# Formal Methods for Industrial Interlocking Verification

Simon Chadwick
*Siemens Rail Automation UK*
Chippenham, UK
Simon.Chadwick@siemens.com

Phillip James
*Computer Science*
*Swansea University*
Swansea, UK
P.D.James@swansea.ac.uk

Markus Roggenbach
*Computer Science*
*Swansea University*
Swansea, UK
M.Roggenbach@swansea.ac.uk

Tom Werner
*Siemens Rail Automation UK*
Chippenham, UK
WT.Werner@siemens.com

***Abstract*—In this paper, we present an overview of research jointly undertaken by the Swansea Railway Verification Group towards verification techniques for automatically checking safety for train control systems. We present a comprehensive modelling of safety principles in first order logic. We conclude by applying verification methods developed by Swansea Railway Verification Group in order to check the modelled safety principles against a real world railway interlocking system.**

***Index Terms*—Verification; Model-Checking; PLC; Formal Methods in Industry.**

## I. Introduction

In 2004, the 18th IFIP World Computer Congress identified the railway domain as a Grand Challenge of Computing Science because it is of immediate concern and as it provides a set of generic, well-understood problems whose solutions would be transferable to various other application domains, e.g., process control in manufacturing, a.k.a. industry 4.0.

A major challenge in the railway domain concerns the verification of safety critical components. Here, in particular the so-called interlocking computer plays a significant role as it provides essential safety functions for railway signalling.

Described in terms of theoretical computer science, an interlocking computer is a relatively simple entity: it can be modelled as a finite automaton, a concept established in the 1960's; and it can be analyzed through using temporal logic to express safety properties and then applying the technique of model-checking, which as a field have both been actively researched since the 1980's.

In this paper, we provide proof of concept that these three fields have matured enough to be utilized in an industrial setting. Here, we focus on the challenge of how to make relevant safety properties accessible for model-checking. This is a process that requires the definition of a bespoke first order temporal logic. Relative to this logic, we can justify why our verification process is correct.

Further, we discuss some results that we achieved in a technology transfer project between academics from Swansea University and railway engineers from Siemens Rail Automation UK.

This work has been supported by Siemens Rail Automation UK.



## II. Background

An interlocking provides a safety layer for a railway. It interfaces with both the physical track layout and the human (or computerised) controller. The controller issues requests, such as to set a route. Upon such a request, the interlocking will determine if it is safe for the operation to be permitted. If it is safe then the interlocking will issue requests to change the physical track layout, informing the controller of the change. Whereas if it is unsafe, the interlocking will not allow the physical track layout to be changed, and will report back to the controller that the operation has not taken place as it would yield an unsafe situation.

Here, we consider TrackGuard Westrace™ interlockings that execute the following typical control flow:

```
initialise
while True do
    read (Input)
(*) State' <- Program(Input, State)
    write (Output) & State <- State'
```

After initialisation, there is a non terminating loop consisting of three steps: (1) Reading of `Input`, where `Input` includes requests from signallers and data from physical track sensors; (2) Internal processing: this depends on the `Input` as well as on the current `State` of the controller; Using these the next state `State'` is computed. (3) Committing of `Output`, which includes passing information back to the signaller, commands to change the physical track layout, as well as an update of the `State` of the controller. Thus, a TrackGuard Westrace™ interlocking follows the design principles of a so-called Programmable Logic Controller (PLC).

In the context of TrackGuard Westrace™ interlockings, `Input`, `Output`, `State`, and `State'` are sets of Boolean variables, where `Output` is a subset of `State'`. The current *configuration* of the controller is given by the values of all variables in the sets `Input` and `State`. The `process` step then depends on the current *configuration*. The TrackGuard Westrace™ interlocking realises this controller in hardware, where the steps `initialise` and `process` depend on the installed control software written in ladder logic.

