Week 3 Assignment

CPSC 59700, Spring 2017

Richard Givens
Student, College of Arts and Sciences
Lewis University
Romeoville, IL, USA

***Abstract*— This document is a review of two scholarly research papers, Introduction to Embedded Software Verification [1], and Comparison of Model Checking Tools for Information Systems [2].**

Keywords—Model Checking, Formal Verification, Information Systems, Logic.

I. Concepts or new knowledge presented

The presented papers, [1] and [2], discuss the methods, tools, and logic behind various formal verification methods involving embedded systems or the more standard information systems found throughout businesses. It is apparent throughout both papers, and in other research on these topics, that there are two critical concepts:

A. The Importance of Diversification

The main theme of these papers suggests Information Professionals of all levels should use a wealth of tools and methods, in fact, every tool and method at their disposal, when undertaking any development project. Limiting oneself to a single tool, or one set of tools, inhibits the ability to test, and creates an environment where an error in programming may remain unnoticed. Each of the presented tools has specific strengths and weaknesses, and while some cover many of the testing requirements for any system, none will cover every requirement. As modeling and formal verification of the program or system expands to test further preconditions and postconditions, the requirement of diverse tools grows.

B. Logic is a Fundamental Skill

Though it should be apparent to anyone in the field of Information Technology, the disciplines of Logic and Critical Thought are fundamental skills required to efficiently and accurately determine the results, intent, or design of any Information System, application, or program. The formal study of Logic, as well as continued learning and practice of the discipline, is paramount.

All microprocessor controlled technology operates with two numbers, either 0, or 1. As such, the logic behind their operation and capabilities is rather like an ON/OFF switch. Even something as simple as designing a formula in Microsoft Office Excel requires the ability to think through the possible steps of the formula, and write an elementary code that makes sense.

II. Tools Presented

In [1], Thomas Reinbacher discusses Theorem Proving and Model Checking. Theorem Proving is the complex method by which and experienced User tests the preconditions, inputs, and postconditions of a program or system. Starting at State 0, the User provides the necessary inputs for the program or function call, and then observes the postconditions or states to determine if the testing is successful. As such, the User must be experienced and knowledgeable enough to "step through" the process.

In Model Checking, temporal logic is applied through methods such as Computation Tree Logic and Linear Temporal Logic, but related but incomparable. The entire model is explored as a finite state machine, with the goal of verifying a property. However, these models may quickly grow to be unmanageable, as such, this method is often limited to smaller subsystem modeling.

Reference [2] provides more detail regarding the tools used in the use case contained therein. It provides a basic introduction of each tool or component of a tool used.

A. Specific Model Checking Tools

a) SPIN: one of the first model checkers, developed in 1980.

b) NuSMV: based on the Symbolic Model Verified, NuSMV is able to check both CTL and LTL formulae.

c) FDR2: an explicit state model checker that utilizes process algebra, and is able to support basic data types.

d) CADP:a modular set of tools.

e) Alloy a symbolic model checker using first order logic with relatins as its only terms.

f) PROB a modeling and animation tool with support for several modeling methods or languages.

B. *Advantages and Disadvantages in the Use Case*

The team in [2] discusses the advantages and disadvantages of each tool used to model the use case. Among the disadvantages mentioned, NuSMV was named for its inability of abstraction, requiring each transaction to be hardcoded into the model as separate instances or entities of the model checking program. PROB was mentioned multiple times as having support for certain features. Though not always intuitively easy, it stood out as having the most support for the requirements of the use case.

III. Area of Applications Covered

Though the use cases in both papers appear to be dramatically different, they are in actuality similar in their base concepts to one another. Reinbacher provides the example of an automated coffee machine, while the team comparing various model checking tools provides the use case of a library information/reservation system. Both tools are similar in that they require a number of steps, or switches, to accomplish their task, though the library system is of a higher complexity. Both use cases require the use of IF/THEN and TRUE/FALSE conditions.

Normally, when one thinks of formal verification of hardware or software, one imagines enormously complex information systems. Yet these papers reveal that formal verification and model checking is important, no matter the device, software or system in question. After all, a calculator that performs division when it should perform addition is of little value, as is a digital clock which cannot keep accurate time. As with the coffee maker described in [1], a simple home or office appliance is as prone to error as much as any other device containing some form of microprocessor.

