Modelling of a PLC-Based Railway Traffic Control System

Kimmo Varpaaniemi

Abstract

This paper describes the modelling of a distributed PLC system that was developed by Mipro Oy and controls the traffic in the railway section between Haapamäki and Seinäjoki. The PROD tool of HUT-TCS is used for analysing the model.

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*.

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 [8]. The PROD tool [12], a successor of PRENA, was created in the beginning of the 90's and has been developed further since then. For PRENA and PROD, the model is given as a high-level Petri net. Some methods for relieving the state space explosion problem have been implemented in PROD. Particular effort has been put into the implementation of the *stubborn set method* [10, 11]. The stubborn set method belongs to the class of partial order methods and is based on the idea that one interleaving of actions is often sufficient for representing several interleavings.

At the end of the 90's, the laboratory started to develop a new high-level Petri net reachability analysis tool, called MARIA. The development is a part in a project which has the name MARIA too. The MARIA project has a few subprojects where case studies have been or are being carried out. One of these subprojects is concentrated on modelling and analysis of a PLC-based railway traffic control system. This distributed PLC system was developed by Mipro Oy, and it controls the traffic in the railway section between Haapamäki and Seinäjoki. PROD is the primary tool used in the analysis. This paper describes the modelling of the system. The described work started in March 2000, and the model was ready for analysis in August 2000.

2 The system which was modelled

The modelled system was developed by using the ELOP I package of HIMA [6]. HIMA also has the ELOP II package which is somewhat more compatible with the IEC standard [7] 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 [5] to the corresponding description in [7]. 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 [7], 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. [[Quotation ends.]]"

Figure 1 shows an essential copy of an actual page in a program listing of one of the PLC programs of the Haapamäki – Seinäjoki system. The figure contains two instances of the function block which we call TR_YPK01. (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 figure, 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 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.

When modelling the execution of a statement, we of course have the freedom to decompose the statement into a sequence of assignments to "imaginary" intermediate variables. (Such decompositions are useful at least for the stubborn set method which has obvious problems with "large atomic actions".)

Let us then look closer at TR_YPK01 and its instances. Figure 2 shows the description of the block. Each execution of the block consists of 15 statements since there are as many statements as there are names on the right-hand side of the figure. 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 in 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

