Specifying Railway Interlocking Requirements for Practical Use

Lars-Henrik Eriksson

Logikkonsult NP AB
Swedenborgsgatan 2
S-118 48 STOCKHOLM
SWEDEN

E-mail: lhe@lk.se

Abstract

An essentially complete formal specification of safety requirements for railway interlockings has been developed. The work is part of as project with the Swedish National Rail Administration investigating the feasibility of using formal methods for the analysis of interlockings in a production setting. An overview of the specification is given and two ongoing case studies on verifying interlockings using the specification are described. Verification is done using the very fast Stålmarck theorem prover for propositional logic. The current limits of the technology is discussed.

1. Introduction

The Swedish National Rail Administration (Banverket) has for some years been using formal methods as one of the techniques to verify new software releases for the Ebilock 850 and 950 interlockings made by ABB-Daimler Benz Transportation (Adtranz) Signal AB in Sweden. Formal methods are also used by the manufacturer itself during software development.

In both cases, the source code of the interlocking logic software is analysed using a software tool – CVT [1] – specifically developed by Logikkonsult NP AB for this purpose. The tool accepts programs written in Adtranz Signal's proprietary programming language STERNO^{2}L and can carry out various kind of analyses.

So far, the use of formal methods has been limited to specific low-level properties of the software - in particular that a certain kind of execution error can never occur. Banverket are also using formal methods to compare different versions of software modules to pinpoint functional changes of the software.

Banverket is currently investigating the possibility of full-scale use of formal methods to communicate safety requirements of interlockings and to verify the correct realisation of these requirements. To this end, Logikkonsult NP AB has

developed a set of formal specifications for safety properties of interlockings under a contract with Banverket. In this paper we will briefly describe that specification and its use. A complete presentation of the specification is given in references [2,3] (in Swedish). The contract also calls for the formal verification of an actual interlocking using the specification. As of this writing, that work is still in progress but some preliminary results can be presented.

2. The specification

In contrast with previous work on specification and/or verification of interlockings [4,5,6,7], this work has from the beginning been intended to describe a set of requirements which is at the same time general and complete. The scope of the specification is essentially limited only by the fact that Banverket's present interest is concentrated on small rail yards[1]. Features relevant mainly to larger yards have been excluded from the specification. To facilitate the acceptance and practical use of the specification, it has been written using concepts traditionally used in Swedish signalling practise.

The specification describes *functional* safety requirements only. Safety requirements relating to the construction of the interlocking, such that certain failure modes must not lead to dangerous situations, are not included. Also, safety requirements about the internal workings of track side objects are not included. The track side objects are regarded as black boxes which the interlocking requires to be in certain states, and which report back to the interlocking whether those states have properly been assumed or not.

The specification has been written using an extension to first-order predicate logic known as "Delphi logic". Formal specification using this logic is supported by the Delphi tool[2] [8,9], a prototype specification tool developed jointly by Logikkonsult NP AB and Ellemtel Telecommunication System Laboratories in Sweden. In the case of the present specification, the important extension involves a way to describe state changes, so that the behaviour of a system under sequences of external events can be described, even though the logic is not temporal.

A Delphi specification consists of two parts: the *conceptual model* and the *rule set*. The conceptual model describes the concepts that are used in the specification, what properties they have and how they are related. The rule set describes (the requirements on) how the system should react to events in the environment.

The conceptual model of the present specification describes the different physical objects and abstract concepts used by interlockings, such as rail yards, signals,

[1] This is because Banverket is presently in the process of introducing a new generation of interlockings intended for small rail yards.

[2] Not to be confused with the commercial program development tool with the same name.

points, train routes[3] etc. The safety properties are described as a set of invariants (axioms) that express relations between the objects that must always hold in order to maintain a safe situation in the yard. Apart from safety requirements, the specification includes requirements describing the possible layouts of a rail yard. This is needed both when using the specification to reason about arbitrary yards, and to verify that a description of a particular rail yard makes sense.

The specification describes general safety requirements. In order to use the specification to describe requirements for a particular interlocking – e.g. for verification – the specification must be supplemented with a description of the layout and properties of the particular rail yard controlled by the interlocking. This description is given as a set of atomic facts in predicate logic.

The predicates of the conceptual model represent the different states of the interlocking, and track side objects, as well as relations between both. For each type of track side object – such as a point – there is a set of predicates used to express the intended states of the object, such as the possible positions of the point.

