Automated Verification of Signalling Principles in Railway Interlocking Systems[1]

Karim Kanso[2] Faron Moller[3] Anton Setzer[4]

Dept. of Computer Science
Swansea University
Swansea, UK

Abstract

In this paper we present a verification strategy for signalling principles for the control of a railway interlocking system written in ladder logic. All translation steps have been implemented and tested on a real-world example of a railway interlocking system. The steps in this translation are as follows: 1. The development of a mathematical model of a railway interlocking system and the translation from ladder logic into this model. 2. The development of verification conditions guaranteeing the correctness of safety conditions. 3. The verification of safety conditions using a satisfiability solver. 4. The generation of safety conditions from signalling principles using a topological model of a railway yard.

Keywords: ladder logic, railway interlocking systems, SAT solvers, verification, automated theorem proving, signalling principles, safety properties.

1 Introduction

In this paper we summarise the work carried out in a small case study, some of which is reported in [9]. Within the scope of this project we have written software which allows for the fully-automated verification of railway interlocking systems using SAT solver technology. This software has been applied to the interlocking system of a small UK railway yard.

Westinghouse Rail Systems, the project sponsor, is currently interested in applying formal methods to the development of software controlling the equipment on the railway, i.e. *signals* and *points*. Software is developed using *ladder logic*, a low level language representing Boolean-valued assignments. This software is simulated by experienced signalling engineers to look for errors. The engineers will try many scenarios, which are typically listed in signalling books.

[1] The research described in this paper was carried out as a Master of Research (MRes) project by the first author under the supervision of the second and third authors, and was supported by Westinghouse Rail Systems, Chippenham, UK.

[2] Email: cskarim@swansea.ac.uk

[3] Email: F.G.Moller@swansea.ac.uk

[4] Email: A.G.Setzer@swansea.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

This technique, commonly used in industry, catches many flaws in software, but does not guarantee correctness of the ladder implementing signalling principles. This research was commissioned to determine whether it is feasible to apply formal methods to ladder logic and to verify that signalling principles hold in a ladder logic program.

Part of the research was to implement a prototype verification system. This system takes as input: the ladder logic to verify, a model of the railway yard, and a signalling principle; if a counter-example is identified, the system provides a LaTeX document detailing the state of the system when the counter-example appears. LaTeX is used, as opposed to simply outputting the state of the system in plain text, so that the produced counter-examples can be elegantly formatted and presented to make it easier for an engineer to understand.

Signalling principles for the UK railway industry are written in plain English. A second component of the research was to define a formal language in which to precisely represent signalling principles. We have written a program which takes signalling principles defined in this language, and produces safety conditions for which the ladder logic is to be verified.

Overview

This paper is structured as follows. We start by providing some background knowledge on railways and interlocking systems. We then provide a discussion of the verification technique used in this research. Then a discussion of the production of safety conditions from signalling principles follows. Finally, we present a survey of related work and some conclusions to the research carried out.

2 Railways

Before explaining how the verification system works, we will provide some background information about the railway domain.

Railways are split up into railway yards – ie, train stations and depots – and open lines connecting the yards. An example railway yard is presented in Fig. 1. This research focuses on interlocking systems controlling railway yards. A railway yard is made up of the following components:

Track Segments. Train lines are split up into segments, and each segment is associated with a *track circuit* which can detect if a train is on the segment.

Signals. Signals are placed between track segments, and a signal is only visible from one direction. Signals show different aspects; these aspects inform the train driver about the state of the line ahead.

Points. Points are a special type of track segment used to merge two lines into one line. A train can drive over a set of points if it has been *locked*, i.e. reached a definite position, and has been so locked into position physically and virtually by software. The two possible positions of a set of points, when it is locked, are called *normal* and *reverse*. The *normal* position is when the points allow trains to travel straight over the points and *reverse* is when the points allow trains to

branch off of, or on to, the line.[5] Each set of points in a railway yard is given a unique identifier in addition to the unique track segment identifier.

Routes. Routes consist of a sequence of sequentially-connected track segments that begin and end at signals, possibly through a set of points. Routes are defined by *control tables* which are created when a railway yard is designed. Routes can be *set* to indicate that a train is using – or about to use – the route.

