

A Critical Analysis of Model-Based Formal Verification Efforts within the Automotive Industry

Arun Chakrapani Rao and Ross McMurran

International Automotive Research Centre (IARC), The University of Warwick

R. Peter Jones

School of Engineering, The University of Warwick

Copyright © 2008 SAE International

ABSTRACT

There are a number of software-controlled features in today's automotive vehicles to meet the increasing number of requirements for comfort, safety, infotainment and reduced emissions. To meet the growing demands from such features, the software content is not only increasing rapidly, but also becoming increasingly distributed within the Electronic Control Units (ECUs), leading to the possibility of unwanted interactions and consequent loss of reliability. Therefore, the automotive software-based features have to be designed and verified using sophisticated tools and techniques.

Formal methods-based techniques and tools have been used on various industrial designs over the past 6 years by one of the authors in development and applied research projects, collaborating with a number of automotive companies. The challenges faced in large projects are discussed in this context. A large number of industrial users have recounted their own experiences in user conferences and workshops, which will be summarized. The results of a short survey of real users will also be discussed in this context.

INTRODUCTION

In this section, background information on automotive software development and formal methods will be provided.

The features of today's automotive vehicles are increasingly being controlled by software. The size of the software can be as large as about 100 Mb, and this could be distributed across several ECUs, which could number close to a hundred! Therefore, it is becoming increasingly necessary to develop this software by using precise languages and methods.

"*Formal methods*" is a general term used to describe such precise languages and methods. It covers various mathematically precise languages such as Z, as well as

mathematical techniques such as model checking [1-3] to enable exhaustive testing of a model against requirements. Around a hundred different formal languages, techniques and tools are listed in [2].

For greater accessibility of formal methods, various research and development projects have focused on a wide variety of areas; at one end of this spectrum there are efforts to develop new theory as well as work relating to enhancing the accessibility of an existing heavy mathematical language by developing a graphical equivalent with a supporting visual framework for systems development, such as in [4]; at the other end there are efforts such as supporting industry-strength modeling languages for designing systems with formal verification tools in such a way that the user is shielded from any heavy mathematics.

Many industries such as aerospace and electronic circuit designers have managed to achieve excellent success using formal verification. These efforts led to the successful deployment of formal verification in Intel's design projects in 1995 [3]. Various other successes were achieved more than a decade ago [5]. Many designers of electronic circuits and communication protocols now use these methods routinely.

Over the past 5 years, although many factors have contributed to the increasing complexity of automotive software and, in parallel, formal verification technology has improved a lot in many respects and continues to achieve successes within many other domains, relatively few studies have been done on model-based formal verification within the automotive industry. This can be attributed to the nature of the vehicle development lifecycle and associated day-to-day pressures, engineers requiring significant knowledge in formal methods, and the use of formal verification software tools requiring specialized skills, among others. The aim of this paper is to take a look at a number of studies that have been done in recent years and report some key points learned from them.

LEARNING FROM PAST AND RECENT INDUSTRIAL STUDIES

SOME PAST INDUSTRIAL STUDIES

In a study done by Scott Ranville of commercially available tools that apply formal methods to a MATLAB/Simulink/Stateflow model [6], the case studies involve trivial Stateflow models used to demonstrate the features of the then commercially available tools. No large scale studies from the industry are included here, but it is concluded that there is a significant potential for a rate of return that can offset the cost and time involved in using such tools. At the 16th IEEE International Conference on Automated Software Engineering [7], Ford and Reactive Systems Inc. [8] reported their joint investigation of a verification tool on a piece of software for a powertrain controller. At the time of the investigation, viz., 2001, the engineers involved had to manually translate the corresponding Simulink/Stateflow model into the input language of the model checker. Despite the efforts involved in manual translation, the work concluded that they were able to identify problems like non-determinism and dead code. In another case study [9], some preliminary results on the use of EmbeddedValidator at DaimlerChrysler were reported concluding that they found the tool usable for practical automotive applications. The EmbeddedValidator was preceded by a prototype version called the Stateflow Verification Environment Prototype, developed by OFFIS [10] in collaboration with DaimlerChrysler and dSPACE. Various features of the modeling language were supported by using several industrial models from DaimlerChrysler. This version continues to be used in automotive companies.

Despite some of the efforts mentioned above, the application of formal verification within the automotive industry over the past 6 years reveals a number of challenges. A number of such case studies have been summarized here, indicating some key knowledge that can be learnt from them.

RECENT INDUSTRIAL STUDIES

In this section, what has been learned from many recent industrial studies done within IARC [11-13] and other work reported at workshops such as [14-15] is summarized. While the details of the models are not disclosed, the learning from all these systems has been assembled without highlighting any particular challenge faced to a specific study. The idea is that one can get an overall idea of the types of challenges that have been faced.