The invariants of the specification express the safety requirements in terms of the predicates. E.g. there is a requirement that if a point is occupied (by an engine or car), the point must be locked (in order to prevent derailing after accidentally manoeuvring the point while there is a train moving across it). This requirement can be expressed simply as:

```
ALL pt (occupied(pt) -> point_locked(pt))
```

...where `pt` is a variable that ranges over points. Given predicates to express the locking of train routes and to relate names of train routes to the parts of the rail yard that makes up the route, the requirement that all points in a locked train route must also be locked can be expressed as:

```
ALL pt (SOME tr (locked(tr) AND part_of(tr,pt)) ->
                 point_locked(pt))
```

The rule set of the specification describes how the state of the interlocking (and of track side objects controlled by the interlocking) may change as a result of events from the environment. As an example, the train dispatcher (or traffic management software) may request that a point be moved to a certain position. The interlocking should accept that request only if the point is not locked. This requirement is described using a Delphi rule:

```
WHEN move_right(pt) IS DETECTED
IF NOT point_locked(pt) CONCLUDE right(pt) AND NOT left(pt)
```

[3] A *train route* is a part of the rail yard intended for the movement of a train. When a train is about to use a particular route, all points in the route must be locked in the correct positions, the route must be protected from cars accidentally rolling into it from the side, the route must be free from obstacles etc.

When a request to put a point into the right position (`move_right`) is received by the interlocking, the rule requires that the *precondition* `NOT point_locked(pt)` is satisfied for the state change to be permitted. If that is the case, the state of the interlocking should be changed so that the point is moved to the right position and is no longer in the left position.

Since points are physical objects, the interlocking can not simply assume that the point is in the correct position as described by the predicates `right` and `left`. A separate predicate, `controlled`, represents information about whether the point has actually moved to the intended position.

The specification has been validated in several ways. It has been used to simulate the behaviour of interlockings, and the behaviour has been checked for safety. Several safety properties not directly expressed by the specification has been formally proved to follow from it. E.g. that two signals leading into the same track from different directions can not both show a drive aspect (green light) simultaneously.

Banverket has also inspected and approved a plain text translation of the specification. We consider approval of the specification by domain experts to be crucial for acceptance of formal methods in an application area. If the domain experts do not have sufficient formal methods training to fully understand the formal specification themselves, a plain text version adhering as closely as possible to the structure and concept set of the formal specification should be used for assessment.

3. Case studies

Presently, case studies about using the specification to formally prove the correctness of existing interlockings are in progress. Although the studies have not been completed, preliminary results have been obtained.

Two verification activities are taking place. Under a contract with Banverket, Logikkonsult is verifying a particular instance of a widely used relay interlocking system (SJ model 59). Also, under the supervision of Logikkonsult's staff, a Ph.D. student funded by the Danish State Railway is doing verification work on the Ebilock 850/950 interlockings mentioned in the introduction.

The interlockings are verified by translating the relay circuits and program code, respectively, into propositional logic and then showing that the various states that can be assumed by the interlocking are all among the states permitted by the specification.

The translation into logic presents no particular difficulties in general. Relay circuits can naturally be regarded as networks of logical gates which are trivially translated into logic formulæ. The few cases where the circuitry does not translate readily into

propositional logic (timing circuits in particular) are handled as special cases outside the logic with little trouble.

The program code of the Ebilock 850/950 interlockings are divided into a number of modules, each of which are translated into a logical formula describing the input-output behaviour of the module. To accommodate all features of the STERNO^{2}L language, the propositional logic is extended with a limited form of arithmetic.

Since the specification has been written without any particular interlocking system in mind, it should be expected that the same concepts are represented quite differently in the specification and in a realisation. E.g., the specification represents a train route as the set of the track sections making up the route, while the relay interlocking represents a train route using a combination of two relays – one for the beginning and one for the end of the route.

To enable analysis of an interlocking, the different representations of the concepts must be related using formulæ in logic. This can be problematic, as it is not necessarily obvious how concepts are represented in the actual interlocking system. In the case of the Ebilock 850/950 interlockings in particular, this has proven to be a major difficulty, due to incomplete documentation.

The actual verification of requirements is essentially done by proving that the requirement formulæ of the specification are logical consequences of the translation into logic of the interlocking system. Any erroneous state will cause the proof process to fail and gives rise to a counter example, showing a situation where the interlocking fails to fulfil the requirements.

The logical proofs are done using the Stålmarck theorem prover for propositional logic [1,10,11,12]. The specification is written in predicate logic, but as any particular rail yard has a fixed number of objects, the specification supplemented with the rail yard description can be translated into propositional logic by expanding the quantifiers. To accommodate translations of STERNO^{2}L programs the theorem prover provides limited support for arithmetic[4].

The size of the formulæ that need to be proved can be quite large. The relay interlocking under study controls a small rail yard with only two points and 12 signals. The translation of the specification into propositional logic for this rail yard results in a formula with about 10 000 logical operators. The translation of the interlocking circuitry itself results in a formula with about 1000 logical operators. This is well within the capability of the theorem prover in most cases.

