Modelling and Analysing a PLC-Based Railway Traffic Control System *

Kimmo Varpaaniemi
Helsinki University of Technology
Laboratory for Theoretical Computer Science (HUT-TCS)
P.O. Box 9205, FIN-02015 HUT, Espoo, Finland
kimmo.varpaaniemi@hut.fi

Abstract

This paper reports a case study where a distributed PLC-based railway traffic control system was modelled and analysed.
Keywords: programmable logic controllers, railway traffic control, formal methods, bounded model checking

1 Introduction

Programmable logic controllers (*PLCs*) are widely used throughout the world, applications covering almost all areas of life. PLCs typically control things that traditionally have been controlled by switching circuits such as relay networks and gate-logic networks. Unlike traditional switching circuits, PLCs are configurable, and this configurability at least partially explains the success of PLCs.

Many PLC systems are distributed, consisting of PLC programs that communicate by sending messages. Since the use of distributed PLC systems in safety-critical tasks keeps increasing, there is an unquestionable need for *verification* techniques and tools. By verification we mean ensuring correctness on one hand and detection of errors on the other hand. A typical approach to verification is *testing*. However, many distributed PLC systems are so complex that even coarse errors may remain undetected, no matter how much effort is put into the design of the tests. In order to extend the coverability of analysis, *formal verification* is needed. In formal verification, a *mathematical model* of a system is constructed, and *mathematically formulated properties* are shown to hold or not to hold in the model.

Reachability analysis is one of the basic techniques used in formal verification. In reachability analysis, a given model of a system is expanded into a *state space*, the *reachability graph* of the model. *Model checking* is a refined form of reachability analysis. In model checking, the property of interest is typically formulated as a *temporal logic formula*, and the goal is then to check whether the formula holds.

The major problem in reachability analysis, widely known as the *state space explosion problem*, is that the reachability graph can be far too large to be fully and concretely constructed. Fortunately, many verification tasks can be reliably carried out without the full concrete construction of the reachability graph. Several approaches have been developed for this purpose, e.g. *symbolic model checking*, *compositional methods*, *partial order methods*, and the use of *symmetries*.

*This work has been funded by The National Technology Agency of Finland, Finnish Rail Administration, Elisa Communications, Nokia Networks, and Nokia Research Center.

HUT-TCS (Helsinki University of Technology, Laboratory for Theoretical Computer Science, former Digital Systems Laboratory) has a long experience in formal verification, especially in reachability analysis. The development of reachability analysis software has been particularly active. The PRENA tool was developed in the 80's and was successfully used e.g. for analysing a railway PLC system [17].

At the end of the 90's, the laboratory started to develop yet another reachability analysis tool, called Maria. The development became a part in a project which had the name Maria too. The Maria project had a few subprojects where case studies were carried out. One of these subprojects was concentrated on modelling and analysis of a PLC-based railway traffic control system. This distributed PLC system had been designed by Mipro Oy and was already in use in the railway section between Haapamäki and Seinäjoki. This paper reports experience obtained in modelling and analysis of that system.

Several tools were tried for the analysis. Careful abstractions and automated slicing were applied, but the model remained huge without containing impressive structural regularities. Nondeterminism was the the worst-of-all source of state space explosion problem. Due to lack of insight on the determination of input signals in the system, the model remained highly nondeterministic w.r.t. the changes in the input signals. Though a combination of overapproximation and abstraction refinement could in principle be used for aggressive circumvention of nondeterminism, the practical problem in the project was that HUT-TCS alone had no expertise to do "reality checks" on its own.

There was also "extreme determinism", i.e. variables interesting for verification were "guarded" up to an extent that random simulation was unlikely to find anything interesting within human time limits. For these reasons, explicit state tools began to seem inappropriate. Instead, this paper emphasizes one form of symbolic model checking, *bounded model checking* [38, 39], and uses a combination of the translator Bc2cnf [41] and the solver Limmat [40] as an example.

Unfortunately, the main result of the experiments is the not very impressive conjecture that too many modelling mistakes have been made, i.e. there have been some false assumptions on the semantics of the system, or some fatal typos have been involved. (HUT-TCS got the original system description only in the form of printed pages, whereas automated scanning of those pages was never seriously considered.)

2 The system which was modelled

The modelled system had been developed by using the ELOP I package of HIMA [35]. HIMA also has the ELOP II package which is somewhat more compatible with the IEC standard [36] than ELOP I. However, ELOP I itself is quite close to the IEC standard as can be concluded e.g. by comparing the description in [34] to the corresponding description in [36]. In discussions with Mipro Oy, it has also turned out that the concept of a function block in ELOP I is essentially the same as in the IEC standard. On page 61 of [36], function blocks are described as follows.