A ladder logic program can be translated into a subset of propositional logic. This translation is straightforward: it

replaces graphical symbols by logical operators, a process which has been automated in [4]. For the rest of the paper we only deal with this representation in propositional logic.

## III. Capturing Safety Principles using Temporal First Order Logic

In order to perform verification of ladder logic programs provided by Siemens, we were required to define a set of safety properties. These safety properties describe restrictions in terms of generic elements of the track plan in order to assure some aspect of safety in the system.

Each ladder logic program prescribes certain state transitions, which relate to real-world actions in the railway system. When implemented for a concrete ladder logic program, each safety property places a restriction on the set of state transitions. The verification process will be to check if the state transitions prescribed by the ladder logic program are allowed by the safety property.

In order to formulate safety properties, we used a publicly available standards document, "Interlocking Principles" [5], as the basis for the formation of the safety conditions that will be used in this project. With consideration of the case studies that Siemens Rail Automation UK were providing, it was decided to select a subset of the safety properties in this document for use in our verification approaches. Each reference is given as a statement in plain English about the conditions that must hold before a movement authority can be granted. The meanings of the terms used in these statements are detailed in the main body of the "Interlocking Principles" document.

### A. A Note on Naming Conventions

While the elements of each property are detailed in the body of the "Interlocking Principles" document, the safety behaviour that is being examined by each reference is open to interpretation. Because of this, further discussion with Siemens Rail Automation UK engineers was required in order to translate these English language conditions into observable state transitions within the format of their ladder logic programs. With the understanding gained from our discussions with Siemens Rail Automation UK, it was possible to describe the English safety properties in terms of ladder logic variables.

Since these properties are generic, in order to carry out verification of a particular track layout and ladder logic program each safety property needs to be converted from its generic form into a concrete instance. Rather than describing the conditions in a general sense, the concrete forms of the safety properties describe actual elements of the track plan, using the variable names that appear in the ladder logic program. To perform this translation from English references into concrete safety properties, a table of variable names was established in order to substitute specific variables for the generic elements such as 'route', 'signal', or 'point'.

This process has the effect of generating many different formulas for a single safety property reference in the table, because a generic 'route' description in English will relate to a number of physical routes in the track plan, each requiring their own implementation of the safety property. For example, "All train detection devices in the route indicate the line is clear", describes a generic 'route' element in the track plan, and the concept of devices on this route indicating clear. When implementing this safety property for our case study, which contains seven 'Main class' routes, seven propositional formula are required, each describing the safety property in the context of a particular route.

### B. A Temporal FOL with Built-in Predicates

The first step for implementing concrete safety properties involves moving from English language descriptions of properties as they appear in "Interlocking Principles" into intermediate generic temporal first order logic formulae [5].

To describe the original English safety properties in a generic way, we employ a many sorted first order logic over states and their successors. We consider models at a point of time as:

$$Models : pairs(T, I)$$

where $T$ is a track plan and $I$ is a propositional model for all propositional variables of $T$, e.g. $I(P106.RL) = true/false$.

In these models, the track plan $T$ contains topological information on the railway section, such as the names of all track elements for example Signals, RouteNames, Points, and TrackSegments. It also contains information regarding the layout of these elements, such as which RouteNames originate at which Signal. Thus, the track plan T corresponds to the combination of a visual track layout, labelled with element names, and a route table that describes route and signal information. This topological information does not change over time.

The set of propositional variables in $I$ would then be formed by the application of variable naming conventions to the track elements in $T$. These models will be given to us through the execution of the ladder logic programs, with each cycle giving a new model.

By looking at subsequent execution cycles of the ladder logic program, from states $s_0, s_1, ...$, we can then form sequences of models that each correspond to a single state of the variables within the ladder logic program.

$$(T, I_0), (T, I_1), ...$$

We then introduce a signature containing:

- 4 sorts,
- 3 functions, and
- unary and binary predicates.

