Electronic Communications of the EASST
No \volume defined!

No \volumetitle defined!

Verification of Railway Interlockings in Scade

Andrew Lawrence and Monika Seisenberger

3 pages

No *ed(s) defined!
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Verification of Railway Interlockings in Scade

Andrew Lawrence* **and Monika Seisenberger**

Swansea University, Wales

Abstract: We present two modelling approaches for the application of model checking to verify railway interlockings. The first translates so-called Ladder Logic into Scade language, the second models a segment of railway from scratch. The verification tool used is Scade.

Keywords: Verification, Model Checking, Scade, Data-flow, Lustre, Railway

The aim of our research is to investigate the use of Scade Suite (Esterel Technologies) for the verification of railway interlockings. This is a feasibility study done in co-operation with Invensys Rail, a leading international company for the design, construction, and validation of railway control systems. We concentrate on the application of modelling and model checking [CGP99]; specifically we present the development of two different modelling approaches. In the first approach, we translate existing specifications written in so-called Ladder Logic into Scade and verify them. In the second, we model a segment of railway from scratch. We applied the first approach to two real world railway interlockings, however, for confidentially reasons, we can only demonstrate the method with a toy example. The track plan and control table of one of these two real world interlockings were simplified and led to the model considered in the second approach.

Scade Suite is a tool designed for the development of safety-critical software. Underlying Scade [Sca, ADS+06] is a formal language referred to as the Scade language which is a dialect of the synchronous data-flow language Lustre [CPHP87]. Models written in this language can have a variety of formal methods applied in order to verify their correctness (e.g. BDDs, Stallmarck's algorithm [SS00]).

Railway Engineers typically use Ladder Logic to describe/program the behaviour of their control systems. Building on previous projects in Swansea [JR09, KMS09], we created a tool (written in Haskell) that translates Ladder Logic programs into Scade language.

As an example we formally model a pelican crossing (traffic lights allowing pedestrians to cross a street upon request). The boolean variable *pressed* indicates whether or not a button is pressed by a pedestrian. *req* and *crossing* represent two internal variables indicating whether a crossing is required or in use. The fragment of Ladder Logic shown in Figure 1 corresponds to the following propositional logic formula

$$crossing' \leftrightarrow \neg crossing \wedge req$$
$$req' \leftrightarrow \neg req \wedge pressed$$

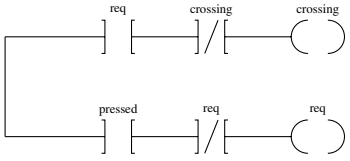

Figure 1: Ladder Logic Fragment

where the new variables $crossing'$ and req' refer to the next state. Informally, this fragment of Ladder Logic can be read as: If the crossing is not in use currently and a crossing is required,

* Acknowledging the support of Invensys Rail

then the crossing will become active in the next step. If the button is pressed and the crossing is not required, then the crossing will become required in the next step.

The Scade language contains temporal operators that enable us to express properties of flows with respect to time. The operator `pre` allows us to access the previous value of a variable and the operator `->` allows us to express that a flow has a specific value initially. For instance, `A = false -> (not(pre A))` models an alternating stream of booleans which is initially set to false.

After this translation, the Ladder Logic fragment becomes the following in Scade.

```
crossing = false -> pre req and (not (pre crossing));
req = false -> (not pre req) and pressed;
```

Similarly, we modelled the traffic lights for pedestrians and cars and proved appropriate safety conditions such as lights for pedestrians and cars never being green at the same time.

This approach was scaled up and applied to two real railway interlockings. To give some impression of the sizes involved: the interlockings have approximately 350 rungs (formulas) and 600 variables; the model checking for the safety conditions considered takes less than a second.

In our second approach we use Scade to concretely model a segment of railway (cf [CMSW99] for such a specification in Lustre). We suggest an alternative approach by modelling components such as tracks, points, lights and routes separately. Figure 2 schematically shows the main part of the railway in question which consists of a station with two incoming routes and two outgoing routes. The full specification comprises 11 segments of track, 4 points, 6 routes, 9 lights and a route controller. Most of these components are modelled by finite state machines. Safety conditions were proven using the model as a whole and on the individual components.

The advantages of this approach are obvious. (1) If components are specified individually, they can be reused later. (2) By including the topology in our model we can formalise and prove additional safety conditions that could not be proven using the Ladder Logic approach.

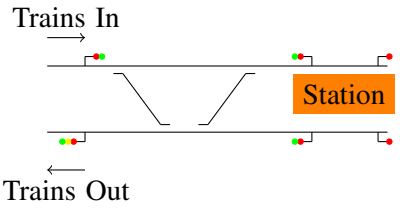

Figure 2: Track Plan

Our research demonstrates that both approaches are applicable to verification problems in the railway domain. In future work, we plan to refine the modelling to exclude false negatives, which occurred in the first approach, as well as to model the full interlocking in the second approach. In the long term, we want to apply a combination of model checking and first order theorem proving, including program extraction.

Acknowledgements: We wish to thank Invensys Rail and the Swansea Verification Group for excellent cooperation and support.

Bibliography

[ADS+06] P. Abdulla, J. Deneux, G. Stålmarck, H. Argen, and O. Akerlund. Designing Safe, Reliable Systems Using SCADE. *Lecture Notes in Computer Science*, 4313:115–129, 2006.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[CMSW99] P. Caspi, C. Mazuet, R. Salem, and D. Weber. Formal design of distributed control systems with Lustre. In *Proc. Safecomp99*, pages 396–409. Springer-Verlag, 1999.

[CPHP87] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: a declarative language for real-time programming. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 178–188, New York, NY, USA, 1987. ACM.

[JR09] P. James and M. Roggenbach. SAT-based Model Checking of Train Control Systems. In *CALCO Young Researchers Workshop CALCO-jnr 2009, Selected Papers, pp. 73–88*, 2009.

[KMS09] K. Kanso, F. Moller, and A. Setzer. Automated Verification of Signalling Principles in Railway Interlocking Systems. *Electronic Notes in Theoretical Computer Science*, 250(2):19–31, 2009.

[Sca] Scade, http://www.esterel-technologies.com.

[SS00] M. Sheeran and G. Stålmarck. A Tutorial on Stålmarck's Proof Procedure for Propositional Logic. *Form. Methods Syst. Des.*, 16(1):23–58, 2000.