"For the purposes of programmable controller programming languages, a *function block* is a program organization unit which, when executed, yields one or more values. Multiple, named *instances* (copies) of a function block can be created. Each instance shall have an associated identifier (the *instance name*), and a data structure containing its output and internal variables, and, depending on the implementation, values of or references to its input parameters. All the values of the output variables and the necessary internal variables of this data structure shall persist from one execution of the function block to the next; therefore, invocation of the same function block with the same arguments (input parameters) need not always yield the same output values. Only the input and output parameters shall be accessible outside of an instance of a function block, i.e. the function block's internal variables shall be hidden from the user of the function block."

In order to demonstrate the mechanism of how a PLC program calls function blocks, we consider a sample page in a program listing of one of the PLC programs of the Haapamäki – Seinäjoki system. The main motivation of why we consider exactly this page is that a single block is called twice

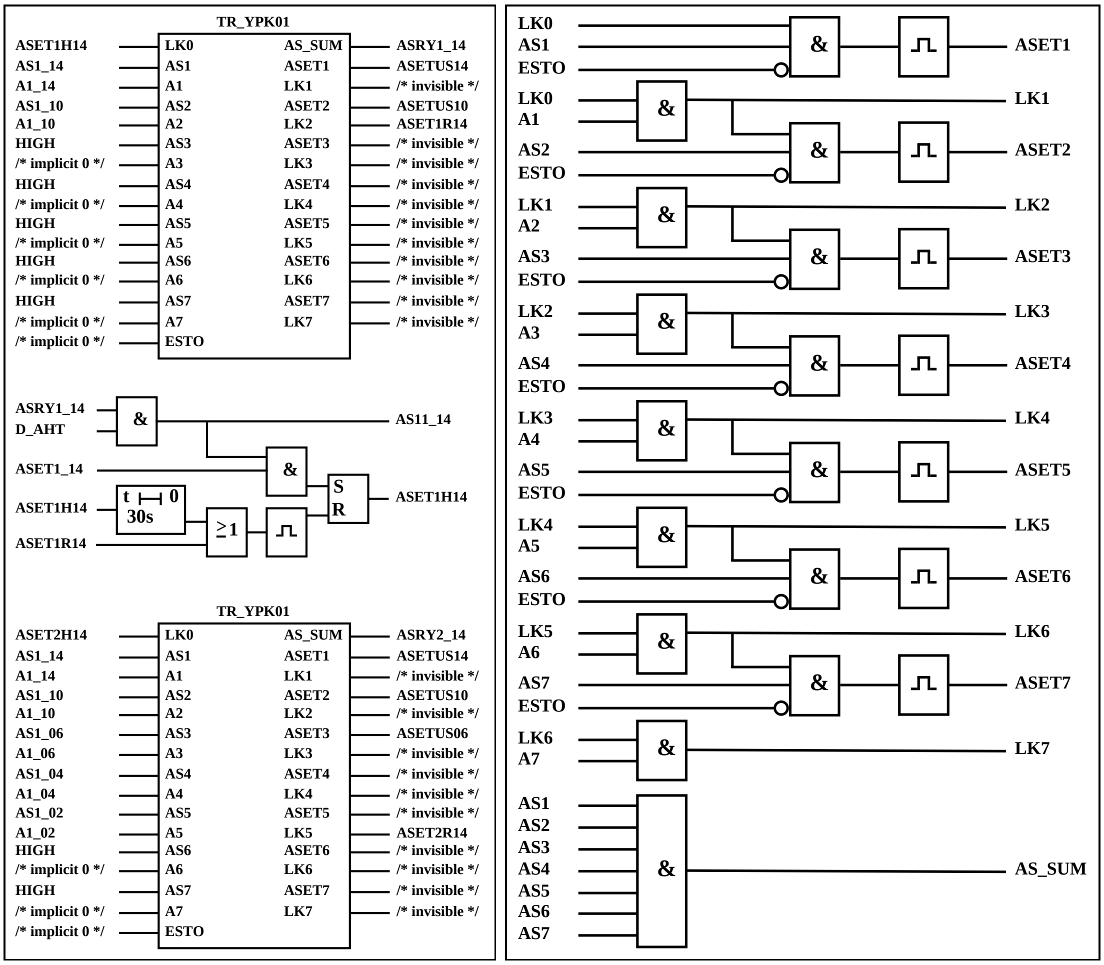

Figure 1: On the left: a sample page in a PLC program listing (essentially a copy of page 16 of Mipro's SK2-KL01.98, version E014). On the right: the logic of the TR_YPK01 block (essentially a copy of page 2 of Mipro's TR-YPK01.10, version 574E).

on the page. The page is related to how the system reacts to certain keyboard commands issued by people at the involved railway stations. The variables occuring on the page indirectly affect the traffic light variables mentioned in Section 4, but so do many other variables on the hundreds of pages that were modelled. Attempts to describe the roles of the variables of the chosen page would not essentially improve understanding of even the program that contains the page. (The project described in the present paper never learnt the semantics of the system up to an extent that would have properly explained the connection between the requirements of the system and the implementation.)

