

SAT-based Model Checking of Train Control Systems

Phillip James*, Markus Roggenbach**
cspj@swansea.ac.uk, csmarkus@swansea.ac.uk

Swansea University, United Kingdom

Formal verification of railway control software has been identified to be one of the “grand challenges” [7] of Computer Science. Various formal methods have been applied to this area, including algebraic specification, e.g. [3], process algebraic modelling and verification, e.g. [14], and also model oriented specification, where e.g. the B method has been used in order to verify part of the Paris Metro railway [4]. In partnership with Invensys, an internationally established company specialised in railway control systems, we explore various verification approaches based on SAT solving [2].

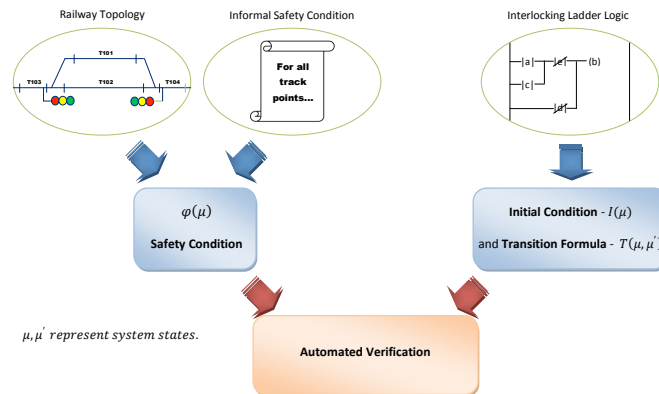


Fig. 1. The basic verification setting

Continuing previous work by Kanso et al. [9] we verify interlockings of real world train stations with respect to safety conditions. Our modelling language is propositional logic, see Fig. 1: The physical layout of the train station together with an abstract safety condition, e.g. ‘trains are

* Acknowledging the support of Westinghouse Rail Systems, Chippenham, UK.

** Acknowledging the support of EPSRC under the grant EP/D037212/1.

separated by at least one empty track segment’, yields a concrete safety condition $\varphi(\mu)$. The initial configuration of a train station is characterised by some initialisation formula $I(\mu)$. The control program (in ladder logic, an ISO standard [1]) of the interlocking system is translated into a transition formula $T(\mu, \mu')$. Here, μ and μ' represent states of the interlocking. All the above translations have been automated in [9]. Using an inductive approach, namely $I(\mu) \Rightarrow \varphi(\mu)$ and $\varphi(\mu) \wedge T(\mu, \mu') \Rightarrow \varphi(\mu')$, Kanso et al [9] successfully verify a medium sized real world interlocking. Some of the required safety properties are fully automatically proven using a SAT solver [10]. However, in some cases the SAT solver produces counter examples. In the context of the interlocking under discussion, manual analysis can exclude these counter examples as they concern unreachable states. For inclusion into the standard development process of interlockings, the company Invensys requires further automation of the verification, namely the exclusion of unreachable states and the production of error traces in the case that a safety property does not hold.

In order to accommodate these requirements, we develop and experiment with verification approaches based on ideas used in bounded model checking. Here, we deliberately stay within boolean modelling: first, it is natural in the given context – the ladder logic program speaks on boolean variables only; second, it allows the direct use of SAT solvers for verification. Concretely, we work with two scenarios:

Forward Iteration: Explore systematically the states B_i reachable in i steps. Should B_i include a violating state, return the result “unsafe” together with an error trace. Should B_{i+1} contain only already visited states, all reachable states have been covered and the system is safe. As the state space is finite, the algorithm below terminates.

```

i ← 0
B0 ← { $\mu \mid I(\mu)$ }
while  $B_i \not\subseteq B_0 \cup \dots \cup B_{i-1}$  do
  for  $\mu \in B_i$ , if  $\neg(\varphi(\mu)) \in SAT$  return “unsafe” and an error-trace; stop
   $B_{i+1} \leftarrow \{\mu' \mid T(\mu, \mu'), \mu \in B_i\}$ 
  i ← i + 1
return “safe”

```

Backward Iteration First, verify using Kanso’s approach. If the SAT solver finds a state violating the safety property, iterate the transition formula backwards from this violating state. If this leads to an initial state, produce an error trace; otherwise the system is safe when a fixed point of states is reached.

Using the CASL institution independent structuring mechanisms [12] in order to specify forward and backward iteration, and with the tools Hets

[11], Paradox [5], and Minisat [6], we successfully and fully automatically debug and verify simple examples of control programs for a Pelican Crossing: in the case of an error, both scenarios effectively produce error traces; if all reachable states are safe, forward and backward iteration effectively and fully automatically verifies the control program.

Concerning real world interlockings, we expect both of our approaches to scale up. Here, we pre-process the transition formula: we remove trivial functional dependencies [8] and apply slicing techniques [13] w.r.t. the safety property under discussion.

References

1. Programmable Controllers - Part 3: Programming languages, 2003. IEC Standard 61131-3.
2. A. Biere, M. J. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.
3. D. Bjoerner. Towards a domain model of transportation. In *Domain Engineering – Technology Management, Research and Engineering*, pages 333–358. JAIST Press, To appear.
4. J. Boulanger and M. Gallardo. Validation and verification of meteor safety software. In *Advances in Transport Vol 7*, pages 189–200. WIT Press, 2000.
5. K. Claessen and N. Sörensson. New techniques that improve MACE-style model finding. In *MODEL’03*, 2003.
6. N. Een and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT 2003*, LNCS 2919. Springer, 2004.
7. R. Jacquart, editor. *IFIP 18th World Computer Congress, Topical Sessions*, chapter TRain: The Railway Domain - A Grand Challenge. Kluwer, 2004.
8. J.-H. R. Jiang and R. K. Brayton. Functional dependency for verification reduction. In R. Alur and D. A. Peled, editors, *Computer Aided Verification*, LNCS 2919. Springer, 2004.
9. K. Kanso, F. Moller, and A. Setzer. Verification of safety properties in railway interlocking systems defined with ladder logic. In M. Calder and A. Miller, editors, *AVOCS08*, Glasgow 2008.
10. O. Kullmann. OK-Library – a library for SAT solving. <http://www.ok-sat-library.org/>.
11. T. Mossakowski, C. Maeder, and K. Lüttich. The heterogeneous tool set. In O. Grumberg and M. Huth, editors, *TACAS’07*, LNCS 4424. Springer, 2007.
12. P. D. Mosses, editor. *CASL Reference Manual*. LNCS 2960. Springer, 2004.
13. F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3), 1995.
14. K. Winter. Model checking railway interlocking systems. *Australian Computer Science Communications*, 24(1), 2002.