Application of Symbolic and Bounded Model Checking to the Verification of Logic Control Systems

Kingliana Loeis, Mohammed Bani Younis and Georg Frey
Juniorprofessorship Agentbased Automation
University of Kaiserslautern
Erwin-Schrödinger-Str. 12, Kaiserslautern
{loeis, baniy, frey}@eit.uni-kl.de

Abstract

The developer of logic control systems is faced with increasing complexity of the functions to be implemented and, at the same time, increasing demands on the reliability of the resulting software. To analyze the reliability of such complex systems formal methods can be applied. One area of the corresponding research is focused on the application of model checking techniques to Programmable Logic Controllers (PLCs). In this paper a new method to formalize PLC programs together with a model of the cyclic behavior of the PLC is presented. The control systems behavior is modeled, and then the program, written in Instruction List, is formalized and integrated into the model. The formalization in SMV language is suitable for verification using BDD and SAT techniques. Both techniques are compared using first results of a case study.

1. Introduction

Control systems run by Programmable Logic Controllers (PLCs) are often complex, safety critical and involve a lot of money. Any failure of these systems might not only result in a significant financial loss, but lead to casualties as well. Thus, next to robustness and reliability of the hardware and system software, their actual programming and the correctness of the programs play a vital role.

Model checking has proven to be a powerful technique not only for the verification of hardware components, but also in the automation field. Symbolic Model Checking employs Binary Decision Diagrams (BDDs) as symbolic representation of the system. This greatly reduces the number of states opposed to a direct (non-symbolic) representation of the systems state space. However, the possibility of a state explosion problem still exists. There is a new technique that avoids the space blow up of BDDs, called Bounded Model Checking (BMC). In BMC the verification problem is transferred into a propositional satisfiability problem (SAT). This new technique is believed to be feasible in cases where the BDD technique produces too large state spaces.

In the presented work, a formal model of the PLC behavior and a formalization approach for PLC programs is introduced using the SMV language. A new algorithm was developed to allow automatic generation of the model for verification. Therefore, errors introduced by users can be avoided. Using the NuSMV [1] tool, the models are suitable for both verification techniques, BDD and SAT.

In the following section a short introduction to verification by model checking is given. Section 3 explains a new algorithm for the translation of PLC program into the SMV language, where the model of the PLC program and the PLC behavior are integrated in a modular way. Some notes on the implementation of this translation are given in section 4. Section 5 presents results from two case studies of technical systems. Section 6 concludes the paper and gives an outlook on further work.

2. Verification by model checking

Model checking is a three steps process where in the first step the system description is converted into a formal model. The second step is to generate the formal description of properties to be verified. In a third step a model checker generates a model of the state space evaluating the properties in each state (Kripke structure). The verification step then shows whether a property holds or not. In case of a property that does not hold a counterexample is created.

For the formal specification of behavioral properties of systems, temporal logic as an indispensable specification language tool is utilized. There is Linear Time Temporal Logic (LTL) as the model of choice to characterize the set of *all execution sequences* beginning at a state of the transition system with its temporal operators of G, F, U and X. For two properties p and q: $G\,p$ is true if p holds in all states of the sequence, $F\,p$ is true if p holds at some time in the future, $q\,U\,p$ is true if q holds until p holds, and $X\,p$ is true if p holds in the next state. On the other hand, the branching time approach, namely Computation Tree Logic (CTL), considers the set of *all execution paths* generated by the system. Two path quantifiers A and E as an addition to the temporal

0-7803-9402-X/05/$20.00 © 2005 IEEE

operators are defined. Here *A* means for all paths whereas *E* means there exists one path. In general the path quantifiers are combined with the temporal quantifiers. As an example: *AG p* is true if p holds in all possible system states and *EF p* is true if the property holds in at least one state.

In generating the state space of the model, symbolic model checking addresses the state explosion problem by using a graph representation called Binary Decision Diagram (BDD) [2]. The BDD based model checker of the NuSMV tool allows the use of LTL and CTL for the specification of properties.

In Bounded Model Checking only execution sequences of a system up to a given length are considered. Based on this restriction, the model checking problem can be reduced into a propositional satisfiability problem (SAT) [3]. This enables the use of efficient propositional SAT solvers to perform model checking. The NuSMV tool can only generate SAT descriptions for properties given in LTL.

