Automated Verification of Safety Properties in Railway Interlocking Systems Defined with Ladder Logic[1]

Karim Kanso Faron Moller Anton Setzer

Dept. of Computer Science, Swansea University, Swansea SA2 8PP, UK
email: `(cskarim, F.G.Moller, A.G.Setzer) @ swansea.ac.uk`

Abstract

In this project the verification of safety conditions for the control of a railway interlocking system written in ladder logic is carried out. All translation steps have been implemented and tested for 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 SAT solver. 4. The generation of specific safety conditions from more generic ones using a topological model of a railway yard.

Keywords: Ladder logic, railway interlocking systems, SAT solvers, verification, automated theorem proving.

1. Introduction

In this project we have written a program which allows the fully-automated verification of railway interlocking systems using SAT solver technology. This software has been applied to a small railway yard.

2. Translation of ladder logic into propositional logic

The railway control system given to us was written in ladder logic, which seems to be the assembly language level for such control systems. Ladder logic is a graphical representation of a sequence of Boolean assignments. After carrying out this translation we obtain a sequence of Boolean valued assignments of the form

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

where φ_i are propositional formulae with variables taken from the set of input, output and intermediate variables (latches).

This ladder is then executed by a program of the form

[1] This research 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 Invensys Railway Systems, Chippenham.

Initialise;

while(true){ output(); input(); $x_1 := \varphi_1$; $\cdots$ $x_n := \varphi_n$; }

In the initialisation phase some variables are set to initial values, others remain undefined. Then a continuous while loop is entered in which: 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 then the ladder is executed. Note that, while executing the assignments, the real world output variables are not modified, therefore correctness is only required at the end of each execution of the ladder. (The system needs not to be safe directly after initialisation – correctness after at least one execution of the ladder suffices, and even this condition can be relaxed.)

To prove the correctness of a safety condition ψ, we need to show that ψ holds after executing the ladder n times for every $n \geq 1$. 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 the execution of the ladder, it holds afterwards as well.

More formally, we define a formula $\varphi_{\mathcal{L}}$ which models the execution of the ladder; assuming for simplicity that the x_i are all different, this has the form

$$\varphi_{\mathcal{L}} = (x'_1 \leftrightarrow \varphi'_1) \wedge \cdots \wedge (x'_n \leftrightarrow \varphi'_n)$$

Here x'_i are new variables representing the variables after execution; and φ'_i is the result of replacing $x_1, \ldots, x_{i-1}$ by $x'_1, \ldots, x'_{i-1}$. Let φ_{I} be the formula expressing the assignments in the initialisation phase. The conditions to be verified are:

$$(\varphi_{\mathrm{I}} \wedge \varphi_{\mathcal{L}}) \to \psi' \qquad \text{and} \qquad (\psi \wedge \varphi_{\mathcal{L}}) \to \psi'$$

where ψ' is the result of replacing x_i by x'_i, thus expressing that the safety condition holds with the variables as they have been adapted after the execution of the ladder. For example, if the initialisation sets variable a to true, the safety condition is $b{=}a$, and the ladder has one rung representing $a := b$, we obtain the formulae

$$((a \leftrightarrow \text{true}) \wedge a' \leftrightarrow b) \to b \leftrightarrow a' \quad \text{and} \quad ((b \leftrightarrow a) \wedge a' \leftrightarrow b) \to b \leftrightarrow a'$$

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 formulae above.

3. Invariants

When verifying the railway interlocking system, false positives were found. There were two reasons for these:

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 least one of A, B, C is chosen, but if the button is malfunctioning (e.g. staples falling into the button) it might be that both A and B or B and C are chosen.

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 the green signal is activated the red one is not activated). When such a possible invariant

ψ_{Inv} was discovered (e.g. $\mathrm{signal_i\_is\_red} \leftrightarrow \neg\mathrm{signal_i\_is\_green}$) we first tried to prove that it is a true invariant, i.e. that it always holds:

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

If it was provable, we could assume that this invariant holds before executing the ladder, thus relaxing the induction statement to:

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

It seems to be a major challenge to identify invariants automatically.

4. Generating specific safety conditions from signalling principles

In order to make it easier to write down safety conditions, we formulated them first in first order logic using general predicates. An example would be *"points in a rail yard should not be set to the normal and reverse positions simultaneously"*: $\forall pt \in Points\colon \neg[normal(pt) \wedge reverse(pt)]$ (*normal* and *reverse* are the two possible positions of points). We used Prolog terms to construct a topological model of the rail yard. Variables in the general safety principles range over finite domains, thus universal quantification can be replaced by a finite conjunction, and existential quantification by a finite disjunction. In this way we obtain propositional formulae referring to the propositional variables used in the ladder, which can then be verified using the machinery introduced above. In order to identify more precisely the reason for a possible counter example, the safety conditions – which often formed a large conjunction – were split into its conjuncts which form more specific safety conditions.

5. Conclusion

Our approach was applied to a model provided by our industrial sponsor of a railway yard with 331 rungs 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 many safety conditions. We found some counter examples, which were however already known to the company but recognised not to be safety critical, being intermittent and occurring for only one cycle of the ladder. This project therefore 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 – which is actually executed – thus avoiding compiler errors.