Title	Model checking methodology for verification of safety logics
Author(s)	Valkonen, Janne; Björkman, Kim; Frits, Juho; Niemelä, Ilkka
Citation	SIAS 2010 - The 6th International Conference on Safety of Industrial Automated Systems, Tampere, 14.-15.6.2010
Date	2010
Rights	Copyright © (2010) Suomen Automaatioseura. Reprinted from SIAS 2010 - The 6th International Conference on Safety of Industrial Automated Systems, Tampere, 14.-15.6.2010, ISBN: 978-952-5183-40-5. This article may be downloaded for personal use only

Model Checking Methodology for Verification of Safety Logics

Janne Valkonen, Kim Björkman
Technical Research Centre of Finland (VTT) P.O. Box 1000, FI-02044 VTT, Finland
Tel. +358 20 722 111, E-mail janne.valkonen@vtt.fi, kim.bjorkman@vtt.fi, http://www.vtt.fi

Juho Frits, Ilkka Niemelä
Aalto University, School of Science and Technology, Dept. of Information and Computer Science
P.0.Box 15400, FI-00076 AALTO, Finland
Tel. +358 9 47001, E-mail ilkka.niemela@tkk.fi, jfrits@tcs.hut.fi, http://www.aalto.fi

KEY WORDS model checking, verification, safety logic, digital I&C

ABSTRACT

Verification of safety critical digital instrumentation and control (I&C) systems is challenging because of more and more complicated control functions enabled by programmable logic controllers. Design verification is an important task in the design flow because it enables to detect design errors earlier and helps to avoid expensive redesign and reimplementation work caused by undetected design errors found later. Systems have been typically verified by testing and simulation techniques. Both approaches have their advantages and are useful in many situations but in cases where exhaustive verification with reasonable effort and time is needed, none of them alone is suitable. Model checking is a computer-aided formal method that can be used for verifying correct functioning of a system design model. In model checking the task is to determine whether a model of a system satisfies a given requirement, which is checked against all executions of the system model. In addition to introducing the model checking methodology, this paper gives examples of industrial cases where model checking has been successfully used and discusses its applicability to verifying safety logic designs.

1 INTRODUCTION

Verification of safety critical digital instrumentation and control (I&C) systems is challenging because programmable logic controllers enable complicated control functions and the state spaces of the designs become easily too large for comprehensive manual inspection. Design verification is a key task in the design flow because it can eliminate tricky design errors which are hard to detect later in the development process and are very expensive to repair leading often to a major redesign and reimplementation cycle.

Typically, systems have been verified by testing and simulation techniques. There are several subtypes of testing (black box, white box, etc.), but the basic idea in all testing is using the system itself for getting evidence of its correctness. In simulation, the aim is to imitate the system behaviour by a model and verify the correctness of the system by simulating different scenarios one by one. That is time-consuming and impractical when the number of reachable states of the system grows up to thousands or millions. An alternative formal method is deductive verification, which uses axioms and proof rules to prove the correctness of systems. It is time-consuming and can be performed only by experts with considerable experience. These three methods have their advantages and they are useful in many situations. However, in cases where exhaustive verification with reasonable effort and time is needed, none of them alone is suitable.

Model checking is a computer-aided formal method that can be used for verifying correct functioning of a system design model. In model checking, the task is to determine whether a model of a system satisfies a given requirement, which is checked against all executions of the system model. This paper introduces the model checking methodology, gives examples of industrial cases where it has been successfully used, and discusses its benefits and limitations in verifying safety logic designs.

2 MODEL CHECKING METHODOLOGY

Model checking /7/ is a computer-aided verification method developed to formally verify correct functioning of a system design model by examining all of its possible behaviours. The models used in model checking are quite similar to those used in simulation, as basically the model must describe the behaviour of the system design for all sequences of inputs. However, unlike simulation, model checkers examine the behaviour of the system design with all input sequences and compare it to the specification of the system. In model checking, at least in principle, the analysis can be made fully automatic with computer aided tools. The specification is expressed in a suitable language, temporal logics being a prime example, describing the allowed behaviours of a system. Given

a model and a specification as input, a model checking algorithm decides whether the system violates its specification or not. If none of the behaviours of the system violates the given specification, the model of the system is correct. Otherwise, the model checker will automatically give a counter example execution of the system demonstrating why the property is violated.