The left-hand side of Figure 1 shows the sample page and contains two instances of the function block which we call TR_YPK01, a description of the block being displayed on the right-hand side of Figure 1. To be precise, the real name of the block is TR-YPK01. For the sake of simplicity, some other transliterations of non-alphanumeric substrings have been made, too. Moreover, the original page uses distinct long names for the formal parameters of the block, whereas we just use the short names that appear in the description of the block itself.

In the middle of the left-hand side of Figure 1, there are two and-gates, one delay-on-timer (the delay being 30 seconds as expressed), one or-gate, one monoflop (which generates an impulse with a pulse length of one cycle of the PLC when its input changes from 0 to 1) and one SR-flip-flop (such that RESET, i.e. the lower input line, is dominant against SET). A PLC program is executed "statement by statement" just like an ordinary program, the only difference being that a "statement" in a PLC program typically has a quite visual layout. In the case of the left-hand side of Figure 1, the upper instance of TR_YPK01 is executed first. (This execution means executing several statements as we shall soon see.) The first statement after that execution is the evaluation of the upper and-gate (including the assignment to AS11_14). Then we have a statement which evaluates the rest of the displayed gates in the same order as they were mentioned in the above sentence. (The assignment to ASET1H14 is included in that statement.) After that, the lower instance of TR_YPK01 is executed.

Let us then look closer at TR_YPK01 and its instances. The right-hand side of Figure 1 shows the description of the block. Each execution of the block consists of 15 statements. This block is simple in the sense that it has no actual internal variables unless the necessary memory bits of the monoflops are counted. The instances on the left-hand side of Figure 1 have three striking labels: HIGH, /* implicit 0 */ and /* invisible */. HIGH is a variable which has the value 1 throughout the life of the PLC program. (To be precise, the variable gets the value 1 in a few microseconds after the booting of the program.) The comment /* implicit 0 */ does not belong to the original syntax but refers to the fact that omitting an actual input parameter has the same effect as having a variable with the value 0 as the parameter. The comment /* invisible */ does not belong to the original syntax either but refers to the fact that (at least for this block), an omitted actual output parameter has the same effect as having an internal variable in the place of the corresponding formal parameter. When modelling the execution of the instances of the block, it is clearly justified to abstract out any statements which have no effect on the history of the "caller".

We have not yet considered how the PLC programs communicate with each other. The protocols for the purpose in the Haapamäki – Seinäjoki system are MODBUS [33, 35, 37] and HIBUS [35], the latter being used in connections where high speed is strongly preferable and the distance is very short.

3 An intermediate modelling language

In order to have a clear connection between analysis tool input and original system description, an intermediate modelling language was designed. The intermediate language is strictly line-oriented. For each type of a gate in the original description, the intermediate language has a corresponding operator. One line typically represents the evaluation of a single gate. There are also operators for presenting the communication between the programs, even though the PLC program listings do not include the actual data transfers. (MODBUS and HIBUS take care of the actual transfers.)

The intermediate language presentation is written into several files in such a way that one file typically corresponds to either a single page in the original description or to a part of such a page. The name of the file identifies (at least) the program, the page and the part of the page. Communication operations are integrated with the original description either by inserting communication lines in "ordinary" files or by creating separate files with appropriately chosen page numbers and subscripts. The names of the files and the order of the lines in the files thus define a flow of control for each of the programs.