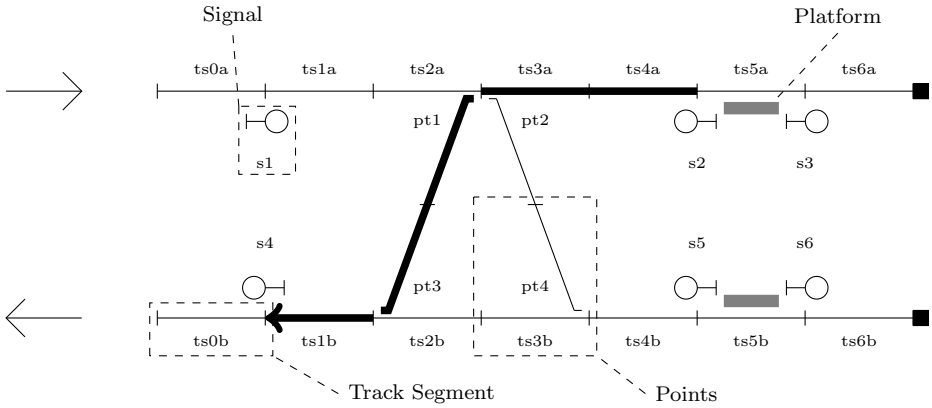

Fig. 1. An example railway yard, all parts of the yard are named. The grey boxes on the right are platforms. The arrows on the left side indicate the direction trains are supposed to travel down the lines. The black boxes on the right are "end of line" markers. The "lollipops" named `s1`, `s2`, ..., `s6` are signals. The big arrow depicts route C, see Table 1.

Track plans, such as presented in Fig. 1, describe how these components are topologically configured. The operation of the various components in a railway yard is defined using control tables. These contain information about when a route can be set, positions of the points, and the aspect a signal should display. Control tables are responsible for enforcing the signalling principles. Table 1 gives an example control table defining four routes.

Route C from the control table is graphically depicted as a large arrow in Fig. 1. Route C starts at signal `s2` and ends at `s4`, and spans track segments `ts4a`, `ts3a`, `ts2a`, `ts2b` and `ts1b`. As a safety precaution, track segment `ts0b` is also required to be unoccupied before a train is allowed to enter the route. Track segments `ts3a`, `ts2a` and `ts2b` are also points; `ts3a` must be locked in the normal position and `ts2a`, `ts2b` must be locked in the reverse position. Points in this scenario are always moved together in pairs so that point `ts3b` must also be locked in the normal position before a train is allowed to enter the route.

3 Interlocking Systems

Railway interlocking systems are designed to implement the constraints in the control tables. The interlocking systems with which this research is concerned are programmed using *ladder logic*, a graphical representation of a sequence of Boolean

[5] Although in many situations, like the example in Fig. 1, it is clear which position is supposed to be normal and which to be reverse, in general it is a matter of convention as to how to make this decision (for instance in the situation where a main line forks into two lines).

G = Green and R = Red

Route Name	Start	Exit	Signal Aspect	Condition	Track Segments	Points Normal	Points Reverse
A	s1	s3	G	Route Set	ts1a, ts2a, ts3a, ts4a, ts5a, ts6a	ts2*, ts3*	
			R	Route Unset			
B	s1	s6	G	Route Set	ts1a, ts2a, ts3a, ts3b, ts4b, ts5b, ts6b	ts2*	ts3*
			R	Route Unset			
C	s2	s4	G	Route Set	ts4a, ts3a, ts2a, ts2b, ts1b, ts0b	ts3*	ts2*
			R	Route Unset			
D	s5	s4	G	Route Set	ts4b, ts3b, ts2b, ts1b, ts0b	ts2*, ts3*	
			R	Route Unset			

Table 1
An incomplete control table for the railway yard of Fig. 1. The 'Start' and 'Exit' columns indicate signals the route begins and ends at; the 'Track Segments' column displays track segments that must be unoccupied for a train to enter the route. The two 'Points' columns together show the position that points must be in for a route to be set. ts$n$* is short hand for ts$n$a and ts$n$b. Route C is depicted in Fig. 1.

assignments

$$x_1 := \varphi_1; \quad \cdots \quad x_n := \varphi_n$$

where each φ_i is a propositional formula with variables taken from the set of input, output and intermediate propositional variables (latches).