Talking about properties as requirement specification, model checking procedures often do not check a formula of interest directly; rather they check its negation. This is due to the fact, that when checking whether the system satisfies *p,* the model checking algorithm constructs an automaton $M_{\neg P}$ that accepts all the satisfying models of $\neg p$, and tests emptiness of the intersection of the languages of the automata of the system and $M_{\neg P}$. This can be exploited to reduce the state space. For instance, *AG p*, "*p* is true in every reachable state" can be shown to be true by showing that $EF\ \neg p$, "there is a reachable state in which *p* is false" is false. Thus, the counterexample for *AG p* is a witness or demonstration of validity for $EF\ \neg p$. Likewise, a counterexample for *AF p* is a witness for $EG\ \neg p$.

In bounded model checking finite sequences of states in the system are considered. This means prior to building the SAT formulation a parameter k_{max} has to be specified giving the maximum length of the sequences to be considered. In NuSMV then, starting from k = 1 until $k = k_{max}$ SAT descriptions are generated and evaluated. However, if a counter-example is found the process terminates. This implies – as expected – that the time needed to find a counter-example by SAT increases with the length of the execution sequence that leads to it.

3. Modeling approach for PLC programs

3.1. Structure of a PLC system and the model

In order to achieve more conformity of the different PLC programming languages, the standard IEC 61131-3 was developed [4]. In this work, Instruction List (IL) is chosen to be applied since it allows compact code and is very close to hardware programming.

The main difference of PLCs to conventional systems such as PCs is the cyclic operation mode which is illustrated in Figure 1. It describes that PLC programs are executed in a permanent loop, where periodically inputs are sampled, computations based on the inputs and the internal state of the controller are performed and afterwards, the corresponding output is emitted.

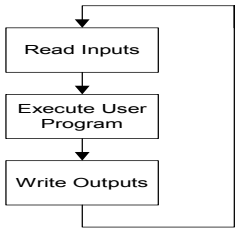


Figure 1. Cyclic execution of a PLC.

The development of a formal description for PLC programs together with the PLCs cyclic behavior can be carried out in a modular way. Each step in the PLC cycle is described in a separate module, where the acknowledgement of each finished step acts as a parameter to the others. The aim is that the formalized IL of a system can be integrated and directly implemented into the proposed model of the PLC cycle. For building up the overall model, there are five modules to be generated: module *Reader* to read the input, module *Executor* to execute the actual control algorithm, module *Writer* to write the output, module *Channel* as the bridge among the three and module *Main* to initialize the system. In the following the different modules are described.

3.2. Modules Reader and Writer

As shown in Figure 2, the module Reader contains two control states and one internal variable to be assigned. The control states are *idle_read* as initial state and *reading*; the internal variable is *read* in Boolean (active or inactive). If the PLC reads the input by an active internal variable *read*, the next state is *reading*. Otherwise, it goes and stays in state *idle_read*.

The module Reader includes the parameters (in italic) from variables belonging to the other modules as their communication protocols. These are *exec*, a command to execute from module Executor; *write* which is a command to write the output from module Writer, and *ack_read* which represents the acknowledgements for finishing input reading step from module Channel.

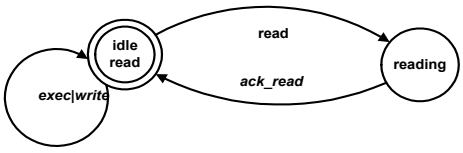


Figure 2. Module Reader.

Like module Reader, module Writer has a generic form for every integrated PLC programs. In this module

there are two control states; *idle_write* as initial state and *writing*. As variable, *write* is a command to write the output. When the PLC writes the output ordered by an active *write*, then the state moves from *idle_write* to *writing*. Otherwise, under cooperation with parameters from other modules, it will stay in *idle_write*.

3.3. Module Executor

The module Executor provides a place for the PLC programs to be integrated. The control system is given or converted into IL and then formalized into SMV. It is afterwards ready to be embodied in a framework structure of the module Executor.

For example, the following is a simple case of fan monitoring. A notice lamp will turn on in long period, when the plant is in operation mode and from three fans, at least two are running. In other cases, the lamp should flash with 0.5-Hz if only one fan runs and with 2-Hz for no running fan. The definition of the variables and the corresponding IL code are given in Figure 3.