Block instances can be handled separately, but this is not optimal w.r.t. the maintenance of the files. Therefore, there is a simple metanotation for describing a block without fixing the instance. Representations of the instances can then be obtained by writing simple filters which produce files for the instances. The metalanguage file for the TR_YPK01 block looks as follows.

```
2? <& lk0* as1* !esto*
aset1* <@ 2?
lk1* <& lk0* a1*
3? <& lk1* as2* !esto*
aset2* <@ 3?
lk2* <& lk1* a2*
4? <& lk2* as3* !esto*
```

```
aset3* <@ 4?
lk3* <& lk2* a3*
5? <& lk3* as4* !esto*
aset4* <@ 5?
lk4* <& lk3* a4*
6? <& lk4* as5* !esto*
aset5* <@ 6?
lk5* <& lk4* a5*
7? <& lk5* as6* !esto*
aset6* <@ 7?
lk6* <& lk5* a6*
8? <& lk6* as7* !esto*
aset7* <@ 8?
lk7* <& lk6* a7*
as_sum* <& as1* as2* as3* as4* as5* as6* as7*
```

The asterisks and the question marks are the only metanotation. The file for the upper instance of TR_YPK01 on the left-hand side of Figure 1 looks as follows. The reduction in the number of lines is due to the fact that many of the formal output parameters are redundant in this case.

```
2 <& aset1h14 as1_14
asetus14 <@ 2
lk1_1000 <& aset1h14 a1_14
3 <& lk1_1000 as1_10
asetus10 <@ 3
aset1r14 <& lk1_1000 a1_10
asry1_14 <& as1_14 as1_10
```

The file for the lower instance of TR_YPK01 on the left-hand side of Figure 1 looks as follows.

```
2 <& aset2h14 as1_14
asetus14 <@ 2
lk1_1010 <& aset2h14 a1_14
3 <& lk1_1010 as1_10
asetus10 <@ 3
lk2_1010 <& lk1_1010 a1_10
4 <& lk2_1010 as1_06
asetus06 <@ 4
lk3_1010 <& lk2_1010 a1_06
lk4_1010 <& lk3_1010 a1_04
aset2r14 <& lk4_1010 a1_02
asry2_14 <& as1_14 as1_10 as1_06 as1_04 as1_02
```

The file representing the behaviour in the middle of the left-hand side of Figure 1 is written "directly by hand" and looks as follows.

```
as11_14 <& asry1_14 d_aht
2 <& as11_14 aset1_14
3 </ 30s aset1h14
4 <| 3 aset1r14
5 <@ 4
aset1h14 <# 2 5
```

In general, a line of the intermediate language is of the form "variable <operator operands". The operands themselves do not include operators, with the exception that negation is allowed in

an operand. The delays of timers are presented by using the same time unit as in the original description. A variable can have the form of an integer. In such a case, the variable represents an output of a gate such that the output has no name in the PLC program listing. (In order to avoid confusion about the meaning of the integers, neither 0 nor 1 is accepted for presenting a variable.) The idea is that these numbers are written onto the printed pages of the program (by using a pencil for the purpose) and then used consistently in the files. Though ELOP I supports full integer arithmetics, non-binary integer constant input lines are very rare in the Haapamäki – Seinäjoki system, and these rare cases become nicely handled by using a specific escape notation.

The intermediate language has specific "pseudo-semaphore operators" for modelling of communication. Depending on the way in which these operators are used, the communication varies between "minimally asynchronous" (by means of storage variables which together form a single message in a buffer of capacity 1) and "effectively synchronous" (due to the fact that programs can be "forced to wait"). This way of modelling the communication has practically nothing to do with the protocols MODBUS and HIBUS which in principle should be modelled. However, the state space explosion problem forces us to make coarse or even virtually absurd compromises in the modelling of communication. Our way of modelling only limits the view of the observer. In other words, the analysis is concentrated on behaviours where the PLC programs just happen to proceed in the modelled way.

The intermediate modelling language has also operators for nondeterministic choices. As said in the Introduction, nondeterminism was the worst-of-all source of state space explosion in the project.

4 Using a combination of Bc2cnf and Limmat

The input to Bc2cnf is a *Boolean circuit.* Bc2cnf transforms the circuit satisfiability problem into a classical propositional formula satisfiability problem, i.e. the formula produced by Bc2cnf (in conjunctive normal, DIMACS CNF format) is satisfiable if and only if the circuit is satisfiable under the constraint included in the input of Bc2cnf. The advantage of using Bc2cnf is that the transformation from PLC "statements" to Boolean gates is more straightforward than to CNF, whereas Bc2cnf uses algorithms that sometimes considerably reduce the circuit so that the final SAT solver is not disturbed by excessive redundancy. In the bounded model checking context below, if some assignment would satisfy a formula produced by Bc2cnf, the assignment would form a counterexample to the conjectured property.

Since the potential side effects of asynchronous communication were more or less out of the scope of the project, the translation into the input language of Bc2cnf assumes effective synchronisation of the PLC programs, i.e. no program can start a new cycle before the other programs have completed their cycles. Moreover, the project conformed to the style of [27] by assuming that any delay in any timer is a constant multiple of a PLC cycle. More precisely, it was assumed that a total cycle formed by each program executing its own cycle once takes 0.1 seconds.

