL's safety properties cannot be decided.

5 Future development

Though our 2^{nd} prototype is a big step towards the ultimate goal to automatically verify Ladder Logic programs, there are still a number of aspects to be addressed before it is complete and qualified (TRL 8) and is proven in the operational environment (TRL 9).

In terms of completeness, it is necessary to include decision procedures that allow to effectively prove or disprove *all* safety properties. Section 5.1 and Section 5.2 discuss ongoing work to address this. In terms of introducing the technology, key enablers and inhibitors to the acceptance and adoption of utilising the 2^{nd} prototype within Siemens Mobility need to be identified, and, based on a data collection, it remains to be shown that overall it is beneficial to use this new technology. Our plans in this direction are discussed in Section 5.3.

5.1 IC3 algorithm

In order to address the 35–40% of safety properties that cannot be decided with the verification technologies implemented in the 2^{nd} prototype, the MRes project by Bryant [2] investigated if Bradley's IC3 algorithm [1] would offer a

suitable solution. Bryant could show that with IC3, all properties can effectively be decided. Here, runtime per verification task was smaller than a second.

5.2 Invariant finding via reinforcement learning

It is accepted that so-called invariants, properties which hold for all states under which a system operates, can help reduce occurrences of false positives. However, automated deduction of these invariants remains a challenge. We are currently exploring the use of reinforcement learning [19] where agents are used to build a dataset of observed states. These observations are then used to compute correlation coefficients between all variables composing a Ladder Logic program. This in turn allows proposals for candidate invariant properties [14,15].

5.3 Measuring cost and benefit

In an interdisciplinary project, spanning Business Management, Engineering and Computer Science we plan to identify key enablers and inhibitors to the acceptance and adoption of utilising the 2^{nd} prototype within Siemens Mobility. To this end, we will embed a longitudinal comparative study to align data collection with Siemens Mobility (i.e., within each development and testing cycle, a parallel stream embedding the Ladder Logic Verification will be added in order to gather data enabling robust comparisons). As part of this data collection, insight will be gained through the interaction with the technical and management teams responsible for setting up and carrying out contracts to understand the complex dynamics at play and identify factors that may prevent the organisation from accepting and embracing the new methodology. Such interaction will be both of a formal and informal nature, spanning meeting observations, interviews, focus groups and semi-quantitative surveys. The multi-faceted nature of the data gathered will enable the development of a robust business case for the introduction and adoption of the novel methodology within Siemens Mobility.

Ideal Outcomes:

Our verifier is integral to any IXL development at Siemens Mobility.
It efficiently decides all properties under discussion.
Rail authorities accept its proofs as evidence within certification.

6 Summary

For us, one important learning outcome of this long-lasting technology transfer project is that it becomes only clear at the end of each phase what the next challenges will be. Having understood the theory (as given in Section 2), it was clear what transformations needed to be implemented and that counterexamples would need to be presented in a user friendly way, leading to the

1^{st} prototype (as discussed in Section 3). Driven by the need to integrate with the Siemens Mobility ecosystem and follow Siemens practices, the 2^{nd} prototype was created (as discussed in Section 4). Here, experience from the 1^{st} prototype aided in system architecture and data type choices. In addition, a closer working relationship with Siemens Mobility allowed for extensive experimentation and brought in deeper domain knowledge. This led to a number of optimisations. The Siemens Mobility research team is now convinced by the technology. However, there is future work to be done (see Section 5). A formal business case remains to be made demonstrating the advantages of the process. Furthermore, experience gained whilst evaluating the second prototype towards verification coverage demonstrates that further verification methods are needed.

Overall, our journey turns out to be a far longer one than expected. Originally, perhaps as naive academics, we thought that the 1^{st} prototype would be the end. When the decision came to build the 2^{nd} prototype, we thought that this would then be the end. Having now implemented the 2^{nd} prototype, we understand that there is still further to go towards a fully deployed technology. Overall, the journey up to now (excluding natural periods of inactivity) has taken about eight years and allowed us to gain expertise on the process of technology transfer.