Table 1 lists some recent industrial systems studied along with the modeling languages used. Specific tools are not mentioned against the studies, as different tools have different features and capabilities at any point in time and the intention and the scope of this paper is not to compare the features of verification tools. However, some of the

available tools that have been used in the context of one or more of the industrial studies are listed separately.

Organization	Modeling Language	System
IARC (for Jaguar LandRover)	Simulink/Stateflow [17]	Hood Controller, Air Suspension
Bosch	UPPAAL [18], Simulink/Stateflow	A passive safety control unit for deploying restraint devices such as air bags
CRF	Simulink/Stateflow	Arbitration logic unit for a fault-tolerant system architecture for actuator control
DaimlerChrysler	Simulink/Stateflow	Advanced Automotive Assistance System
PSA	SCB [19], Statemate [20]	Unstated / unknown
ZF	Simulink/Stateflow	Unstated / unknown
MAN Nutzfahrzeuge	Simulink/Stateflow	Automotive drivetrain software for trucks – control software for engine, brakes, cruise control, adaptive cruise control
Airbus	Simulink/Stateflow	Airbus Landing Gear Extension/Retraction System
Ford, USA	Simulink/Stateflow	Software for a powertrain controller
Ford, Research and Advanced Engineering., Aachen	Simulink/Stateflow	A system for engine stop-start strategy

Table 1: Automotive Industry Studies in Formal Verification

Industrial systems studied

It can be seen from Table 1 that a wide variety of automotive electronic systems have been studied over the last few years, and these include body, chassis, powertrain, driver assistance and fault-tolerant systems for actuator control. Also included in this table is a recent study conducted by Airbus, UK.

Tool support for modeling languages

Commercial tools that support modeling languages have been listed in Table 2. Most of these have been used with the industrial studies listed in Table 1.

Challenges faced and some related solutions

In this section, various challenges that have been faced during the application of formal verification technology on industrial software-based systems are described.

Modeling Language	Formal Verification Tools
Statemate	Statemate ModelChecker and ModelCertifier [20]
Simulink/ Stateflow	Reactis [8], EmbeddedValidator [16], SCB Blockset [19], Simulink DesignVerifier [21]
SCADE	SCADE Design Verifier [22, 23]
UPPAAL (timed automata, temporal logic)	UPPAAL

Table 2: Tool support for modeling languages

Requirements for verification

The process of identifying suitable requirements that can be verified is a challenge. If formal verification is not planned for from the beginning of the project, it becomes harder later to look into requirements specifications and other documents for this purpose. Some tools such as described in [16] offer specification patterns for help in the formalization of requirements. These can be used in the early stages of the system development process to record requirements better, thereby making the process of identifying requirements for formal verification much easier.

Model preparation

Models have to be designed in such a way that they are suitable for verification. This would also influence the choice of the modeling language because suitable verification tools have to be available to support them.

For some modeling languages, there might be a good verification tool, but there may be lack of support from a good production code generator. Designing multiple models in different languages for different purposes is impractical, not only because it requires additional special skills and resources, but also ensuring that these multiple models are equivalent becomes another challenge. Once the language and the tools are chosen, any existing models would usually have to be adapted for several reasons. Most models not designed for verification usually lack additional information regarding model variables, such as their range values. However, with some effort, this process is usually simplified.

Modeling style

The reason why guidelines are needed is because they determine if a model can be verified or not. The complexity of a model is a function of several factors, such as the number of variables, their types and any information on their ranges, the type of features of the modeling language and the way in which they have been used.

$$\text{mdl_complexity} = f(\text{nvar}, \text{rang}, \text{sts}, \text{feat}, \text{sty})$$

where nvar = number of variables; rang = ranges of those variables; sts = number of graphical states used; feat = features of the language used; and sty = style of modeling.

There are some published guidelines such as MAAB guidelines [24], several internal company guidelines and some publications [25]. They relate to guidelines such as conventions for identifier names, orientation of data flows such as left to right, conditions and actions not being put on the same transition and how to make a model suitable for production code generation. In this section, some key modeling guidelines will be indicated with reference to the challenges they pose.

1. Guidelines for recording model data - For any chosen language and modeling tool, companies must detail exactly where and how detailed information regarding model variables (such as ranges and scaling values) should be recorded. In the absence of this, a lot of unnecessary difficulty will be introduced into the formal verification process.
2. Guidelines regarding counter and timer variables - Certain elements within the model such as timers and counters can lead to very large verification times. This can be overcome by choosing arbitrarily smaller values for them, especially for the sake of verifying some functional requirements. For this purpose, the model must have the possibility of customizing these counter and timer values from outside the model. In this manner, a model can be used for several different purposes such as simulation, verification and production code generation much more easily.