IV. Applications of Concepts Within the Scope of Professional Activities.

VisionTek develops no software, nor produces hardware, therefore applications for these tools within the scope of employment is limited. However, within the scope of a career in Information Technology, one possible application of these tools is the thorough testing of an access control application. Just as there are multiple tools none of which cover every possible need, truly robust testing will require multiple tools, each testing a portion of the application's model or models.

One area of the application which will be relatively easy to model and test will be the determination of whether an individual is authorized or unauthorized to access portions of the system. To reframe, consider the case of an employee or visitor using an access badge. Though many other subsystems within the access control application may come in to play, such as cameras and facial recognition technology, the simple act of using an access control card or badge is a simple test of Boolean Logic, as demonstrated below:

The person wishing access through the control point must have a badge AND the badge must be presented to the card reader or device, which then reads and transmits the data to an application appliance or device, containing or accessing the data list of authorized personnel.

The data transmitted from the badge is compared to the list of authorized personnel. IF the data on the card matches AT LEAST ONE member of the list of authorized personnel, THEN access is granted/authorized, and the application receives a Boolean TRUE or 1 to indicate access.
In the case of the data from the card transmitted does not match the list of authorized persons, then the application returns a Boolean FALSE or 0, indicating the attempted access is denied.

Once this simple TRUE/FALSE behavior is modeled, it represents the trigger for other events, such as the facial recognition technology previously mentioned. Using such technology, the 0 OR 1 test acts as a switch, prompting the system to either compare the person's face with a database of stored images, or capturing photographic/video documentation of an attempted unauthorized access. It is in the case of further steps or conditions within access control that additional modeling tools become necessary.

Another possible model checking scenario is that of a fire alarm/suppression system. As with other use cases, machine states leading up to the precondition, as well as postconditions which occur after the specific model, require separate testing. This paper will only discuss the case of a smoke alarm alerting emergency services.

In this use case, the precondition (State 0) of the model is the normal operation of the system, in which no alarms are triggered, nor emergency response required. The triggering event is the detection of smoke through any number of building smoke alarms, and the postcondition is the notification of emergency services. The model would test for the successful notification of emergency services through a Boolean value, like the access control use case, in which State 0 may be SMOKE = 0. By setting the Boolean value to SMOKE = 1, the postcondition is tested. A successful test results in passing the value of SMOKE to a child process, which may be a model itself.

Though any of the tools mentioned in the papers would suffice in testing these use cases, the use of Assembly Language (ASM) mentioned in [1] stands apart as the possibly easiest solution. As both use cases involve embedded systems, the use of ASM appears to offer the fullest featured, robust level of tests required of such devices. A test run of the source code, and subsequent examination of the CPU registers would provide the necessary feedback required to determine the success of the test. However, it must be stressed that such a test is of the specific component of the overall system, in this case Fire Detection and Access Control, rather than the whole of the system itself. As such, separate use cases for each possible event must be designed, and may grow in complexity as the number and type of devices changes. Furthermore, it is

also necessary to state that no program, system, device or process is guaranteed to be free from faults.

Formal Verification of either system, access control or fire detection/suppression represents a critical requirement, with consequences for failing to adequately test possibly resulting in the loss of property at best, and at worst the potential loss of life.

V. CONCLUSION

Formal Verification and Model Checking of Information Systems reduces the amount of runtime errors experienced by users of the system. Without a robust testing method, in place during development and prior to final release, developers risk the chance of the system or systems behaving in unforeseen ways, malfunctioning, or providing false results to inputs. The cost to debug, and patch these errors may be considerable, and at times may eliminate any advantage or cost savings the Information System was intended to incur.

Information Professionals at all levels, from the entry level analyst to the experienced designers and developers, must rely on a diverse set of tools to accurately and efficiently test their systems for faults. Over reliance on one tool or a few tools increases the chance of missing some critical flaw, or inhibits the development of the project in such a way that it compensates for the lack of appropriate tools.

Finally, Information Professionals must also develop, understand, and use logic in both the design and analysis of any system. Understanding the basic flow of a system or program requires the use of logic, as the computer or embedded system itself ultimately uses only pure logic in its operation.

However, any system, model checker, program or device is only as good as the team designing it, and the team operating it. Deficiencies in knowledge and ability may result in unforeseen results during the model checking process, and may add a layer of complexity.

REFERENCES

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] T. Reinhacker, “Introduction to Embedded Software Verification,” 2008

[2] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, and M. Ouenzar, “Comparison of Model Checking Tools for Information Systems,” June 16, 2010