E	38.1	Fan 1
E	38.2	Fan 2
E	38.3	Fan 3
M	99.1	2-Hz blinking sign
M	99.2	0.5-Hz blinking sign
M	100.0	Temporary sign 1
M	100.1	Temporary sign 2
A	42.4	The plant is in operation mode
A	51.7	The notice lamp

```
0000   :U    E  38.1  //AND Operation
0002   :U    E  38.2
0004   :O             //OR Operation
0006   :U    E  38.1
0008   :U    E  38.3
000A   :O
000C   :U    E  38.2
000E   :U    E  38.3
0010   :=    M 100.0  // at least two Fans run
0012   :UN   E  38.1  // ANDN Operation
0014   :UN   E  38.2
0016   :UN   E  38.3
0018   :=    M 100.1  // no running Fan
001A   :U(
001C   :O    M 100.0  // Continuous Light
001E   :O
0020   :U    M 100.1
0022   :U    M  99.1  // Flashing with 2 Hz
0024   :O
0026   :UN   M 100.0  // Only one Fan runs
0028   :UN   M 100.1
002A   :U    M  99.2  // Flashing with 0,5 Hz
002C   :)
002E   :U    A  42.4  // Active LCD lamp
0030   :=    A  51.7
0032   :BE
```

Figure 3. IL code of the example.

The input signals are assigned to be Boolean (active or inactive) as they are binary inputs. After the execution of the algorithm is finished and acknowledged by the variable *ack_exec*, the output is saved into the output memory (PAA). The amount of output memory depends on the total existing output. While the PLC is sending output to the environment through actuators (module Writer), the module Executor goes back to the *idle_exec* condition. To have this situation, the two final states of the original Moore machine are connected to the state *idle_exec* as illustrated in Figure 4. The conditions depicted on the transition system of Figure 4 are as follows:

```
I1 =   E 38.1 AND E 38.2 OR E 38.1 AND E 38.3
       OR E 38.2 AND E 38.3
I2 =   NOT E 38.1 ANDN E 38.2 ANDN E 38.3
I3 =   ( M 100.0 OR M 100.1 AND M 99.1 OR NOT M
       100.0 ANDN M 100.1 AND M 99.2)AND A 42.4
```

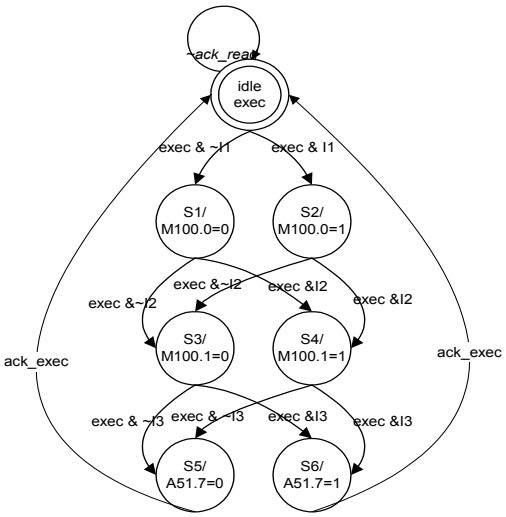


Figure 4. Executor model for the example.

3.4. Modules Channel and Main

The Channel performs as a bridge among the three other modules. It basically realizes the PLC cycle presented in Figure 1. There are three control states to be assigned, namely *to_read*, *to_exec*, and *to_write*.

Additionally there is a module Main defined. It acts as starting point to build the finite state model in SMV. Variables in module Main are those previous four modules containing their corresponding parameters. The specifications to be checked are also placed in this module.

4. Implementation

The encoding of the algorithm is implemented generically in Java, where an SMV file is generated as an input for verification process based on the algorithm and the PLC cycle. This SMV file is generated step by step starting from the IL of the PLC program (cf. Figure 3), which is formalized according to the algorithms given in [5] into a Mealy automaton. This Mealy automaton is again transformed to an automaton of Moore type (compare Figure 4) which is automatically converted to SMV language (cf. Figure 5). The actual code of the PLC algorithm is embodied into the SMV

code of the module Executor. The SMV of the other modules remains unchanged. This implementation employs XML as an intermediate step for all these transformations. The XML for the PLC program is obtained using the method presented in [6] and is manipulated to give another XML with categories for the instructions which allows applying the algorithms to transform the IL code to a Mealy automaton.

