Real-Time Verification of STATEMATE Designs

Udo Brockmeyer and Gunnar Wittich *

OFFIS, Escherweg 2, 26121 Oldenburg, Germany
{Brockmeyer,Wittich}@OFFIS.Uni-Oldenburg.de

Abstract. This paper presents a toolset for real-time verification of STATEMATE[1] designs. STATEMATE is a widely used design tool for embedded control applications. In our approach designs including all timing information are translated into untimed finite state machines (FSMs) which are verified by symbolic model-checking. Real-time requirements are expressed by TCTL formulae interpreted over discrete time. A reduction from TCTL model-checking to CTL model-checking is implemented in order to use a CTL model-checker for the verification task. Some experimental results of the toolset are given.

1 Introduction

In this paper we present a toolset for real-time verification of STATEMATE designs [8–10]. STATEMATE is a widely used graphical specification tool for embedded control applications. The STATEMATE toolset captures the phases of specification, analysis, design and documentation of real-time systems. To cope with the complexity of real life applications, a system under development may be described graphically from three different viewpoints within STATEMATE. They cover structural (*Module-Charts*), functional (*Activity-Charts*) and behavioral (*Statecharts* [7]) aspects of a system.

For the real-time verification of STATEMATE designs we use the technique of model-checking. Model-checking is an automatic method for proving that a given implementation of a design meets its requirement specification represented by a temporal logic formula. As specification language, we use TCTL as introduced in [2] restricted to a discrete time domain. Our TCTL model-checking procedure aims at reuse of an industrial CTL model-checker [6] and contains two major new components: first a translation of STATEMATE designs into untimed FSMs and second an embedding of the discrete time TCTL model-checking problem into CTL model-checking.

The semantical foundation of our translation from STATEMATE designs into untimed FSMs [3], as required by the model-checker [6], can be found in [5]. Our

* Part of this work has been funded by the Commission of the European Communities under the ESPRIT project 20897, SACRES and the German BMBF project KORSYS, grant number 01-IS-519-E-0

[1] STATEMATE is a registered trademark of i-Logix Inc.

toolset supports real-time verification for the synchronous (step) semantics as well as for the asynchronous (super-step) semantics provided by the STATEMATE simulator and therefore for both of the semantics given in [9]. Furthermore, in addition to almost the complete language of Statecharts, the language of Activity-Charts is also covered by our toolset.

In this tool-paper we demonstrate the feasibility of our approach to real-time model-checking on some case studies. Two of them are industrial sized applications provided by our project partners. The first one originates from the SACRES project and is provided by British Aerospace. It is a Storage Management System of an aircraft. The second one was provided by ESG[2] in the KORSYS project. This case study is a Helicopter Monitoring System which monitors engine and fuel parameters.

2 Modeling Real-Time Features of STATEMATE

STATEMATE distinguishes between the synchronous simulation semantics (*step semantics*) and the asynchronous simulation semantics (*super-step semantics*). In the step semantics, each step of a design corresponds to exactly one discrete time unit, time increases uniformly and the environment can influence the valuation of variables at every step. In contrast, in the super-step semantics a system performs a chain of internal steps until a stable state (no more internal steps are possible) is reached. Only in a stable state time progresses and the system accepts new stimuli.

In order to perform real-time verification of STATEMATE, designs have to be translated into a format interpretable by the model-checker. Our toolset translates designs in two steps. A STATEMATE design is first translated into an intermediate language called SMI (STATEMATE *InterMediate*). We defined SMI as a language for the translation of high-level formalisms into FSMs[3]. In a second phase, the generated SMI code is translated into a FSM for model-checking.

SMI is a simple imperative programming language containing concepts to model hierarchy, parallelism, and nondeterminism of STATEMATE designs. The data-types and expression language of SMI are powerful enough to cover a wide range of STATEMATE types. The cyclic behavior of a STATEMATE design is represented as a non-terminating loop in SMI code. One execution of this loop corresponds to exactly one step of the design. In SMI all control information, all variables and all events of the STATEMATE design are encoded by variables.

STATEMATE provides two ways to introduce explicit timing information into a Statechart which both relate events and actions to the discrete virtual simulation clock. The first alternative allows to trigger transitions by *timeout events*. The second alternative for introducing timing information into a Statechart allows to delay the execution of actions for some time units by a *scheduled action*. To cope with timing aspects of a design the translation process introduces clock variables

[2] Elekronik Systeme GmbH, Munich, Germany

[3] In other projects, we translate VHDL, a subclass of Petri-Nets, and a subclass of OCCAM into SMI

for timeout events and scheduled actions. All clocks are running synchronously. Because we require all time expressions to evaluate to a constant at compile time, finite domains for the clocks can be determined.

Because after the translation of a STATEMATE design into SMI all necessary clocks are represented by a finite number of bounded variables, untimed FSMs can be generated out of the code. The construction is such that one step of the FSM corresponds to one execution of the complete loop-body of the SMI code. Thus, in step semantics in each state of the FSM timers are increased by one. In super-step semantics, timers are increased only in certain states, while they remain unchanged in all other states.

3 Real-Time Model Checking

As specification logic we use TCTL as introduced in [2] interpreted over discrete time. Verification is performed by translating TCTL into CTL automatically and model-checking a suitable extended model against the resulting formulae with a slightly enhanced CTL model-checker.

To model-check a TCTL formula with a CTL model-checker, we transform the FSM by adding an additional specification clock. The upper bound of this clock is determined out of the given TCTL formula. This specification clock is incremented whenever time progresses. According to the selected semantics, these states are characterized by a time condition given as a SMI expression.