Using model checking for system design verification involves typically the following four steps: (i) modelling the design, (ii) requirement specification, (iii) running the model checker, (iv) interpreting the results. Next we briefly describe the methodology that we have used for these steps. For model checking, the system design needs to be captured using the modelling language supported by the model checking tool to be used. Modelling languages of state-of-the-art model checking tools support well modelling of typical components used in digital I&C system designs. For systematic modelling methodology, a key issue is to develop an approach that leads to models that are easy to review, revise and update and that support traceability. In our approach this means identifying the system boundaries and the interface between the system and its environment and exploiting the component structure of the design to create a corresponding modular modelling approach. Another important issue is choosing an appropriate level of abstraction for the model so that irrelevant details are abstracted away and, thus, the computational cost of performing the model checking task remains reasonable. When using model checking to verify whether a system design satisfies a specification, this is done against an environment model describing how the environment and the system interact. In our approach we use simple environment models that allow the environment to behave quite freely and independent of the system design under verification. This leads to safe model checking results: if the model checking tool determines that the system model satisfies the specification, then this is the case in all kinds of environments and the correct behaviour is not based on assumptions on the environment.

Another key step is requirement specification where requirements are typically given as natural language statements that are formalized in the specification language supported by the model checking tool to be used. This is a challenging task where often ambiguous and vague requirements need to be formalized as precise statements in the formal specification language (such as temporal logic) of the model checking tool. This task interacts with modelling where the level of abstraction and the system/environment interface as well as component interfaces need to be designed to support requirement specification so that requirements can be formalized as statements on these interfaces. Given the system model and a specification formalizing a given requirement statement, running the model checking tool on them is often the most straightforward part. Interpreting the results produced by the tool is in principle quite simple if the created model supports traceability, i.e., supports interpreting the behaviours of the model as behaviours of the original design. However, counter example executions demonstrating that a specification is violated by the system design can be quite complicated and further tool support may be needed to illustrate the underlying erroneous system behaviour.

3 MODEL CHECKING IN I&C DESIGN

The basic idea of model checking was introduced in the early 1980s but only in the 1990s novel symbolic model checking techniques led to new tools such as SMV and a breakthrough in scalability /6/. Nowadays, for example all major microprocessor manufacturers use model checking techniques to verify their processor designs. Other areas where model-checking has been widely applied include verification of data communications protocols with tools like Spin /9/ and real-time controllers with tools like UPPAAL /3/. Also model checking of real source code has become feasible with new techniques. A good example of success in this area is the SLAM project /1/ at Microsoft where a model checker for C programs has been developed for verifying the use of locking primitives of Windows device drivers and adopted in the Windows driver development kit /2/.

3.1 Practical experiences

The applicability of model checking for verifying various kinds of I&C designs has been studied by analysing several industrial cases with the NuSMV and UPPAAL model checkers. The analysed cases include an emergency cooling system of a nuclear reactor /12,13/, an industrial arc protection system /10,11,12,13,15/, a changeover switching unit for a busbar, a stepwise shutdown system /4,5/, and an embedded control software of an uninterruptible power supply (UPS) /8/. The last two cases are introduced below.

The stepwise shutdown system is used for the stepwise control of an industrial process towards the normal operating state in case of disturbances. The purpose of the system is to reduce the possibility that the process enters an undesired state where the more complicated actual shutdown function is required. The system design is composed of logic gates and a timed loop to make the control stepwise, i.e., the process is driven towards a safer state for a certain period after which it waits another period and continues this cycle as long as necessary. The

performance and applicability of two model checking tools, NuSMV and UPPAAL, were analysed and compared /4,5/. Both tools were successfully employed for verifying basic safety properties of the system and were able to reveal the same hidden design errors. Model checking times of the NuSMV model ranged between 0.3s and 30s depending on the used time step (10-1000ms) and the verified property. The size of the state space of the most complicated scenario modelled with NuSMV was 10^{18}. The computation times of the UPPAAL model were between 9s and 20s, being without a few exceptions a bit longer than those of NuSMV.