The Boolean-valued assignment $z := (w \wedge \neg x) \vee y$, as it would be presented in ladder logic, is graphically depicted in Fig. 2. The variables w, x, y and z

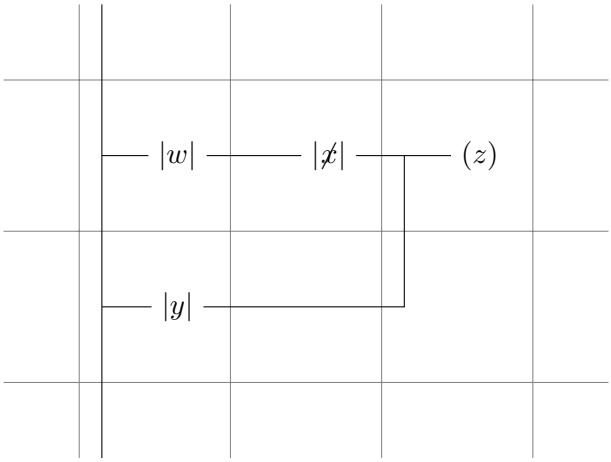

Fig. 2. Assignment Expressed in Ladder Logic

represent latches (*propositional variables*), $\not{x}$ is a negation, and the brackets around z indicate that it is the resultant. Literals in series such as $w \wedge \neg x$ in Fig. 2 represent conjunctions and literals in parallel represent disjunctions. The diagram's semantics are very similar to that of a circuit diagram, as ladder logic was originally developed to program microchips.

A ladder is executed by a program of the form

```
Initialise;
while(true){
  output();
  input();
  x_1 := φ_1;
      ⋮
  x_n := φ_n}
```

In the initialisation phase, some variables are set to initial values, while others remain undefined. A perpetual loop is then entered in which the following steps are carried out: the values of the output variables are sent to the signals, points, etc.; the input variables are set to the inputs (states of buttons from the control panel, sensors from the track segments, sensors from the points, etc.); and the ladder is executed. Note that, while executing the assignments, the output variables are not modified; therefore, correctness is only required at the end of each execution of the ladder. (The system need not be safe directly after initialisation, since the system will be used by trains only after the ladder has been executed a given number of times, say n times. We require that the system is correct after at least one execution of the ladder, but it would be sufficient to require correctness after at least n steps.)

4 Verification

Verification of safety properties in systems defined with ladder logic can be achieved in a number of different ways. Ladder logic is conceptually trivial to translate into propositional logic; this is exploited to allow the verification to be performed within the framework of propositional logic. Thus, safety conditions to be verified are also defined in propositional logic.

The safety conditions are propositional formulæ in which the atomic propositions range over the atomic propositions within the ladder. In this paper, ψ is used to denote a safety condition, or the conjunction of safety conditions.

To prove the correctness of a safety condition ψ, we need to show that ψ holds after executing the ladder n times for every $n \geq 1$. Note that ψ is not required to hold when $n = 0$ because the initial state is allowed to violate the safety conditions. In our system, we prove this by induction: we show that ψ holds after initialisation and one execution of the ladder; and that, if ψ holds before an execution of the ladder, it holds afterwards as well. This technique is a strengthening of the first method introduced by Fokkink in [8]; see our Section 7 for a detailed comparison of the two approaches.

More formally, we define a propositional formula ψ_I which defines the initial state of the system (the ladder logic program does not assign a fixed value to all variables in the initial state). For instance, if variables x, y, z are initially set to values a, b, c, then $\psi_I = (x \leftrightarrow a) \wedge (y \leftrightarrow b) \wedge (z \leftrightarrow c)$. Furthermore, we define a formula $\varphi_{\mathcal{L}}$ which models the execution of the ladder. Assuming for simplicity that the x_i are all different and represent the state of variables before execution of the

ladder, then $\varphi_{\mathcal{L}}$ has the form

$$(x'_1 \leftrightarrow \varphi'_1) \wedge \cdots \wedge (x'_n \leftrightarrow \varphi'_n)$$

Here, x'_i are new variables representing the state of the variables after execution; and φ'_i is the result of replacing $x_1, \ldots, x_{i-1}$ by $x'_1, \ldots, x'_{i-1}$ in φ_i. The first proof formula, corresponding to the base case, has the form

$$\psi_I \wedge \varphi_{\mathcal{L}} \rightarrow \psi'$$

where ψ' is the result of replacing each atomic proposition x in ψ by x'. It expresses that after the first iteration of the ladder the interlocking system is in a safe state. The second formula is the inductive step, and proves that from an arbitrary state where the safety condition ψ holds, after executing the ladder the safety condition still holds.

$$\psi \wedge \varphi_{\mathcal{L}} \rightarrow \psi'$$

These two formulæ should always hold to prove correctness of the safety condition in the ladder. When employing a SAT solver, both formulæ are negated; thus, if the safety condition holds, neither formula should be satisfiable.