A similar reduction for a derivate of dense time TCTL is given in [11]. Unlike as in the approach in [11], where additional time transitions between transitions of the system are introduced, we can avoid this blow up by extending CTL (and thus the model-checker, too). Thus, we reduce the number of steps performed by the model-checker while doing its work significantly.

4 Experimental Results

In this section we present some experimental results obtained with our tools. Figure 1 gives a coarse overview of our toolset. STATEMATE designs are translated by STM2SMI into SMI code out of which FSMs are generated by the tool SMI2FSM. FSM2FSM-T serves to add the specification clock to a FSM. Finally, TCTL2CTL realizes the reduction from TCTL to CTL. The model-checker (MC) we use is the ROBDD [1] based assumption/commitment style CTL model-checker provided by our project partner SIEMENS [6].

Table 1 overviews the results for three examined case studies[4]. The TLC is the well known traffic light controller enhanced by timing information modeling the delay of changing the lights. The second example is a component of a Storage Management System (SMS) of an aircraft. This industrial sized application was provided by our project partner British Aerospace. Finally, we model-checked a Helicopter Monitoring System (HMS) which was provided by our project partner

[4] All results were evaluated on a Sun SPARC 20 running at 60 MHz

ESG. The second column contains the times needed for the translation from STATEMATE into SMI. For the TLC we chose super-step semantics, while the other two designs were translated for step semantics. The third column shows the times to generate FSMs. Column four and five are indicating the complexity of the studies. Finally, in the MC column, times for model-checking of relevant real-time properties on the given models are presented.

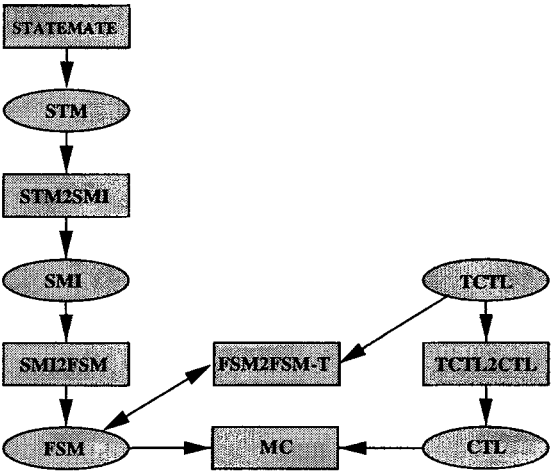

Fig. 1. The Toolset

Beyond these experiences with verifying moderately sized STATEMATE designs against TCTL formulae, we already have very encouraging results on verifying substantially larger STATEMATE designs against CTL formulae. Some of these results are presented in [3, 4]. There we have shown, that our tools are very powerful in generating FSMs and performing CTL model-checking. Industrial sized applications with several hundred state bits could be handled. These models already contain all clocks that model timeouts and scheduled actions of STATEMATE designs. Because for TCTL model-checking only the additional specification clock has to be added, we will apply our toolset on these designs, too, and we expect to be able to verify relevant real-time properties for them.

Model	stm2smi in s	smi2fsm in s	# of bits input/state	# of BDD nodes	MC in s
TLC	2.56	0.45	18/33	2485	12.1
SMS	4.82	6.41	13/53	3284	11.6
HMS	6.78	1.60	32/103	4195	87.4

Table 1. Experimental Results

5 Conclusions and Future Work

In this paper a toolset for real-time verification of STATEMATE designs against TCTL formulae has been presented and its usability on some case studies was demonstrated. Because of the complexity of STATEMATE, there are some rare used features not yet covered by the tools. Our future work is about closing this gap in order to support even these features. Also, we have a lot of ideas for optimizations that can be performed in order to generate smaller FSMs out of STATEMATE designs. Some of these ideas have already been implemented and results have been presented in [3]. Applying these optimizations, we expect to be able to verify real-time properties of much bigger designs in the near future.

Acknowledgment. We thank our project partners British Aerospace, ESG, SIEMENS and i-Logix for providing the tools, case studies and for discussions. Furthermore we thank Werner Damm and Martin Fränzle for helpful discussions.

References

1. S.B. Akers. Binary decision diagrams. In *Transactions on Computers* , No. 6 in Vol. C-27, pages 509-516, IEEE, 1978
2. R. Alur, C. Courcoubetis and D. Dill. Model-Checking for Real-Time Systems. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 414-425, Philadelphia, June 1990.
3. U. Brockmeyer and G. Wittich. Tamagotchis need not die – Verification of STATEMATE Designs. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, March 1998
4. W. Damm, U. Brockmeyer, H.J. Holberg, G. Wittich and M. Eckrich. Einsatz formaler Methoden zur Erhöhung der Sicherheit eingebetteter Systeme im KFZ. VDI/VW Gemeinschaftstagung, 1997
5. W. Damm, H. Hungar, B. Josko and A. Pnueli. A Compositional Real-Time Semantics of STATEMATE Designs. In Proceedings of COMPOS 97, edt. H. Langmaack and W.P. de Roever, Springer Verlag, to appear 1998
6. T. Filkorn, SIEMENS AG. Applications of Formal Verification in Industrial Automation and Telecommunication. In Proceedings, *Workshop on Formal Design of Safety Critical Embedded Systems,* April 1997
7. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming 8*, 1987.
8. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull–Trauring and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. In *IEEE Transactions on Software Engineering* , 16:403 – 414, 1990
9. D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. In *ACM transactions on software engineering and methodology, Vol 5 No 4* , 1996
10. D. Harel and M. Politi. Modeling Reactive Systems with Statecharts: The STATEMATE Approach. *i-LOGIX INC., Three Riverside Drive, Andover, MA 01810,* June 1996. Part No, D–1100–43
11. T. A. Henzinger and O. Kupferman. From Quantity to Quality. In *Proceedings of Hybrid and Real-Time Systems (HART'97)*, March 1997