SM2SMV - A Tool for Facilitating Dependable Software Requirements Analysis Using Model Checking

Huigang Li
Embry Riddle Aero. Univ.
lih@db.erau.edu

Andrew J. Kornecki
Embry Riddle Aero. Univ.
korn@db.erau.edu

David P. Gluch
Software Engineering Institute
dpg@sei.cmu.edu

1. Introduction

The complexity of dependable software and electronic systems is growing at a rapid pace. As systems become more and more complex, defect detection becomes a difficult problem. One of the available techniques to facilitate early defect detection within requirements specifications is model checking. The technique is based on building a finite model of a system and checking if the model possesses the desired/required properties [1].

One of the tools used in model checking is the Symbolic Model Verifier (SMV), developed by K. L. McMillan at Carnegie Mellon University [2]. Using the SMV tool consists of four major phases:

(1) creating a finite state machine model for the system;
(2) translating the state machine model into SMV input program;
(3) creating the expected properties and translating them into SMV program;
(4) running the SMV program and interpreting the results.

Currently, the phase (4) can be completed automatically by the available SMV tool. Phase (1) to phase (3) still need manual work.

In the past year, we carried out several experiments applying SMV to verify system requirements for a variety of dependable systems (e.g. a heart pacemaker, military avionics, etc). The experience shows that most of the work in phase (2) involves mechanical translation from the state machine graphical representation to SMV input program written in SMV language. The detailed syntax and grammar for SMV input language can be found in the SMV user manual [2][4].

Since phase (2) is a translation from a state machine diagram to SMV input code, it may be possible to do this with a software tool. The potential advantages of a graphically based automated code generator would be the time saved by an engineer and improved correctness of the SMV input program. The work described here is the first step in exploring these issues.

Recently, a research group in Germany did similar research on translating STATEMATE designs into SMI code. The model checker used in their research is a tool developed by SIEMENS [3].

2. Symbolic Model Verifier

The Symbolic Model Verifier (SMV) provides a mean of representing the system as a set of states and transitions. It allows for the specification of the required system properties in Computational Tree Logic (CTL) notation and automatically checks these properties against the state machine representation. Thus, SMV is a tool to check finite state systems against their specifications. SMV uses an Ordered Binary Decision Diagram (OBDD) based symbolic model checking algorithm. In SMV models are checked against the specified property for a confirmation or a non-confirmation with counter example information [1].

A SMV program consists of a set of modules, including a main module. The relationship among these modules can be hierarchical or parallel. The common structure for a simple one-level module has four parts: module header, variable declaration, assignments, and specification. The module name is given in the module header section, the first part of the code. The second part of the SMV code includes variable declarations, i.e. the states and events. In the third part, the transitions of the state machine are presented in SMV language. The user claims, representing the required properties of the system, are included in the fourth part. For more information on SMV input language, please refer to SMV user manual [2][4].

3. Graphical SM2SMV Prototype

A prototype graphical SMV code generating tool SM2SMV (State Machine To Symbolic Model Verifier) was developed to demonstrate the feasibility of automatic conversion of a state machine diagrams into SMV input code. The tool was implemented in Java using a rapid-prototype software development method.

The tool contains the following two major functions: (1) **State Machine Drawing and Editing**: the user is able to use a mouse to click and draw a state machine, with a set of states and transitions representing the software requirements, in a panel. The user can modify and update this diagram using a mouse.
(2) **Code Generating**: The system creates SMV code representing the state machine. Once the state machine model is changed, the system allows the user to update the SMV code to reflect the changes.

The tool has a simple graphical user interface. Figure 1 presents a sample interface of this tool.

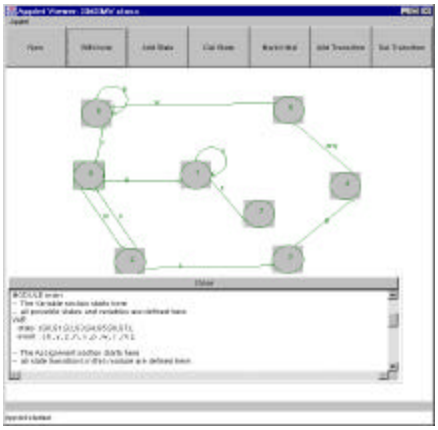

Figure 1: SM2SMV Graphical User Interface

For simplicity of design and implementation the current version of the prototype only allows a limited number of states and events. States are represented as numbers (0-9) and transitions are represented by English alphabetic characters (a-z). SMV input code is shown in a pop up window (see Figure 1). If the state machine diagram is changed, an equivalent SMV representation is automatically updated by clicking the *SMVcode* button.

4. Conclusions

A prototype tool for facilitating model based verification of software requirements for dependable systems, using the Symbolic Model Verifier technique, was developed to demonstrate the concept of bringing together a graphical state machine and a formal SMV code representation.

The testing and experimentation with the current version of this protocol tool has shown that it met the initial requirements. The user can create and modify a single one-level state machine model using the graphical user interface. The tool enables the user to create a syntactically correct SMV input program based upon the state machine model.

A rapid development prototype method was used to implement this tool. In order to reduce the coding time, some Java code from outside sources was used. The code needs to be optimized to improve display speed and memory usage. For example, the response for refreshing a screen is relatively slow. It usually takes 2 to 5 seconds to update a picture on the screen. The actual results varied according to the size of available memory and the CPU speed.

Although the current tool has met initial goals, there is additional work required to establish feasibility and to improve the prototype. These include:

(a) Supporting state charts: The tool supports a one level state machine instead of a hierarchical state chart;
(b) Increasing the number of states and events: Enhancements include lifting the limitation on the number of states and events, and modifying the transition to enable Boolean functions of events;
(c) Adding CTL claims: The CTL claims (desired / required properties) are not generated;
(d) Improving performance: The response time of updating a display screen is relatively slow (2 to 5 seconds) and the memory usage is not optimized.

This prototype tool has shown that it is possible to generate a SMV representation from a simple state machine diagram. Additional work will be needed to demonstrate that the approach can scale to more complex systems while providing acceptable system response times, time savings, and improved correctness of the SMV code.

5. References

[1] Gluch, D., Brockway, J. “An Introduction to Software Engineering Practices Using Model-Based Verification” Software Engineering Institute, CMU, Pittsburgh, Pa. April 1999.

[2] McMillan K. L. “Symbolic Model Checking” Kluwer Academic Publishers, Massachusetts, 1993

[3] Brockmeyer U., Wittich G. “Real-Time Verification of STATEMATE Designs”, Proceedings of Computer Aided Verification, CAV’ 98, Lecture Notes in Computer Science 1427, Springer-Verlag, 1998, pp. 537-541

[4] CMU and University of Milano, NUSMV Manual, http://afrodite.itc.it:1024/~nusmv, 1999