Example 1
If

- the initialisation sets variable x to $true$:
  $\psi_I := x \leftrightarrow true$
- the safety condition is $y \leftrightarrow x$:
  $\psi := y \leftrightarrow x \quad \text{and} \quad \psi' := y \leftrightarrow x'$
- and the ladder has one assignment representing $x := y$:
  $\varphi_{\mathcal{L}} := x' \leftrightarrow y$

then we obtain the formulæ

$$\big((x \leftrightarrow true) \ \wedge \ x' \leftrightarrow y\big) \rightarrow y \leftrightarrow x'$$

and

$$\big((y \leftrightarrow x) \ \wedge \ x' \leftrightarrow y\big) \rightarrow y \leftrightarrow x'$$

which, in this toy example, are provable. For the verification, we use a SAT solver to search for a satisfying assignment which falsifies one of the two formulæ above.

Limitations
The proof system described above suffers from the problem that we may obtain a false positive when trying to verify a safety condition, that is, a counter-example which can not actually arise. There may be a state in which the safety condition holds, but such that after the execution of the ladder the safety condition is violated; however it may be that the original state is *unreachable*. In order to find out whether the counter-example is genuine, it is necessary to find a trace from the initial state to the identified counter-example. This is not straight forward with our inductive proof system [6].

[6] Solutions for producing error traces are known but have not been explored in this research. One such solution is to use time copies as introduced by Fokkink in [8] or to apply a model checking technique that

To mitigate the identification of false positives the inductive statement is relaxed to:

$$(\psi \wedge \varphi_{\mathcal{L}} \wedge \psi_{\text{Inv}}) \rightarrow \psi'$$

where ψ_{Inv} is an invariant of the ladder. We used two orthogonal techniques for identifying such an invariant ψ_{Inv}:

1) Not all choices of input variables correspond to physically possible states. An example is a 3-way switch which has 3 positions A, B, C (e.g. "control from central panel", "control by local station" and "control by emergency panel"). The output of such a switch would then be represented by 3 variables, one indicating whether A was chosen, one for B and one for C. At any time at most one of A, B or C is chosen (possibly none of these is chosen, e.g. if the switch is between positions). Therefore we obtain the invariant

$$\begin{aligned} & A \rightarrow (\neg B \wedge \neg C) \\ \wedge \;\; & B \rightarrow (\neg A \wedge \neg C) \\ \wedge \;\; & C \rightarrow (\neg A \wedge \neg B) \end{aligned}$$

2) Some combinations of variables are unreachable. When looking carefully at false positives, it was usually found that some variables were in a state which should not be reachable, typically when two variables are related to each other; e.g. if a signal's green aspect is activated, its red aspect should not be activated, and vice versa. In this instance we would obtain the invariant

$$\text{signal}_{\text{i}}\text{\_is\_red} \;\leftrightarrow\; \neg\text{signal}_{\text{i}}\text{\_is\_green}.$$

When such a possible invariant ψ_{Inv} is discovered we try to prove that it is in fact an invariant, i.e. that it always holds:

$$(\varphi_{\text{I}} \wedge \varphi_{\mathcal{L}}) \rightarrow \psi'_{\text{Inv}} \qquad \text{and} \qquad (\psi_{\text{Inv}} \wedge \varphi_{\mathcal{L}}) \rightarrow \psi'_{\text{Inv}}$$

If this is provable, then we can assume that this invariant holds before executing the ladder. Alas, it is a major area of research to efficiently identify invariants automatically.

5 Translating Signalling Principles to Safety Conditions

Signalling principles, as used in this research, refer directly to the railway industry. They are used as heuristics by the designers and are typically written in a natural language as precisely as possible.