Sorts are interpreted in the track plan, for example:

$$Signal_T = \{S100, ...\} : \text{iff} \;\; T \; has \; signals \; S100, ...$$

Functions are interpreted in the track plan, for example:

$$routesOf_T(s) = \{r_1, ..., r_n\}$$

$$\text{iff } in \; T, \; signal \; s \; has \; routes \; r_1, ..., r_n$$

Unary and binary predicates are interpreted in the track plan for example:

$$p \; isInCorrectPositionFor_{T,I} \; r \quad holds \quad \text{iff}$$

- *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.

The unary predicates take an element of a sort as an argument. When instantiated with an element from a sort, the unary predicates interpretation is similar to that of Boolean variables within the ladder logic program. For example, the unary predicate $proceed(< Signal >)$ could be interpreted using a physical signal, say S100. The truth value of this unary predicate is then dependent of the value of the signal in a specific state. In the ladder logic program, there will be a corresponding Boolean variable that will eventually be used for the interpretation of this predicate. Each unary predicate and argument combination matches a single variable name within the ladder logic, such that the generic formula can be substituted for these variable names at a later stage to produce a concrete formula. In our example signal S100, the corresponding signal proceed variable in the ladder logic is found to be "S100.G".

In addition to unary predicates, a number of binary predicates were also defined. These binary predicates are solely used to relate Points to RouteNames, and are required due to the unique naming scheme used by points within the ladder logic. When describing the naming of points, in many variable classes, the orientation of the point influences the prefix or suffix that is used, i.e. whether the point should be in normal or reverse position. Since the required point orientation is information that is state dependent, interpreting a predicate with a track plan T is not sufficient. Instead, these binary predicates must be interpreted using some propositional model I, in which the required point orientations may be evaluated. The required Point orientation will be one of two cases shown in the binary predicate definition.

With the defined sorts, functions, and unary predicates, it is possible to describe the English safety properties in a generic logical form. By using the 'forall' quantifier over one or more sorts, a generic formulae may be written that formalises the safety property in question.

*Example:* A typical safety property would be: "*All train detection devices in the route indicate the line is clear*", which is one of the principles stated in [5]. Using the convention that a primed predicate denotes the next state, the above safety property can be formalized as:

$$\begin{array}{l} \forall s \in Signal, rn \in RouteName, t \in TrackSegment : \\ \quad rn \in routesOf(s) \wedge t \in tracksOf(rn) \implies \\ \quad\quad ((not(proceed(s)) \wedge proceed'(s) \wedge set(rn)) \implies \\ \quad\quad\quad (not(occupied(t)))) \end{array}$$

Both primed and unprimed symbols are used within this formula, thus referring to two subsequent states – this is the 'temporal' aspect of our logic. Therefore, two models are required to evaluate the truth value of this formula. Due to the relation to ladder logic execution, a further restriction must be enforced requiring that these states must be subsequent.

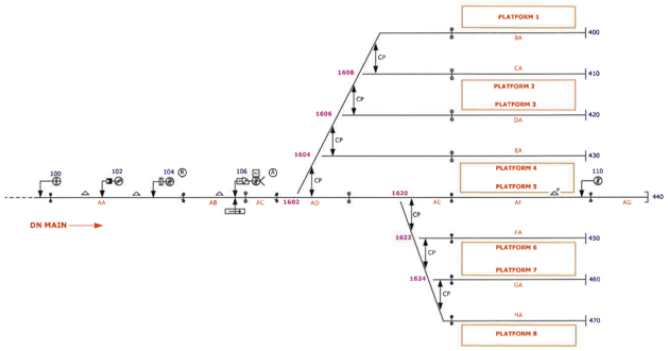


Fig. 1. *A sample trackplan.*

For two single models

$$(T, I), (T, I')$$

a formula $\phi$ holds if:

all unprimed symbols are interpreted over $(T, I)$,
and all primed symbols are interpreted over $(T, I')$, and
and under these interpretations $\phi$ evaluates to true.