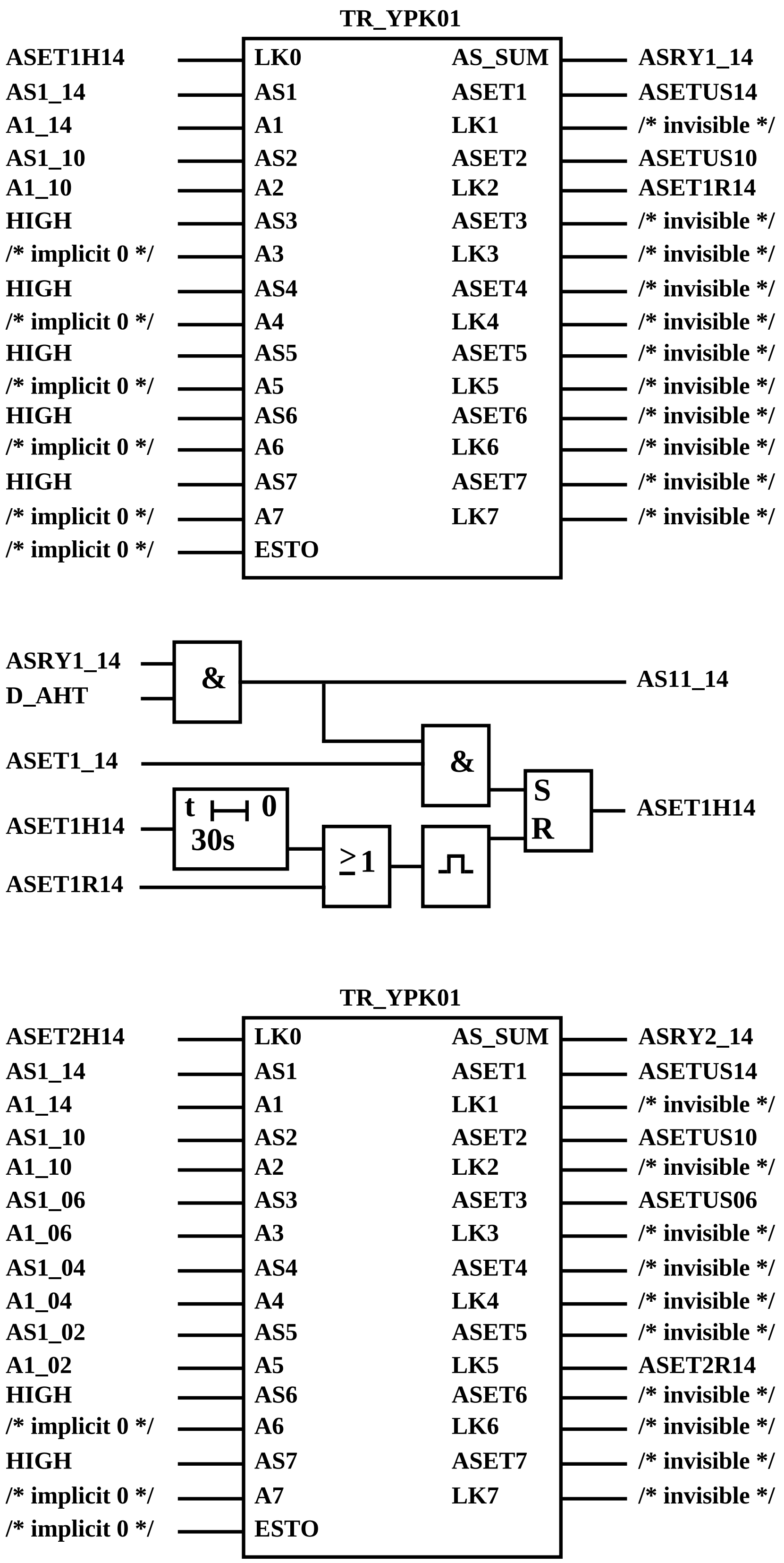

Figure 1: A sample page in a PLC program listing (essentially a copy of page 16 of Mipro's SK2-KL01.98, version E014).

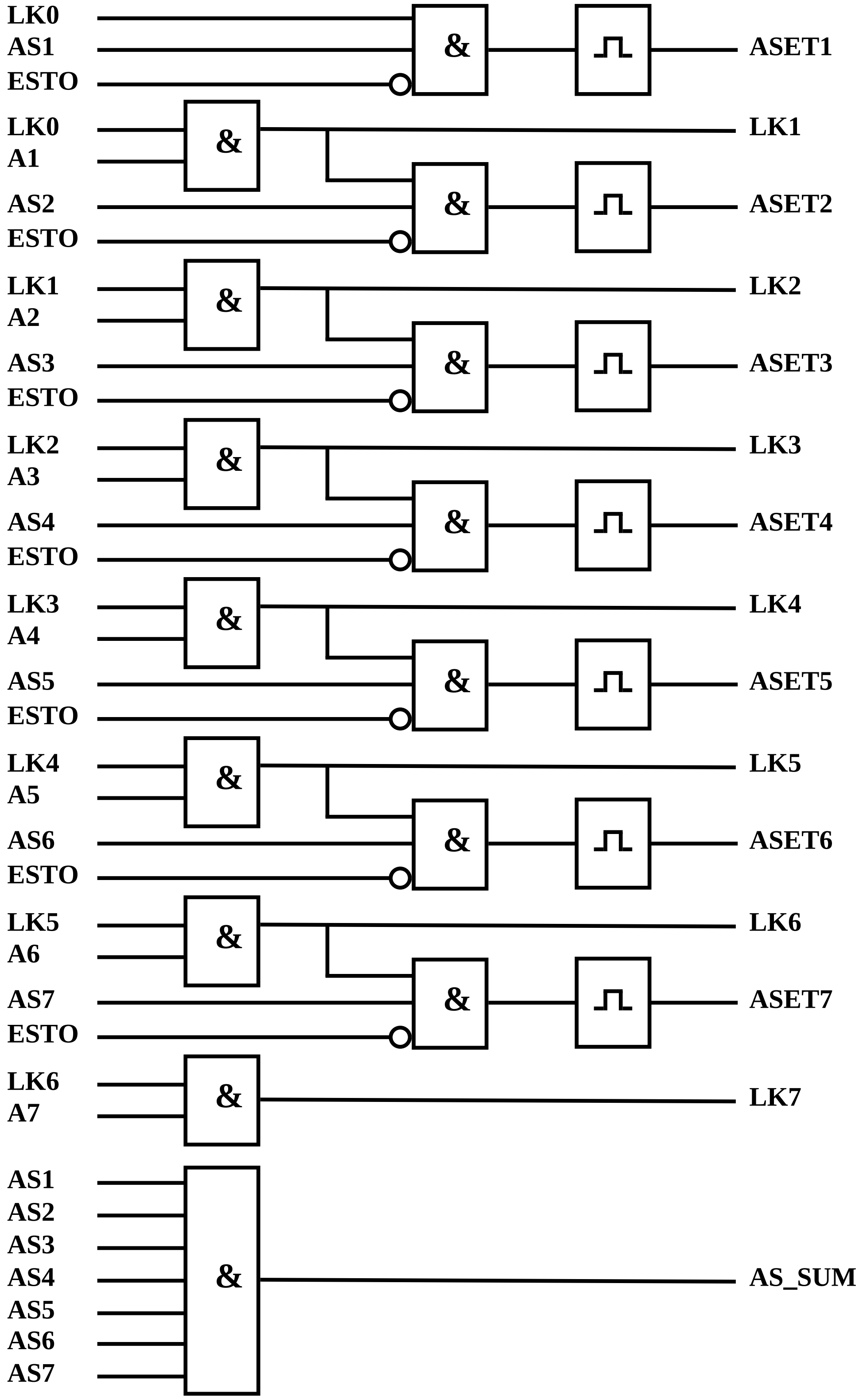

Figure 2: The logic of the TR_YPK01 block (essentially a copy of page 2 of Mipro's TR-YPK01.10, version 574E).