One aim of the research is to define a formal unambiguous language with which to formulate signalling principles. A typical signalling principle would be:

"Points in a railway yard should not be set to
the normal and reverse positions simultaneously."

successively identifies sets of reachable states from the initial state to the counter-example, yielding the computation path [1,4].

Normal and *reverse* are the two possible positions of a set of locked points. Signalling principles do not refer directly to any specific railway yard, or the entities within them. First-order logic with general predicates is ideal for formally expressing these principles; the above principle would be translated to:

$$\forall\, pt \in Points : \neg\, [normal(pt)\ \wedge\ reverse(pt)]$$

These first-order formulæ need to be translated into a propositional formula (*safety condition*); to do this we build a topology model of the railway yard for which the interlocking system was designed. A Prolog database is used for this topology model. The entities in a railway yard are given names, and relations are used to model the topographic aspect. For instance, two connected track segments would be related using the binary predicate `connected`. For this research, the track plans and control tables were (manually) converted into a Prolog database. This database can then be automatically queried to help translate the signalling principles.

The translation has two steps: the first removes quantification, and the second resolves predicates into literals from the ladder or a constant Boolean value depending on the context. Variables in the signalling principle range over finite domains, as all railway yards are finite. Thus, universal quantification can be replaced by a finite conjunction, and existential quantification can be replaced by a finite disjunction. The topology model would be queried for a finite set of quantified values. For instance the variable pt in the example signalling principle introduced ranges over the domain of all points in the railway yard.

Secondly, the predicates are resolved into literals. This is done by specifying a list of predicates along with how they are reduced. This list is unique for each railway yard, as different railway yards follow different naming conventions. For instance, the predicate $normal(pt)$ used in the example signalling principle would be reduced to a literal "$pt.Normal$" by means of a string concatenation operation. Predicates that are not specified in the railway yard specific list are resolved using Prolog, and the topology model, to a constant Boolean value (see Example 2 below). Thus, the second class of predicates greatly simplifies the formulation of signalling principles, as a safety condition can be given a *guard*.

Example 2

Consider a signalling principle such as

"All points that are part of a route must be locked if the route is set."

This is formalised as

$$\forall\, pt \in Points : \forall\, rt \in Routes : \texttt{point\_part\_of}(pt, rt) \rightarrow [set(rt) \rightarrow locked(pt)]$$

where the predicates $set(rt)$ and $locked(pt)$ are reduced to literals; and $\texttt{point\_part\_of}(pt, rt)$ is reduced to *true* if point pt is part of route rt within the topology model, and to $false$ otherwise. In this case, the verification consists of proving that $set(rt) \rightarrow locked(pt)$ holds for all cases where point pt is part of route rt.

Example 3
Consider a simple railway yard with only two points *pta* and *ptb* and a signalling principle:

$$\forall\, pt \in Points : \neg[normal(pt) \ \wedge \ reverse(pt)]$$

After removal of the quantification and predicates, the following safety condition is produced:

$$\neg[pta.Normal \ \wedge \ pta.Reverse] \ \wedge \ \neg[ptb.Normal \ \wedge \ ptb.Reverse]$$

In order to identify more precisely the reason for a possible counter-example, the safety conditions – which often form a large conjunction – are split into their conjuncts which form more specific safety conditions.

6 Implementation

The software implemented for this research takes as input a signalling principle, an interlocking system's ladder logic, and a topology model; using these inputs, it generates clause sets and starts the verification. LaTeX documentation is produced if a counter-example is identified. The SAT-Solver used for this project is called OKSolver, written by Kullmann [12,10], which is part of the OKlibrary [11]. The interlocking system verified has 331 assignments and 599 variables. For illustration purposes, two signalling principles have been verified; Table 2 contains information about the verification of the clauses. The first section in the table verifies that the interlocking system can never move the points to the normal and reverse position in the same execution cycle. The second section shows that counter-examples have been identified while attempting to verify that *if a point is occupied, then it is locked into position*. This second signalling principle is only for demonstration purposes and does not mean the railway is unsafe, as the proof system allows for trains to magically appear and disappear. Thus, if a point is not locked, then the SAT-Solver will place a train on the point, thus creating a counter-example.

Interestingly, the first signalling principle, when the clause sets are all unsatisfiable, has a very fast running time while verifying the clause sets. The second signalling principle, when the clause sets are all satisfiable, has a greater average running time, especially through the inductive steps.