In this section, we consider a "verification sanity check". There is a safety-critical mutual exclusion property that essentially says that traffic lights close to each other in opposite directions are never simultaneously green. It took long before it was observed that the model may be vacously safe in the sense that there is never any green light. So, the mutual exclusion specification was replaced by a conjecture that no light is ever green. At the time of writing this, no counterexample to this conjecture has been found though both random simulation and bounded model checking have been tried. However, the several timers with relatively long delays in the system may well cause delay in the functioning of lights as well. Thus it was decided to shorten the delays of delay-on and delay-off timers.

The alternatives considered here are as follows, where the time is in seconds, and non-integer results are assumed to be cut after the first decimal: (i) keep the original delays, (ii) replace a delay x by $\min(x, 5.0)+((x-\min(x, 5.0))/10.0)$, (iii) replace a delay x by $\min(x, 2.0)+((x-\min(x, 2.0))/20.0)$, (iv) replace a delay x by 0 (i.e. ignore it).

The number of cycles executed by each program is now the depth k of bounded model checking. The number of non-input gate definitions in the input file of Bc2cnf is $7969 \cdot k$ in case (i), $7651 \cdot k$ in case (ii), $7381 \cdot k$ in case (iii), and $5821 \cdot k$ in case (iv). In all cases, the number of input gates is $362 \cdot k$. The variation in the number of non-input gates is caused by that a counter having a narrow range can be represented with fewer bits than a counter having a wide range.

The machine for the experiments had a 1668.736 MHz CPU and 1 GB of RAM. We chose $k = 70$ because depths slightly greater than that had already caused segmentation faults in Limmat. (Note that the number of variables in the input formula of Limmat is of the same magnitude as the number of gates in the input of Bc2cnf. Limmat was actually chosen because of its good robustness when compared to some other famous SAT solvers.) The lower and upper bounds below refer to the elapsed time taken by an experiment in a setting where each experiment was repeated 20 times, whereas the actual processor time consumed by an experiment was always more than 90 percent of the elapsed time, thus confirming that there was no high competition on resources. (The concept of actual processor time is slightly academic since a program of interest can well be a primary contributor to "swapping delays" that are typically not included in the actual processor time.)

The production of the Boolean circuit took between 99 and 101 seconds in case (i), between 93 and 95 seconds in case (ii), between 88 and 90 seconds in case (iii), and between 60 and 62 secons in case (iv). The run of Bc2cnf took between 72 and 93 seconds in case (i), between 68 and 88 seconds in case (ii), between 64 and 77 seconds in case (iii), and between 25 and 35 seconds in case (iv). The run of Limmat took between 2183 and 2205 seconds in case (i), between 665 and 679 seconds in case (ii), between 2533 and 2635 seconds in case (iii), and between 2602 and 2622 seconds in case (iv). In all cases, Limmat concluded its input formula to be unsatisfiable, thus reporting a failed attack on the conjecture above.

Somewhat greater depths were covered by checking one traffic light at a time. This was not due to slicing: all the traffic light variables were interdependent. Instead, it turned out that some lights were much slower to handle than others. The constraint referring to all the lights could not have been handled faster than a slow-to-handle light alone.

5 Conclusions and related research

Applying formal methods to railway systems or programmable logic controllers is by no means any new topic. However, the publications listed in the bibliography below almost uniformly assume that at least one of the following holds: (i) analysis is integrated with the design of the system, or (ii) experts on the design of the system are deeply involved in the analysis project. Neither of these conditions was fulfilled in the project considered in this paper. The question remains whether "insightless model checking" will ever be realistic.

Most though not all publications in the first subsection in the bibliography below are included in the survey [2]. For simplicity, we comment only two papers (and do the same for the second subsection). In [9], bounded model checking is considered to be more appropriate for error detection than OBDDs. The author of the present paper came to a similar conclusion when trying analysis with several tools. On the other hand, the claim "we do not have any modeling activity" in [9] is surprising if not oversimplifying. The remarks on benefits and limitations of slicing in [20] are easy to agree with. The estimate 10^{100000} in [20] on the number of states in a system about which relevant analysis results have been obtained is impressive, especially together with the comments that indicate that the case study has been done without maximal knowledge about the system.

The publications in the PLC-specific subsection below pay much attention to the precise execution model, i.e. how exactly the programs synchronise or communicate. Unfortunately, whatever the precise execution model of the Haapamäki – Seinäjoki system might be, HUT-TCS did not have the "energy" to find it out. Looking at [28, 32], it is refreshing to observe that confusion about ambiguities in descriptions is not always due to lack of expertise.