In addition to verifying the correct behaviour of the design, the NuSMV tool was used to analyse the fulfilment of the single failure criteria with several different failure models. The failure scenarios were created by combining the following properties: the failures were detected or they remained undetected, failed input signals were given non-deterministic values or they kept their previous values, and input signals might fail or recover at any time step. The case study demonstrated that in addition to finding design errors, it is possible to determine the fulfilment of single failure criteria with model checking.

The applicability of model checking to timed embedded software was studied with the UPPAAL model checker in a case concerning the verification of control software of an Uninterruptible Power Supply (UPS) /8/. UPS devices provide backup power and protect the connected equipment in case of power disturbances. The control of the UPS device requires several time delays of different scale making the validation of the control software challenging. Several failure cases related to the operation of the UPS were investigated. The model of the system was verified against the system specifications obtained from the failure cases and model checking revealed that the control software of the UPS does not operate properly in two of the failure situations. The two errors found with model checking were related to timing. In addition to verifying single failure tolerance, the specifications were also verified against a model in which several failures can happen simultaneously but in that case most of the properties were not satisfied any longer. The model checking times for the single failure models ranged between 1s and 34s. The size of the state space varied depending on the verified property and the largest state space was about 205,000 states. The results of the case study indicate that model checking can be used for verifying the correct operation and finding errors from embedded control software.

3.2 Benefits and limitations

The case studies have clearly demonstrated the power of model checking in verification of I&C systems designs. The method is directly usable for verifying designs of safety I&C systems containing tens of inputs and rather complicated timing behaviour.

An advantage of the model checking method compared to traditional testing and simulation activities is that it can provide full coverage for verification (see Figure 1). This is realized in cases where the reachable system state space becomes too large for efficient exhaustive simulation or manual inspection. For example, if a combination circuit contains 60 inputs the number of different input combinations is $2^{60} = 1.2 \cdot 10^{18}$. Assuming there was a highly efficient simulator that could simulate 10^7 input combinations per second, it would take over 3600 years to go through all the input combinations. On the other hand, the model checking times of such a circuit range typically from a few seconds to a few minutes per verified property.

As simulation and testing are able to handle only one case at a time, model checking examines all the possible behaviours in one execution. Model checking enables extensive verification of both positive and negative properties, such as “All system executions lead to a certain system state” or “None of the system executions lead to a certain state”, which is not possible with testing or simulation.

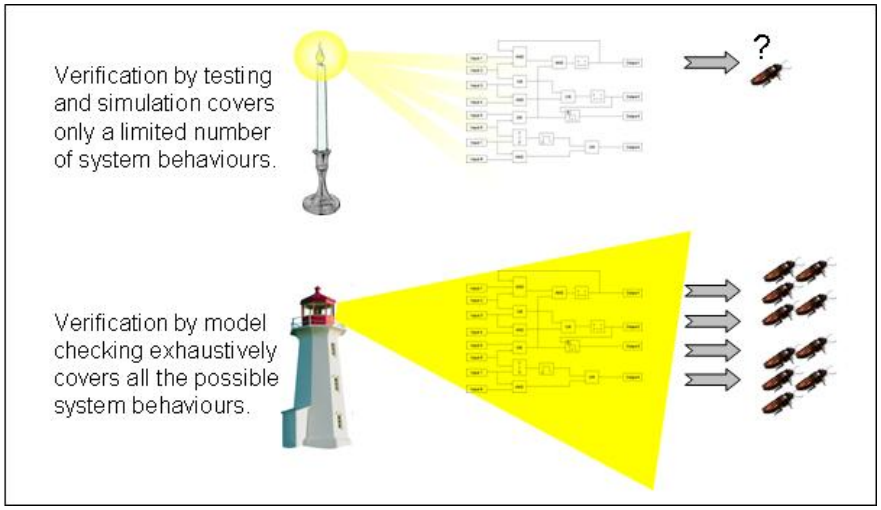

Figure 1. *Model checking vs. testing and simulation.*

The main benefits of model checking in the verification of digitalized safety logics are achieved at the design phase of the development process. It is possible with reasonable effort to model a system on an adequate level, formulate required safety properties in the specification language, and perform full verification of the properties. Once the model is made, it is rather easy to check different scenarios and see how small changes, for instance, in the input signals affect the behaviour of the system. The analysed case studies show how minor changes in the design may lead to unexpected errors that are hard to detect.