7 Related Work

There have been many attempts to apply formal methods to railways and their associated interlocking systems. Indeed, this is the subject of the TRain *Grand Challenge* proposed by Dines Bjørner [3].

Eriksson has applied formal methods to the problem with great success for over ten years, notably on behalf of Banverket (the Swedish National Rail Administration) [5,6,7]. This approach works by creating two mathematical models: the first is that of the interlocking system and consists of rules, and the second is of the topological aspects of the railway yard for which the interlocking system has been designed. Verification proceeds by proving that a signalling principle holds for the interlocking system model in the topology model of the railway yard. The NP-Tools

Clause Set	**Number of Clauses**	**Number of Variables**	**OKSolver Running Time** (*Seconds*)
pointsNotNormalAndReverse0	14713	4076	0.06
pointsNotNormalAndReverse0.ind	12916	3559	0.06
pointsNotNormalAndReverse1	14713	4076	0.13
pointsNotNormalAndReverse1.ind	12916	3559	0.14
occupiedPointsLocked0	14713	4076	0.25
occupiedPointsLocked0.ind	12930	3560	1.34
occupiedPointsLocked1	14713	4076	0.21
occupiedPointsLocked1.ind	12930	3560	1.33
occupiedPointsLocked2	14716	4076	0.25
occupiedPointsLocked2.ind	12930	3560	1.37
occupiedPointsLocked3	14713	4076	0.27
occupiedPointsLocked3.ind	12930	3560	1.3

Table 2
Clause sets and there verification time, the clause sets in italic are satisfiable. Clause sets that end with *ind* are the inductive step of the verification, those without are the base cases.

software produced by the company Prover [7] was used for the verification [5]. NP-Tools is a collection of tools packaged with a proof engine; these tools translate various problems into an acceptable format for the proof engine to process. The proof system implemented by NP-Tools is documented in [15]. NP-Tools has been used by many other companies for formal verification of critical systems such as ADTranz, Saab and Volvo.

Morley applied formal methods to the British Rail Solid State Interlocking (*SSI*), focussing on safety properties and communication protocols between the SSI's [14,13]. Our approach is somewhat different as we focus on the low level Boolean logic whereas SSI's are programmed at a high level with a language which merges the logic with geographic data.

Fokkink demonstrated how an interlocking system programmed using ladder logic can be automatically verified to ensure that it implements the control tables correctly [8]. This work did not cover the direct verification of signalling principles; only safety conditions that were derived from the control tables were verified. The paper discusses two verification techniques. The first proves that a safety condition is a logical consequence of executing the ladder. Let $\varphi_{\mathcal{L}}$ be a model of the ladder in propositional logic and ψ be a safety requirement. The proof obligation used by

[7] www.prover.com

Fokkink is

$$\varphi_{\mathcal{L}} \rightarrow \psi'$$

If this obligation holds it proves that *after any execution of the ladder the safety requirement will always hold,* even if the system was in an unreachable state before executing the ladder. Note that our approach only demands that the obligation holds if, before an execution of the ladder, the system was in the initial state or in a state where the safety requirements hold as well. Our approach, therefore, restricts the number of states for which the safety condition is required to hold to a smaller subset of states which contains all reachable states and possibly some unreachable states. By adding invariants, we further cut down the number of unreachable states to be considered, therefore reducing the number of false positives.

The second technique introduced by Fokkink creates *time copies* of the propositional model of the ladder. He introduces variables $x_i(j)$ denoting the state of variable x_i after j executions of the ladder[8]. A time copy $\varphi(i)$ would be the same as φ with all of the atomic propositions x in φ replaced by $x(i)$. This technique does not show that after any execution of the ladder the safety requirement will hold, but only after a finite number k of executions of the ladder. The proof obligation is

$$\varphi(0) \wedge \varphi(1) \wedge \ldots \wedge \varphi(k) \rightarrow \psi(k)$$

This technique can be used to prove temporal safety requirements, but is deprecated as such safety conditions are verified for only a finite number of iterations; there will always be uncertainty as to whether the safety requirements hold beyond k iterations of the ladder. However, if a counter-example is found, then it is the case that the counter-example is reachable, and from a falsifying assignment we obtain a trace from the initial state to it.