Acknowledgements

The research tasks in the railway subproject of the Maria project were defined in co-operation by HUT-TCS, Finnish Rail Administration and Mipro Oy. Jarmo Tuomi represented Finnish Rail Administration, and Sari Becker and Matti Katajala acted as Mipro's main contact persons. Certain documents on HIMA systems and products were provided by Hans-Leo Ross from HIMA. The people associated with the subproject at HUT-TCS were Leo Ojala, Nisse Husberg, Jan Elfström, Ilkka Herttua, Lauri Tiittula, and Kimmo Varpaaniemi. Though the actual railway subproject ended in the end of the year 2000, HUT-TCS was permitted to continue the research in an untimed academic mode.

References

Related research on railway systems

[1] Cinzia Bernardeschi, Alessandro Fantechi, Stefania Gnesi, Salvatore Larosa, Giorgio Mongardi, and Dario Romano, "A Formal Verification Environment for Railway Signaling System Design," *Formal Methods in System Design*, Vol. 12, No. 2, March 1998, pp. 139–161.

[2] Dines Bjørner, "New Results and Trends in Formal Techniques & Tools for the Development of Software for Transportation Systems — A Review," in Géza Tarnai and Eckehard Schnieder (Eds.), *Proceedings of FORMS 2003, Symposium on Formal Methods for Railway Operation and Control Systems, Budapest, Hungary, May 15–16, 2003*.

[3] Arne Borälv, "Case Study: Formal Verification of a Computerized Railway Interlocking," *Formal Aspects of Computing*, Vol. 10, No. 4, 1998, pp. 338–360.

[4] Tadeusz Cichocki and Janusz Górski, "Formal Support for Fault Modelling and Analysis," in Udo Voges (Ed.), *Computer Safety, Reliability and Security, 20th International Conference, SAFECOMP 2001, Budapest, Hungary, September 26–28, 2001, Proceedings*, Lecture Notes in Computer Science, Vol. 2187, Springer-Verlag, Berlin, Germany, 2001, pp. 190–199.

[5] Alessandro Cimatti, Fausto Giunchiglia, Giorgio Mongardi, Dario Romano, Fernando Torielli, and Paolo Traverso, "Formal Verification of a Railway Interlocking System Using Model Checking," *Formal Aspects of Computing*, Vol. 10, No. 4, 1998, pp. 361–380.

[6] William John Cullyer and Wai Wong, "Application of Formal Methods to Railway Signalling—A Case Study," *Computing & Control Engineering Journal*, Vol. 4, No. 1, February 1993, pp. 15–22.

[7] Werner Damm and Jochen Klose, "Verification of a Radio-Based Signalling System Using the STATEMATE Verification Environment," *Formal Methods in System Design*, Vol. 19, No. 2, September 2001, pp. 121–141.

[8] Henning Dierks and Josef Tapken, "Tool-Supported Hierarchical Design of Distributed Real-Time Systems," in *Proceedings of the 10th Euromicro Workshop on Real-Time Systems, Berlin, Germany, June 17–19, 1998*, IEEE, 1998, pp. 222–229.

[9] Giovanni Dipoppa, Giovanni D'Alessandro, Roberto Semprini, and Enrico Tronci, "Integrating Automatic Verification of Safety Requirements in Railway Interlocking System Design," in Proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering, HASE'01, Boca Raton, FL, USA, October 22–24, 2001, IEEE, 2001, pp. 209–219.

[10] Cindy Eisner, "Using Symbolic CTL Model Checking to Verify the Railway Stations of Hoorn-Kersenboogerd and Heerhugowaard," *International Journal on Software Tools for Technology Transfer*, Vol. 4, No. 1, 2002, pp. 107–124.

[11] Lars-Henrik Eriksson and Kjell Johansson, "Using Formal Methods for Quality Assurance of Interlocking Systems," in Brian Mellit, Roland John Hill, Jeff Allan, Giuseppe Sciutto, and Carlos Alberto Brebbia (Eds.), *Computers in Railways VI* (Proceedings of COMPRAIL'98, the 6th International Conference on Computer Aided Design, Manufacture and Operation in the Railway and Other Advanced Mass Transit Systems, Lisbon, Portugal, September 2–4, 1998), Advances in Transport Series, Vol. 2, Wessex Institute of Technology Press / Computational Mechanics Publications, Southampton, UK, 1998.

[12] Jan Friso Groote, Sebastiaan Franciscus Maria van Vlijmen, and Jan Willem Cornelis Koorn, "The Safety Guaranteeing System at Station Hoorn-Kersenboogerd," in *Systems Integrity, Software Safety and Process Security, Proceedings of the 10th Annual Conference on Computer Assurance, COMPASS'95, Gaithensburg, MD, USA, June 25–29, 1995*, IEEE, 1995, pp. 57–68.