For a countable sequence $\sigma =< (T, I_0), (T, I_1), ... >$ a formula is true if it holds for each pair

$$(T, I_i), (T, I_{i+1}), \; i \geq 0.$$

We write $\models_T \; \phi$ for $\phi$ being true over all model sequences with first component $T$.

### C. Semantics Preserving Formula Translation

For each generic formula, a translation process was formed that would produce propositional formulae with concrete variables for each of the case studies under inspection.

*Step 1:* Replace all universal and existential quantifiers by appropriate conjunctions and disjunctions, respectively, by using the topological information given through the trackplan. The resulting formula will be variable free, as all variables have been replaced by constant symbols corresponding to the finitely many elements of the track plan.

*Example:* For the trackplan in Figure 1, and the safety formula in the last example, we obtain:

$$\begin{array}{ll} & S100(AM) \in routesOf(S100) \\ & \wedge \; AA \in tracksOf(S100(AM)) \implies \\ & \quad (not(proceed(S100)) \wedge proceed'(S100) \wedge \\ & \quad set(S100(AM))) \implies \\ & \quad\quad (not(occupied(AA))) \\ \wedge & S100(AM) \in routesOf(S106) \\ & \wedge \; AA \in tracksOf(S100(AM)) \implies \\ & \quad (not(proceed(S106)) \wedge proceed'(S106) \wedge \\ & \quad set(S100(AM))) \implies \\ & \quad\quad (not(occupied(AA))) \\ \wedge & S100(AM) \in routesOf(S110) \\ & \wedge \; AA \in tracksOf(S100(AM)) \implies \\ & \quad (not(proceed(S110)) \wedge proceed'(S110) \wedge \\ & \quad set(S100(AM))) \implies \\ & \quad\quad (not(occupied(AA))) \\ \wedge & \ldots \end{array}$$

| Element | Property | Prefix | Suffix |
|---|---|---|---|
| Track | Occupied | T | .OCC(IL) |
| Route | Set | S<SIGNAL(ROUTE)> | .U |
| Signal | Proceed | S<SIGNAL> | .G |

Fig. 2. A typical variable naming scheme.

Here, *S100(AM)* is a route name, *S100* is a signal name, and *AA* is a track name from the trackplan as shown in Figure 1.

*Step 2:* Eliminate true premises; eliminate subformulae with false premises. After Step 1, the formula consists of a number of subformulae joined by conjunctions. Each of these subformulae involves an implication using elements of the fixed track plan and its associated route table, relative to which the premise of each every subformula can be evaluated.

*Example:* The first subformula from the Step 1 example is:

$$\begin{aligned} &S100(AM) \in routesOf(S100) \wedge \\ &AA \in tracksOf(S100(AM)) \implies \\ &\quad (not(proceed(S100)) \wedge proceed'(S100) \wedge \\ &\quad set(S100(AM))) \implies \\ &\quad\quad (not(occupied(AA))) \end{aligned}$$

According to the trackplan, the premise of this subformula is true (*S100(AM)* is a route that starts at signal *S100*; track *AA* belongs to route *S100(AM)* as route *S100(AM)* starts at signal *S100* and ends at signal signal *S104*, and track *AA* is on the path from *S100* to *S104* as can be seen on the trackplan). Thus, we keep

$$\begin{aligned} &(not(proceed(S100)) \wedge\ proceed'(S100) \wedge \\ &set(S100(AM))) \\ &\quad \implies\ (not(occupied(AA))) \end{aligned}$$

from the first subformula.

By examining another subformula resulting from Step 1, a case in which the premise evaluates to false can be found:

$$\begin{aligned} &S100(AM) \in routesOf(S106) \wedge \\ &AA \in tracksOf(S100(AM)) \implies \\ &\quad (not(proceed(S106)) \wedge proceed'(S106) \\ &\quad \wedge set(S100(AM))) \implies \\ &\quad\quad (not(occupied(AA))) \end{aligned}$$

Since route *100(AM)* is not contained within the routes of signal *S106*, the premise of this subformula is false. Thus, we delete the whole subformula.