Despite the clear benefits and potential of the model checking methodology, it still has some limitations and unsolved problems. Logical circuits containing both complex timing aspects and a large number of input variables are challenging to model and verify. The number of inputs and internal states in the design determine the size of the state space. The complexity of the timing behaviour and the size of the state space of the model affect the model checking time. One limitation of the employed model checking tools is that they support only the basic mathematical operations and integer values. That creates challenges when modelling complex functions, such as PID controllers.

Requirements verified by model checking are typically originating from high level system documentation such as overall requirements specifications and functional descriptions. Those upper level requirements need to be broken into several more specific sub-requirements to reach the level of detail where model checking can be used. Verification of one upper level requirement is typically done by verifying several sub-requirements that are divided into several temporal logic statements verified by model checking. This is how the overall verification can be constructed. It is not an easy task because the coverage of each temporal logic statement has to be considered carefully to make sure that all the aspects of the original upper level requirement have been taken into account.

To make model checking feasible, the system model and the environment model are typically abstracted views of the real system still covering the essential behaviour. The real challenge is to abstract the system model to sufficient level because the abstraction is always a trade-off between accuracy and performance.

It has to be remembered that model checking always concerns a model made of the actual system. Hence, when choosing an appropriate abstraction of a system to be modelled one should be careful to cover all the relevant system behaviour because only then the correctness of a positive model checking result can be guaranteed. On the other hand, a negative model checking result indicating a counter example execution violating a specification can often be checked against the actual system. Hence, model checking can be beneficial for generating interesting test cases concerning improbable and unexpected situations of the actual system that a test designer would not normally consider. In case of verifying a design model, the design error implied by a negative model checking result can normally be found by following the given counter example execution manually. Experience has shown that finding the same error by performing only manual inspections is difficult and that computerized tools are needed.

4 CONCLUSIONS

Modern digitalized I&C systems consist of complicated control tasks that are challenging to verify. Traditional verification methods like testing, simulation, and deductive verification have their advantages but none of them alone is suitable for exhaustive verification with reasonable effort. Model checking is a formal verification method enabling complete verification of a system design model. The task is to create a model of the design and its relevant environment and use them to determine whether a given requirement is satisfied by its specification, which is checked against all executions of the system model.

In this paper, basic model checking methodology was introduced, examples of industrial cases where it has been successfully used were given and its benefits and limitations, and applicability to verifying safety logic designs were discussed. Some of the biggest advantages of model checking are its ability to handle large state spaces and provide full coverage for verification by examining all the possible behaviours of the system model. Creating and modifying models for model checkers is rather straightforward and fast compared to making simulation models or running tests in a real system environment. The main benefits of model checking in the verification of digitalized safety logics are achieved at the design phase of the development process. It helps in defining interesting test cases for the actual system testing and often reveals design errors that easily could be left unfound with only traditional verification methods or be found much later in the development cycle causing extra effort and cost.

In spite of the increased computing power and sophisticated model checking algorithms, the state explosion problem is still present and limits the use of model checking. Systems having highly complicated timing behaviour or complex control functions are challenging to verify. Models are always abstracted views on the real system. The real challenge is to find a suitable level of details to be included in the model to end up to a sufficient trade-off between accuracy of the model and the performance of the model checker. Making the model is just the start of the verification process. Finding the requirements for the system and deriving the temporal logic statements to be given to the model checker are challenging phases of verification. In case the model checker finds an error it gives a counter example demonstrating the sequence of state transitions violating the checked property. Analysing this counter example and finding the cause of the error is challenging and may be time consuming. Computerized tools for counter example analysis are needed.

The purpose of model checking is not to replace testing and simulation but to complement and support them. Each of the methods has its own benefits and limitations and there is a proper role for all of them in the overall system development and verification process.