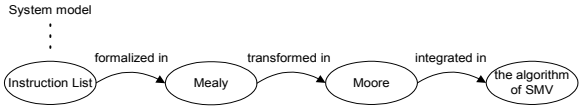


Figure 5. From instruction list to SMV.

5. Case studies

An air compressor system [7] and a station of a Bosch Rexroth manufacturing line at the LURPA (ENS Cachan, France) [8] are used as case studies. The designs are written in SIPN [9] and Grafcet [10] respectively. The reachable states are 810 for the first and 2^{20} states for the second example. One requirement from the first example states: if the air pressure falls under 6.1 bars, i.e. the sensor *ps1* switches on (*E10* = 1) and *ps2* stays off (*E11* = 0), then one motor should run. The running of a motor is forced by the outputs A 11 for motor 1 and A12 for motor 2 respectively. The property can be formalized in CTL as follows:

```
AG(executor.E10&!executor.E11&reader.read)→AF((
exector.A11=1&executor.A12=0)|(executor.A11=0&e
xecutor.A12=1)))
```

To find a counterexample, the requirement has to be written in its opposite by using the duality of temporal logic and De Morgan's laws as follows:

```
EF((executor.E10&!executor.E11&reader.read)&EG!
((exector.A11=1&executor.A12=0)|(executor.A11=0
&executor.A12=1)))
```

To use the SAT solver the property has to be re-formulated in LTL. This re-formulation should start from the original CTL property:

```
!G((executor.E10&!executor.E11&reader.read)→F((
exector.A11=1&executor.A12=0)|(executor.A11=0&e
xecutor.A12=1)))
```

The BDD based method finds a counterexample in 0.7 seconds, whereas the SAT solver needs 83 seconds with a bound of k = 11. By doing the same comparison for one requirement from the second example which states: if a pallet reaches the working position then the lifting process is stopped, BDD needs 15 seconds to get the counterexample, whereas SAT needs 1425429 seconds with a bound of 126. The evaluation of further properties showed a similar behavior.

However, in reasoning about these results it has to be kept in mind that all the checked properties have been of similar structure. The structure seems to be not favorable for SAT because the properties are required to hold for all complete system sequences (*AG*). This leads to high values of the bound and consequently to long execution time.

6. Conclusions and outlook

The paper presents a new approach for the formal modeling of PLC programs to be used in model checking. All the described steps in converting a PLC program to SMV code are implemented in a prototypical Java tool. The SMV code was used to compare model checking based on BDD and on SAT. The investigations by now show that the BDD technique seems to be favorable as long as the state explosion problem is not hit and general properties are investigated.

In further studies it is expected that for several special properties the SAT based methods will be found superior. Especially, in the case where a property itself is defined over a bounded horizon better results for SAT are possible. Furthermore SAT is a relatively new approach in comparison to BDD based model checking. Therefore, further improvements in the algorithms could be expected.

References

[1] NuSMV homepage: http://nusmv.irst.itc.it/

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. Dill, J. L. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond", *Information and Computation, 98(2):* pp. 142-170, June 1992.

[3] A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu, "Symbolic Model Checking without BDDs*", In TACAS'99*, *LNCS 1579*. Springer, pp. 193-207, 1999.

[4] International Electrotechnical Commission, IEC International Standard 1131-3, Programmable Controllers, Part 3, Programming Languages, 1993.

[5] M. Bani Younis, G. Frey, "Formalization of PLC Programs to Sustain Reliability," *Proceeding IEEE, RAM-2004*, Singapore, pp. 613-618, Dec. 2004.

[6] M. Bani Younis and G. Frey, Visualization of PLC Programs Using XML. *Proceedings of the American Control Conference (ACC2004),* Boston, USA, June 30 - July 2, pp. 3082-3087, 2004

[7] L. Litz, *Grundlagen der Automatisierungstechnik,* Oldenburg Verlag München, 2005.

[8] D. Evrot, *Identification de machines séquentielles parallèles en tenant compte de la structure PO. D. E. A. De Production Automatisée,* Ecole Normale Supérieure de Cachan, July 2004.

[9] G. Frey, *Design and Formal Analysis of Petri Net Based Logic Control Algorithms,* ISBN 3-8322-0043-6, Dissertation, University of Kaiserslautern, Shaker Verlag, Aachen, April 2002.

[10] R. David, H. Alla, *Petri Nets & Grafcet Tools for Modelling Discrete Event Systems,* Prentice Hall, New York, London, 1992.