A Journey through Software Model Checking of Interlocking Programs

Simon Chadwick[1], Phillip James[2], Faron Moller[2], Markus Roggenbach[2], and Thomas Werner[1]

[1] Siemens Mobility, Chippenham, U.K.
{simon.chadwick,tom.werner}@siemens.com
[2] Swansea University, U.K.
{p.d.james, f.g.moller, m.roggenbach}@swansea.ac.uk

In this paper, we report and reflect upon successful technology transfer from Swansea University to Siemens Mobility over the years 2007-2021. This transfer concerns formal software verification technology for programming interlocking computers from Technology Readiness Level TRL 1-7.

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*. 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 rules 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 rules (we work with about 300 rules) that shall guarantee safe rail operation. This poses the question of how one can verify that a given program written in ladder logic fulfils a safety property.

Theoretical Foundations (TRL 1 & 2). Software model checking verification of interlockings is well established within the railway domain. Already 25 years ago, Groote et al. [3] used it to verify the interlocking of Hoorn Kersenbooger station. In this approach, ladder logic programs are represented in temporal propositional logic. Generic safety properties are formulated in temporal first order logic; for concrete track layouts these generic properties can equivalently be expressed in the same temporal propositional logic as used for representing the programs. This allows interlocking verification to be formulate as software model checking for temporal propositional logic [2, 5].

Academic Experiments (TRL 3 & 4). Software model checking for ladder logic programs requires the automation of two steps. First, one needs to transform a program into a logical formula Ψ via the Tseitin transformation. Then, one needs to instantiate a generic safety property with track plan information and derive a formula ϕ in temporal propositional logic. One can then verify that ϕ is a consequence of Ψ using a standard SAT solver.

Our experiences [4, 6], using a mix of programming languages (Haskell, Java, Prolog) and experimenting with two ladder logic programs and c. 5 safety properties, suggest the following: the verification concept works and scales up to real

world programs; applying program slicing is worthwhile in order to reduce verification times; rather than looking purely for *hold / does not hold* decisions, bounded model checking is useful for debugging; visualisation of counter examples turns out to be a challenge; and verification via k-induction fails.

Technology Transfer (TRL 5-7). Further development required deeper collaboration between academic and industrial partners, in order to interpret verification failures and to expand the number of examples treated concerning the encoding of safety properties [1, 7]. The software architecture needed revision to cater for tool interoperability, usability and error treatment. The resulting tool is able to find mistakes in ladder logic programs that cannot be found with traditional testing methods. Also, it has a turn-around time in the order of hours, as compared to turn-around times in the order of a week for testing.

Reflections. A number of themes permeate the described technology transfer, including faithful modelling, scalability, accessibility, and interoperability. Depending on the TRL, these themes recurred with a different focus. For instance, when considering the theoretical foundations, faithful modelling concerned the definitions of the logic and the transformations. When it came to the academic experiments, it meant correctness of slicing, and true representation of selected properties and track layouts. Finally, in technology transfer, we had to reflect how "complete" the set of safety properties was.

Future development (Towards TRL 8 & 9). Until now, we have only analysed artefacts from completed projects. However, we have concrete plans for a trial under real software production conditions. For this, we need to revisit the theoretical foundations. For example, rather than a one-step next operator, the logic needs to provide k-step next operators (for $k>1$), to cater for *fleeting outputs* of ladder logic programs: outputs which are "unstable" for a limited number of cycles. This will allow to "relativise" safety properties up to fleeting. Also, experiments with model checking algorithms are needed to mitigate the effect of *false positives*: the reporting of an error where there is none.

References

1. S. Chadwick. Formal verification, 2020. Invited Talk at BCTCS 2020, https://cs.swansea.ac.uk/bctcs2020/Slides/Chadwick.pdf.
2. S. Gruner, A. Kumar, T. Maibaum, and M. Roggenbach. *On the Construction of Engineering Handbooks*. Springer, 2020.
3. J. K. J. Groote, S. van Vlijmen. The safety-guaranteeing system at station hoornkersenboogerd. In *COMPASS'95*. IEEE, 1995.
4. P. James. SAT-based model checking and its applications to train control software, 2010. Master of Research, Swansea University.
5. P. James et al. Verification of solid state interlocking programs. In *SEFM 2013 Collocated Workshops*, LNCS 8368. Springer, 2013.
6. K. Kanso. Formal verification of ladder logic, 2010. Master of Research, Swansea University.
7. T. Werner. Safety verification of ladder logic programs for railway interlockings, 2017. Master of Research, Swansea University.