5 REFERENCES

1. Ball, T. and Rajamani, S. The SLAM project: debugging system software via static analysis. In Conference Record of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002), pages 1-3. Association for Computing Machinery, 2002.

2. Ball, T., Cook, B., Levin, V., and Rajamani, S. SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft. In Proceedings of the 4th International Conference on Integrated Formal Methods (IFM 2004), volume 2999 of Lecture Notes in Computer Science, pages 1-20. Springer-Verlag, 2004.

3. Behrmann, G., David, A., Larsen, K., Håkansson, J., Pettersson, P., Yi, W., and Hendriks, M. UPPAAL 4.0. In the Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), pages 125-126. IEEE Computer Society, 2006.

4. Björkman, K., Frits, J., Valkonen, J., Lahtinen, J., Heljanko, K., Niemelä, I., and Hämäläinen, J.J. Verification of safety logic designs by model checking. In Proceedings of the Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies NPIC&HMIT 2009, Knoxville, Tennessee, April 2009. http://www.vtt.fi/inf/julkaisut/muut/2009/Bjorkman_etal_NPIC_HMIT_2009.pdf

5. Björkman, K., Frits, J., Valkonen, J., Heljanko, K., and Niemelä, I. Model-based analysis of a stepwise shutdown logic. VTT Working Papers 115, VTT Technical Research Centre of Finland, Espoo, 2009. http://www.vtt.fi/inf/pdf/workingpapers/2009/W115.pdf

6. Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L. Symbolic Model Checking: 10^20 States and Beyond. Information and Computation 98 (2): 142-170 (1992).

7. Clarke, E., Grumberg, O., and Peled, D. Model Checking. The MIT Press, 1999.

8. Frits, J. Model Checking Embedded Control Software. Research Report TKK-ICS-R28, Aalto University School of Science and Technology, Department of Information and Computer Science, Espoo, Finland, March 2010.

9. Holzmann, G. The model checker Spin. IEEE Transactions on Software Engineering, 23(5):279-295, 1997.

10. Koskimies, M., Applying model checking to analysing safety instrumented systems. Research Report TKK-ICS-R5, Helsinki University of Technology, Department of Information and Computer Science, Espoo, Finland, June 2008. http://lib.tkk.fi/Reports/2008/isbn9789512294787.pdf

11. Lahtinen, J., Model checking timed safety instrumented systems. Research Report TKK-ICS-R3, Helsinki University of Technology, Department of Information and Computer Science, Espoo, Finland, June 2008.. http://lib.tkk.fi/Reports/2008/isbn9789512294459.pdf

12. Valkonen, J., Pettersson, V., Björkman, K., Holmberg, J.-E., Koskimies, M., Heljanko, K., Niemelä, I. Model-Based Analysis of an Arc Protection and an Emergency Cooling System. VTT Working Papers 93, VTT, Espoo, 2008, http://www.vtt.fi/inf/pdf/workingpapers/2008/W93.pdf

13. Valkonen, J., Koskimies, M., Pettersson, V., Heljanko, K., Holmberg, J.-E., Niemelä, I., Hämäläinen, J., Formal Verification of Safety I&C System Designs: Two Nuclear Power Plant Related Applications. Enlarged Halden Programme Group Meeting. Proc. Man-Technology-Organisation Session. Loen, Norway, 18 - 23 May, 2008

14. Valkonen, J., Karanta, I., Koskimies, M., Heljanko, K., Niemelä, I., Sheridan, D., Bloomfield, R.E., NPP Safety Automation Systems Analysis – State of the Art. VTT Working Papers 94, VTT, Espoo, 2008, http://www.vtt.fi/inf/pdf/workingpapers/2008/W94.pdf

15. Valkonen, J., Koskimies, M., Björkman, K., Heljanko, K., Niemelä, I., and Hämäläinen, J.J., Formal verification of safety automation logic designs. In Automaatio XVIII 2009 Seminaari, 2009. http://www.vtt.fi/inf/julkaisut/muut/2009/Formal_Verification_of_Safety_Automation_Logic_Designs_paper.pdf