Thanks to a long standing collaboration between industry and academia, it has been possible to explore the applicability of a set of formal methods to a challenge in the real world (with all its complexities). Though our endeavour continues, substantial progress has been made and there is honest belief by all involved that the above stated ideal outcomes will be achieved.

Acknowledgment The authors would like to thank Siemens Mobility for the long-standing, fruitful and successful research collaboration, the students and colleagues in the Swansea Railway Verification Group for their support and helpful feedback and discussions, and Erwin R. Catesbeiana (Jr.) for pointing out that logic is not everything.

References

1. Bradley, A.R.: Understanding IC3. In: SAT. Lecture Notes in Computer Science, vol. 7317, pp. 1–14. Springer (2012)
2. Bryant, H.: Exploring the ic3 algorithm to improve the siemens-swansea ladder logic verification tool (2023), MRes Dissertation (under submission), Swansea University
3. Chadwick, S., James, P., Moller, F., Roggenbach, M., Werner, T.: A journey through software model checking of interlocking programs. In: Leveraging Applications of Formal Methods, Verification and Validation: 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece, October 17–29, 2021, Proceedings. vol. 13036, p. 495. Springer Nature (2021)
4. Diaconescu, R.: Institution-independent Model Theory. Birkhäuser (2008)
5. Eén, N., Sörensson, N.: Temporal induction by incremental sat solving. Electronic Notes in Theoretical Computer Science **89**(4), 543–560 (2003), bMC'2003, First International Workshop on Bounded Model Checking

6. Fokkink, W., Hollingshead, P.: Verification of interlockings: from control tables to ladder logic diagrams. In: FMICS'98 (1998)
7. Gruner, S., Kumar, A., Maibaum, T., Roggenbach, M.: On the Construction of Engineering Handbooks – with an Illustration from the Railway Safety Domain. Springer (2020)
8. Programmable Controllers - Part 3: Programming languages (2003), IEC Standard 61131-3
9. J. Groote, S. v.Ṽlijmen, J.K.: The safety guaranteeing system at station hoornkersenboogerd (1995), technical Report, Utrecht University
10. James, P.: Sat-based model checking and its applications to train control systems (2010), MRes Dissertation, Swansea University
11. James, P., Roggenbach, M.: Automatically Verifying Railway Interlockings using SAT-based Model Checking. In: Proceedings of AVoCS'10. Electronic Communications 35 of EASST (2010)
12. Kanso, K.: Formal verification of ladder logic (2008), MRes Disseration, Swansea University
13. Lawrence, A.: Verification of railway interlockings in SCADE (2011), MRes Dissertation, Swansea University
14. Lloyd-Roberts, B., James, P., Edwards, M.: Mining Invariants from State Space Observations (2022), Extended abstract at 33rd Nordic Workshop on Programming Theory, NWPT
15. Lloyd-Roberts, B., James, P., Edwards, M., Werner, T., Robinson, S.: Improving Railway Safety: Human-in-the-loop Invariant Finding. In: Case Studies of HCI in Practice, CHI 2023. ACM (to appear)
16. Pantekis, F., James, P., O'Reilly, L., Archambault, D., Moller, F.: Visualising railway safety verification. In: Hasan, O., Mallet, F. (eds.) Formal Techniques for Safety-Critical Systems. Springer (2020)
17. Improving processes and policies in the UK railway industry, `https://results.ref.ac.uk/(S(ozgare1un34qrlg44nt3gsh3))/DownloadFile/ImpactCaseStudy/pdf?caseStudyId=5798`
18. Improving performance, safety and software development of railway signalling, `https://results2021.ref.ac.uk/impact/a117e4ed-a960-4dc6-8e13-8c98d8ea5aef?page=1`
19. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)