*Step 3:* Next we replace all predicates with propositional variables according to a variable naming scheme for ladder logic programs.

*Example:* A result of the example in Step 2, was the subformula

$$\begin{aligned} &(not(proceed(S100)) \wedge proceed'(S100) \wedge \\ &set(S100(AM))) \implies \\ &\quad (not(occupied(AA))) \end{aligned}$$

Now we replace the state describing predicates with propositional variables:

$$\begin{aligned} &(not(S100.G) \wedge S100.G' \wedge S100(AM).U) \implies \\ &\quad (not(TAA.OCC(IL))) \end{aligned}$$

To this end, we apply a variable naming scheme as shown in Figure 2.

## IV. Experimental Results

Over the last decade, Swansea have developed a verification tool specifically for verification of ladder logic programs for railway interlockings [3], [4]. This tool is an assortment of software that each handles one aspect of an overall verification procedure. For this work, an adaptive maintenance phase took place focusing on adapting the software to support the newly presented format of safety properties.

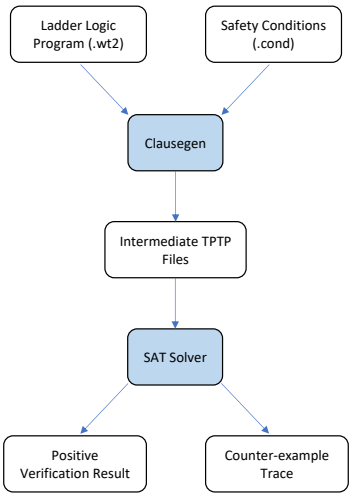


Fig. 3. The Software Structure of the Verification Tool

The core of the Swansea tool can be divided into two separate operations. The first operation is the translation from generic safety properties in temporal first order logic into concrete formulae in temporal propositional logic. In order to perform this translation, some encoding of a track plan is required that provides layout and naming information which are substituted into the generic formulae, see the transformations described in the previous section. The second operation performed by the tool is larger in scope, see Figure 3. Here the software accepts a ladder logic program, a safety condition file (containing at least one temporal propositional formula describing a safety condition), and a number of arguments describing the required verification approach. It then handles the verification process, including utilisation of a SAT-solver to produce the verification output resulting from the chosen methodology. The verification approaches that are supported include inductive verification [2]–[4], bounded model checking [2], [3] and temporal (or K) induction [2], [3]. As these approaches have been developed in other papers, here we simply refer the reader to those papers for details.

## V. Application to a Siemens Interlocking

To illustrate how the full verification approach works, the ladder logic for controlling a scheme plan provided by Siemens Rail Automation UK for illustrative purposes, has been verified against a series of newly modelled safety principles, see Figure 4 for a summative overview of which verification attempts were successful and which produce counter-examples. When verification is successful, there is no need for

further action. However, when verification produces a counter-example, an error analysis needs to take place. As the yellow and red areas in Figure 4 show, verification failed in a number of cases.

### A. Abstract Error Analysis

In principle there are a number of reasons why verification of a safety property might fail. They include:

1) Wrong encoding of the safety property in FOL.
2) Wrong use of names of propositional variables.
3) A deliberate deviation from the property in the ladder logic program.
4) A false positive.
5) A mistake in the ladder logic program.

A wrong encoding of a safety property might arise due to communication problems. Experts in formal methods are not necessarily experts on railway, and rail engineers have seldom received training in formal methods. Similarly, naming conventions in ladder logic programs are often not properly documented. For the rail engineer, the chosen conventions make sense, although the formal methods expert will have to take a given mapping without having a means of control at hand. However, as safety properties and naming conventions remain stable over longer periods of time, this kind of mistakes can be eliminated by use, i.e., after 'many' verification attempts have been carried out successfully, the proportion of these mistakes will decrease.