So far in the project, only a few requirements have actually been verified although we expect a full verification to present no difficulties. An example of a verified requirement is that a signal cannot show a drive aspect (green light) when there are obstacles in the route behind the signal.

[4] In particular, the theorem prover is not complete when arithmetic is used.

The particular Ebilock system under study is intended for a larger rail yard with 22 points and 33 signals. A translation of the program for this interlocking results in a formula with about 400 000 logical operators. This touches the current limits of the theorem prover used. For all but the most simple properties an analysis cannot be made in reasonable time. The simplest requirements could be analysed in about 5 minutes on a SUN 4/20 computer, while the analysis of more complex requirements needed a day or more of CPU time.

On the other hand, since the Ebilock 850/950 interlocking uses general program modules that are combined according to the layout of a particular rail yard to form a program for a interlocking intended for that yard, it is possible to verify the program modules separately, a few modules in combination, or even entire parts of the rail yard. In this way, it has been possible to analyse many requirements in only a few minutes.

Unfortunately, the analysis of the program code has so far not led to any conclusive results about the correctness or incorrectness of the program. For most of the requirements tried, counter examples have been found. However, it is very difficult to tell if the situations corresponding to the counter examples can actually occur. The reason is that correct operation of each program module depends crucially on it being passed correct data from other program modules during execution. When less than an entire rail yard is analysed, it is necessary to be able to characterise formally what "correct data" from the excluded modules mean, in order to avoid false counter examples. This has proven to be difficult since sufficiently detailed documentation of the software has not been available.

This example shows that in order to formally verify large systems in practise, the systems have to be designed with formal verification in mind so that they can be decomposed cleanly into parts that can be analysed separately.

Although the two case studies concern rail yards of very different sizes, the difference in problem complexity can not only be explained by this. Another important factor is the complicated design of the interlocking program and the fact that the interlocking program is very general – intended to cope with all possible rail yard configurations – while the relay interlocking is tailored to the specific yard. While this brings the advantage that the same software can be used for different yards, it does mean that attempts to verify interlockings for simple yards will still be burdened with having to deal with program code that is unnecessary in that particular instance.

References

1. Stålmarck G, Säflund M: Modelling and Verifying Systems and Software in Propositional Logic. In: Daniels BK (ed) Safety of Computer Control Systems 1990 (SAFECOMP'90). Pergamon Press, Oxford, 1990.

2. Eriksson L-H. Formalisering av krav på ställverk (delrapport fas 1). Report NP-K-LHE-001. Logikkonsult NP AB, Stockholm, 1995. (in Swedish)

3. Eriksson L-H. Formalisering av krav på ställverk (slutrapport). Report NP-K-LHE-003. Logikkonsult NP AB, Stockholm, 1996. (in Swedish)

4. Groote JF et.al. The Safety Guaranteeing System at Station Horn-Kersenboogerd. Logic Group Preprint Series No. 121. Department of Philosophy, Utrecht University, Utrecht, 1994.

5. Hansen KM: Validation of a Railway Interlocking Model. In: Naftalin, Denvir, Bertran (eds.) FME'94: Industrial Benefit of Formal Methods. Springer-Verlag, Heidelberg, 1994. (Lecture Notes in Computer Science no. 873)

6. Morley MJ. Modelling British Rail's Interlocking Logic: Geographical Data Correctness. Technical Report ECS-LFCS-91-186. Department of Computer Science, University of Edinburgh, Edinburgh, 1991.

7. Morley MJ: Safety In Railway Signalling Data: A Behavioural Analysis. In: Joyce, Seger (eds.) Higher Order Logic Theorem Proving and its Applications. Springer-Verlag, Heidelberg, 1993. (Lecture Notes in Computer Science)

8. Höök H. Delphi – A General Description of the Language. Report F 91 0881. Ellemtel Utvecklings AB, Stockholm, 1993.

9. Stålmarck G, Widebäck F. Definition av Delphi. Report NP-FW-001. Logikkonsult NP AB, Stockholm, 1991. (in Swedish).

10. Stålmarck G, Åkerlund O: Formal verification of hardware and software systems using NP-Circuit. In: Malmén Y, Rouhiainen V (eds.) Reliability and safety of processes and manufacturing systems. Elsevier, London, 1991.

11. Säflund M: Modelling and formally verifying systems and software in industrial applications. In: Proc. of the Second International Conference on Reliability, Maintainability and Safety (ICRMS'94). International Academic Publishers, Beijing, 1994.

12. Widebäck F. Stålmarck's Notion of *n*-saturation. Report NP-K-FW-200. Logikkonsult NP AB, Stockholm, 1996.