8 Conclusion

Our approach was applied to a model provided by our industrial sponsor of a modest yet typical railway yard with 331 assignments and 599 variables, representing a station with two platforms and one railway line with two tracks feeding into it. The running time of the SAT solver itself was never longer than a couple of seconds. We were able to prove a large variety of safety conditions. We found some counter-examples, which were already known to the company but recognised not to be safety critical, being intermittent and occurring for only one cycle of the ladder. In order to prove that these counter-examples really occur only for at most one cycle, we could adapt the proof obligation and prove that if the system is in a state in which the safety condition ψ does *not* hold, then it *will* hold after a single execution of the ladder. The proof formula would be

$$\neg\psi \wedge \varphi_{\mathcal{L}} \rightarrow \psi'$$

and we could restrict it to states fulfilling the invariant, i.e.

$$\neg\psi \wedge \psi_{\text{Inv}} \wedge \varphi_{\mathcal{L}} \rightarrow \psi'$$

We do not know how well our approach scales up, since we have only applied it to a modest railway yard. Current interlocking systems being developed have

[8] So in our notation x_i denotes $x_i(0)$ and x_i' denotes $x_i(1)$.

over 3000 assignments. We do not anticipate any serious problems although the nature of the satisfiability problem means that the computational complexity will grow exponentially when attempting to verify interlocking systems with more and more assignments.

This project demonstrates that automated verification of railway interlocking systems, at least for smaller examples, is feasible. The main advantages of our approach is its simplicity and that it verifies safety at the lowest level – the level at which it is actually executed.

References

[1] Baier, C. and I. Katoen, J.P., "Principles of Model Checking," The MIT Press, 2008.

[2] Biere, A., M. Heule, H. van Maaren and T. Walsh, "Handbook of Satisfiability," IOS Press, Amsterdam, (to be published) 2008.
URL http://www.st.ewi.tudelft.nl/sat/handbook/toc.html

[3] Bjørner, D., *TRain: The Railway Domain*, in: *Building the Information Society*, IFIP International Federation for Information Processing **156/2004** (2004), pp. 607–611.
URL http://www.springerlink.com/content/527p7237102w5741/

[4] Clarke, E., O. Grumberg and I. Peled, D.A., "Model checking," Springer, 1999.

[5] Eriksson, L., *Formal Verification of Railway Interlockings*, Swedish National Rail Administration Technical Report **4** (1997).

[6] Eriksson, L., *Formalising Railway Interlocking Requirements*, Swedish National Rail Administration Technical Report **3** (1997).

[7] Eriksson, L. and M. Fahlén, *An Interlocking Specification Language*, ASPECT IRSE **99** (1999).

[8] Fokkink, W., P. Hollingshead, J. Groote, S. Luttik and J. van Wamel, *Verification of interlockings: from control tables to ladder logic diagrams*, Proceedings 3rd Workshop on Formal Methods for Industrial Critical Systems (FMICS'98) (1998), pp. 171–185.

[9] Kanso, K., "Formal Verification of Ladder Logic," Master's thesis, Swansea University, Swansea, SA2 8PP, UK (2008).

[10] Kullmann, O., *Investigating the behaviour of a SAT solver on random formulas*, Technical Report CSR 23-2002, Swansea University, Computer Science Report Series (available from http://www-compsci.swan.ac.uk/reports/2002.html) (2002).

[11] Kullmann, O., *The OKlibrary: A generative research platform for (generalised) SAT solving*, Technical Report CSR 1-2008, Swansea University, Computer Science Report Series (http://www-compsci.swan.ac.uk/reports/2008.html) (2008).

[12] Kullmann, O., *Present and future of practical SAT solving*, in: N. Creignou, P. Kolaitis and H. Vollmer, editors, *Complexity of Constraints*, Lecture Notes in Computer Science (LNCS) **5250**, Springer, 2008 pp. 283–319.

[13] Morley, M., *Safety in Railway Signalling Data: A Behavioural Analysis*, LECTURE NOTES IN COMPUTER SCIENCE (1994), pp. 465–465.

[14] Morley, M., *Safety-level communication in railway interlockings*, Science of Computer Programming **29** (1997), pp. 147–170.

[15] Stälmarck, G. and M. Saflund, *Modeling and verifying systems and software in propositional logic*, Safety of Computer Control Systems (SAFECOMP90) (1990), pp. 31–36.