A similar argument will apply to the deliberate deviation from the property in the ladder logic program: this will not happen only once, but will happen only as an established programming practice. In this case, it would be adequate to change the safety property accordingly, in order to verify that the deviating behaviour has been encoded correctly.

A 'false positive' will arise when the safety property to be verified is not 'inductive', i.e., all reachable states are safe, however, there exists a safe, unreachable state with a transition into an unsafe state. In order to exclude false positives, one needs to add a suitable invariant to the verification. The effect of such an invariant is to reduce the considered state space, hopefully excluding all safe unreachable states with a transition to an unsafe state. Finding such invariants is a challenge, but can work, see, e.g., [1].

Only in the last case is there a need to actually change the ladder logic program. It takes experience and (manual) work, to isolate this case from the others.

### B. Concrete Error Analysis

In particular for our results, see Figure 4, Ref 1 highlights an incorrect use of propositional variables (reason 2). This was resolved after reviewing the counter-examples with Siemens Rail Automation UK: highlighted by the change of the row from yellow (top) to green (bottom).

Ref 22 highlights an incorrect encoding of the safety property in FOL (reason 1). This was resolved after reviewing the counter examples with Siemens Rail Automation UK: highlighted by the yellow row in the bottom table.

| Property Reference | Inductive Result | BMC Result |
|---|---|---|
| 1 | Yellow | Yellow |
| 6 | Red | Yellow |
| 12 | Yellow | Yellow |
| 22 | Red | Red |
| 32 | Green | Green |
| 35 | Yellow | Yellow |

| Property Reference | Inductive Result | BMC Result |
|---|---|---|
| 1 | Green | Green |
| 6 | Red | Yellow (*) |
| 12 | Yellow | Yellow (*) |
| 22 | Yellow | Yellow (*) |
| 32 | Green | Green |
| 35 | Green | Green |

Fig. 4. Results before (top) and after (bottom) consultation with Siemens Rail Automation UK engineers. The left hand column shows the results of inductive verification, whilst the right shows the results of bounded model checking for 50 steps. Green means that all concrete instances of a safety principle were verified; Yellow means that some of the concrete instances were verified, others not; Red means all concrete instances were not verified.

Finally the star ($*$) in the right column of the bottom table indicates that some instances fail due to reason 3 or 4. Here further scientific investigation is on going into automated methods for excluding false positives.

Overall we could verify just under 50% of the properties using inductive verification, whilst for 85% of the properties, no counter examples were found whilst running bounded model checking.

## VI. Conclusions and Perspectives

The verification results achieved show that the technology developed is ready for implementation in interlocking design processes: in terms of run-time and memory-usage, it is possible to verify actual interlocking programs with thousands of lines of code. However, what currently is missing is that verification can't fail due to the reasons "Wrong encoding of the safety property in FOL" or "Wrong use of names of propositional variables". We are positive, that Siemens Rail Automation UK engineers together with Swansea academics will run enough experiments to achieve this aim. In future work we plan to add automated invariant finding, c.f. [1].

*Acknowledgement.* We thank Erwin R. Catesbeiana (Jr.) for inspiring our summative visualization approach.

## References

[1] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Infinite-state invariant checking with ic3 and predicate abstraction. *Formal Methods in System Design*, 49(3):190–218, Dec 2016.
[2] Phillip James. SAT-based Model Checking and its applications to Train Control Software. Master's thesis, Swansea University, 2010.
[3] Phillip James and Markus Roggenbach. SAT-based Model Checking of Train Control Systems. In *Calco-Jnr 2009*, March 2010.
[4] Karim Kanso. Formal verification of ladder logic. Master's thesis, Swansea University, 2008.
[5] Railway Group Standard. Interlocking principles, 2003. Standards Document GK/RT0060.