[4, 6, 9] and HIBUS [6], 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 the high-level Petri net model and the 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 in Figure 1 looks as follows. An appropriate choice for the name of the file is `sk2_kl01_98_p16_0.mip`. The reduction in the number of lines is explained by the fact that many of the formal output parameters are simply redundant w.r.t. this instance.

```
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
```

Respectively, the file for the lower instance of TR_YPK01 in Figure 1 looks as follows. An appropriate choice for the name of the file is `sk2_kl01_98_p16_2.mip`.

```
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 Figure 1 is written “directly by hand” and looks as follows. An appropriate choice for the name of the file is `sk2_kl01_98_p16_1.mip`.

```
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.

We have not yet mentioned the worst-of-all source of state space explosion problem in the Haapamäki – Seinäjoki system, namely the so called *testable inputs*. Testable inputs are variables which can become changed fully arbitrarily, each time the program “measures” them. The intermediate language has an operator for describing such nondeterminism. Though many of the testable inputs defined in the programs can be and actually have been abstracted away, there are still sufficiently many of them to keep the number of states astronomical. The unfortunate thing w.r.t. PROD is that the stubborn set method alone is strictly insufficient against this kind of nondeterminism. (This follows from the fact that the nondeterminism remains even if the system is artificially sequentialised by means of static priorities.)

4 On the high-level Petri net model

There is a filter which produces a high-level Petri net description for PROD from the files written in the intermediate language. The translation is mostly not of general interest. The problem of how the semantics of the gates should be translated can be reduced to a problem of how a few macros should be defined. Due to the state space explosion problem and the lack of a time concept in PROD, some compromises w.r.t. the timers are necessary. An apparently good approximation is to assume that every delay is a constant multiple of a PLC cycle. This is what has been done in [3]. Having that approximation and using [5, 7], it was actually very easy to define the needed macros. If it later turns out that some of the macros has an inappropriate definition, it is easy to simply change the definition.

From the programmer's point of view, the translation had several more or less interesting problems. One of the problems was how to keep the number of high-level transitions small. Namely, PROD itself uses a compilation approach where the amount of produced code is proportional to the number of high-level transitions. The solution was to define distinct high-level transitions for only those operations which really seemed to need distinct high-level transitions. Since PROD's net input language has always had the C language on the bottom, it was natural to use constant C arrays in the concrete transition descriptions.

From the analysis point of view, the model is challenging. It has already inspired one methodological "dirty trick": non-branching points in a reachability graph can be eliminated without even temporary book-keeping when the graph is known to have sufficiently many branching points which exclude the possibility of getting into an infinite loop in the elimination.

5 Conclusions and acknowledgements

This work has been funded by The National Technology Agency of Finland, Finnish Rail Administration, Elisa Communications, Nokia Networks, and Nokia Research Center. The research tasks in the railway subproject of MARIA have been defined in co-operation by HUT-TCS, Finnish Rail Administration and Mipro Oy. Jarmo Tuomi has been the main contact person at Finnish Rail Administration. Sari Becker and Matti Katajala, the main contact persons at Mipro Oy, have provided all the program listings and explained the semantics of the system. HIMA documents have been provided by Mipro Oy and by Hans-Leo Ross from HIMA.

Jan Elfström wrote specifications [1] on the basis of the safety documents of Finnish Rail Administration (e.g. [2]). These specifications were ready for analysis in September 2000. The analysis itself should produce results by the end of the year 2000. Due to the state space explosion problem, the amount and impressiveness of the results depend mostly on luck.

Also the earlier years in the subproject included interesting modelling and analysis problems. Lauri Tiittula and Ilkka Herttua did some of the work then, but the author of this paper is responsible for the false alarms that were received by Finnish Rail Administration in January 2000.

References

[1] Jan Elfström, *Haapamäen ja Seinäjoen välisen rataosuuden liikenneohjausohjelmiston vaatimusten formalisointi* (an internal report of the MARIA project), Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, October 2000.

[2] Finnish Rail Administration (Ratahallintokeskus, RHK), *Elektronisen asetinlaitteen vaatimukset*, RHK 1232/732/97, Helsinki, Finland, September 1997.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] Modicon, Inc., Industrial Automation Systems, *Modicon Modbus Protocol Reference Guide (PI-MBUS-300)*, Modicon, Inc., North Andover MA, USA (http://www.modicon.com/techpubs/toc7.html).

[10] Antti Valmari, *State Space Generation: Efficiency and Practicality*, Doctoral thesis, Tampere University of Technology, Publications 55, Tampere, Finland, December 1988, 170 p.

[11] Kimmo Varpaaniemi, *On the Stubborn Set Method in Reduced State Space Generation*, Doctoral thesis, Helsinki University of Technology, Digital Systems Laboratory Report A 51, Espoo, Finland, May 1998, 105 p.

[12] Worldwide web, page http://www.tcs.hut.fi/Software/prod/index.html.