[13] Vicky Hartonas-Garmhausen, Sergio Campos, Alessandro Cimatti, Edmund Melson Clarke, Jr., and Fausto Giunchiglia, "Verification of a Safety-Critical Railway Interlocking System with Real-Time Constraints," *Science of Computer Programming*, Vol. 36, No. 1, January 2000, pp. 53–64.

[14] Anne Elisabeth Haxthausen and Jan Peleska, "Formal Development and Verification of a Distributed Railway Control System," *IEEE Transactions on Software Engineering*, Vol. 26, No. 8, August 2000, pp. 687–701.

[15] Tomas Hlavaty, Libor Preucil, and Petr Stepan, "Case Study: Formal Specification and Verification of Railway Interlocking System," in *Proceedings of the 27th Euromicro Conference, Warsaw, Poland, September 4–6, 2001*, IEEE, 2001, pp. 258–263.

[16] Michael Huber and Steve King, "Towards an Integrated Model Checker for Railway Signalling Data," in Lars-Henrik Eriksson and Peter Alexander Lindsay (Eds.), *FME 2002: Formal Methods — Getting IT Right, International Symposium of Formal Methods Europe, Copenhagen, Denmark, July 22–24, 2002, Proceedings*, Lecture Notes in Computer Science, Vol. 2391, Springer-Verlag, Berlin, Germany, 2002, pp. 204–223.

[17] Johan Lilius and Patric Östergård, "On the Verification of Programmable Logic Controller Programs," in *Proceedings of the 12th International Conference on Application and Theory of Petri Nets, Gjern, Denmark, June 26–28, 1991*, pp. 310–328.

[18] Matthew John Morley, "Safety-Level Communication in Railway Interlockings," *Science of Computer Programming*, Vol. 29, No. 1–2, July 1997, pp. 147–170.

[19] Fredrik Orava, "Formal Analysis of a Railway Interlocking System," in Volkan Atalay, Uğur Halıcı, Kemal İnan, Neşe Yalabık, and Adnan Yazıcı (Eds.), *Proceedings of the Eleventh International Symposium on Computer and Information Sciences, ISCIS XI, November 6–8, 1996, Antalya, Turkey, Vol. I*, Middle East Technical University, Ankara, Turkey, 1996, pp. 17–27.

[20] Jakob Lyng Petersen, "Automatic Verification of Railway Interlocking Systems: A Case Study," in Mark Ardis and Joanne Atlee (Eds.), *Proceedings of the 2nd ACM SIGSOFT Workshop on Formal Methods in Software Practice, Clearwater Beach, FL, USA, March 4–5, 1998*, ACM Press, New York, NY, USA, 1998, pp. 1–6.

[21] Gunnar Stålmarck and Mårten Säflund, "Modeling and Verifying Systems and Software in Propositional Logic," in Barry K. Daniels (Ed.), *Safety of Computer Control Systems 1990 (SAFECOMP'90), Safety, Security and Reliability Related Computers for the 1990s, Proceedings of the IFAC/EWICS/SARS Symposium, Gatwick, UK, 30 October – 2 November 1990*, IFAC Symposia Series, No. 17, Pergamon Press, Oxford, UK, 1990, pp. 31–36.

[22] Kirsten Winter, "Model Checking Railway Interlocking Systems," *Australian Computer Science Communications*, Vol. 24, No. 1, January – February 2002, pp. 303–310.

Related research on PLCs (unless already listed)

[23] Nanette Bauer and Ralf Huuck, "Towards Automatic Verification of Embedded Control Software," in *Proceedings of the 2nd Asia-Pacific Conference on Quality Software, Hong Kong, China, December 10–11, 2001*, IEEE, 2001, pp. 375–383.

[24] Arne Borälv and Herman Ågren, *Formal Verification of Programmable Logic Controllers*, Uppsala Master's Theses in Computing Science 82, Computing Science Department, Uppsala University, Uppsala, Sweden, November 1995, iv+88 p.

[25] Ed Brinksma, Angelika Mader, and Ansgar Fehnker, "Verification and Optimization of a PLC Control Schedule," *International Journal on Software Tools for Technology Transfer*, Vol. 4., No. 1, 2002, pp. 21–33.

[26] Peter Deussen, "Partial Order Verification of Programmable Logic Controllers," in José Manuel Colom and Maciej Koutny (Eds.), *Applications and Theory of Petri Nets 2001, 22nd International Conference, ICATPN 2001, Newcastle upon Tyne, UK, June 25–29, 2001, Proceedings*, Lecture Notes in Computer Science, Vol. 2075, Springer-Verlag, Berlin, Germany, 2001, pp. 144–163.

[27] Monika Heiner and Thomas Menzel, "Time-related Modelling of PLC Systems with Time-less Petri Nets," in René Boel and Geert Stremersch (Eds.), *Discrete Event Systems: Analysis and Control* (Proceedings of WODES2000, the 5th Workshop on Discrete Event Systems, Ghent, Belgium, August 21–23, 2000), The Kluwer International Series in Engineering and Computer Science, Vol. 569, Kluwer Academic Publishers, Boston, MA, USA, 2000, pp. 275–282.

[28] Anders Hellgren, Martin Fabian, and Bengt Lennartson, "On the Execution of Discrete Event Systems as Sequential Function Charts," in Proceedings of the 2001 IEEE International Conference on Control Applications, CCA'01, Mexico City, Mexico, September 5–7, 2001, IEEE, 2001, pp. 428–433.

[29] Il Moon, "Modeling Programmable Logic Controllers for Logic Verification," *IEEE Control Systems Magazine*, Vol. 14, No. 2, April 1994, pp. 53–59.

[30] Mathias Rausch and Bruce H. Krogh, "Formal Verification of PLC Programs," in *Proceedings of the 1998 American Control Conference, Philadelphia, PA, USA, June 21–26, 1998, Vol. 1*, IEEE, 1998, pp. 234–238.

[31] Norbert Völker and Bernd J. Krämer, "Automated Verification of Function Block Based Industrial Control Systems," *Science of Computer Programming*, Vol. 42, No. 1, January 2002, pp. 101–113.

[32] Martin Öhman, Stefan Johansson, and Karl-Erik Årzén, "Implementation Aspects of the PLC Standard IEC 1131-3," *Control Engineering Practice*, Vol. 6, No. 4, April 1998, pp. 547–555.

Manuals and standards

[33] HIMA (Paul Hildebrandt GmbH + Co KG Industrie-Automatisierung), *Description of the Building Block HK-MMT-1: MODBUS Master with Telephone Modem*, Edition 9425, HIMA, Brühl, Germany, 21 p.

[34] HIMA (Paul Hildebrandt GmbH + Co KG Industrie-Automatisierung), "Features and Extensions of the Logic Symbols," in Chapter IV in one of the manuals of HIMA System Software ELOP I, HIMA, Brühl, Germany, pp. 55–57.

[35] HIMA (Paul Hildebrandt GmbH + Co KG Industrie-Automatisierung), *Safety Manual TI 96.05 (9814)*, HIMA, Brühl, Germany.

[36] IEC (International Electrotechnical Commission), *International Standard IEC 1131-3; Programmable Controllers — Part 3: Programming Languages*, First edition 1993-03 (Reference number IEC 1131-3: 1993(E)), IEC Central Office, Geneva, Switzerland, 1993, 207 p.

[37] Modicon, Inc., Industrial Automation Systems, *Modicon Modbus Protocol Reference Guide (PI-MBUS-300)*, Modicon, Inc., North Andover, MA, USA.

Primary references on bounded model checking

[38] Nina Amla, Robert P. Kurshan, Kenneth L. McMillan, and Ricardo Medel, "Experimental Analysis of Different Techniques for Bounded Model Checking," in Hubert Garavel and John Hatcliff (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7–11, 2003, Proceedings*, Lecture Notes in Computer Science, Vol. 2619, Springer-Verlag, Berlin, Germany, 2003, pp. 34–48.

[39] Armin Biere, Alessandro Cimatti, Edmund Melson Clarke, Jr., and Yunshan Zhu, "Symbolic Model Checking without BDDs," in Walter Rance Cleaveland II (Ed.), *Tools and Algorithms for the Construction and Analysis of Systems, 5th International Conference, TACAS'99, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22–28, 1999, Proceedings*, Lecture Notes in Computer Science, Vol. 1579, Springer-Verlag, Berlin, Germany, 1999, pp. 193–207.

Software

[40] Armin Biere, *Limmat Satisfiability Solver*, Swiss Federal Institute of Technology Zürich, Department of Computer Science, Institute of Computer Systems, Zürich, Switzerland, http://www.inf.ethz.ch/personal/biere/projects/limmat/.

[41] Tommi Junttila, *Bc2cnf — A Program for Translating Constrained Boolean Circuits to Equi-Satisfiable Boolean Formulae in DIMACS CNF Format*, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, http://www.tcs.hut.fi/users/tjunttil/public_html/circuits/.