Applications of Model Checking

A Review of Literatures on Model Checking

John Hornik
Assignment 3
Research in Computer Science
Lewis University
Romeoville, IL, USA
johnphornik@lewisu.edu

***Abstract*—This paper reviews concepts presented in Introduction to *Embedded Software Verification* [1] and *Comparison of Model Checking Tools for Information Systems* [2]. It begins by going over some of the main concepts covered in the two works. It then recounts the model checking tools which are presented for embedded software verification as well as information systems verification. The paper concludes with an overview of the applications presented in the two papers as well as a new, proposed application in a manufacturing environment. Each section is broken up into two sub-sections: IESV presents ideas from [1] while CMCTIS presents ideas covered in [2]. Section IV contains an additional sub-section which introduces the proposed application.**

Keywords—embedded systems, information systems, software verification, software validation, model checking

I. Introduction

In the realm of software development, it is very important to assure that a software system fully satisfies all of its expectations. Essentially, there are two main processes in quality assurance or quality control which are implemented in order to make sure that this software fulfills these specifications: verification and validation. Validation is generally exemplified with the question "Are we building the correct software?" while verification can be epitomized by the question "Are we building the software correctly?" These two questions are explored in Comparison of Model Checking Tools for Information Systems [2] and Introduction to Embedded Software Verification [1], respectively. This paper focuses on presenting the concepts of software verification and validation utilizing model checking tools which are presented in these two literatures.

II. Concepts

A. IESV

The concepts which are presented in [1] revolve around embedded systems and the implementation of model based formal verification for verifying these systems. The author discusses past failures to utilize these verification methods due to previous model checking tools being limited to handling smaller amounts of code and embedded systems being too complex to generate reliable assumptions for verification. These days, however, research has allowed this type of verification to be implemented into embedded systems. The discussion on past methods examines issues with verification, debugging, and testing. Verification is essentially a process which is used to prove that a program meets its specifications. Debugging is the process of diagnosing errors in a given program and correcting these errors so that the program meets its specifications. Testing is the process of examining a program in order to either verify that it meets its specifications or to identify differences between expected results versus actual results. IT is noted that with today's systems, these processes are in a way obsolete due to the complexity of designs. Because of this, other methods, such as model checking, are highly sought after.

B. CMCTIS

The concepts presented in [2] relate to the formal validation of information systems using model checking. Six models are presented and compared: Alloy, cadp, fdr2, NuSMV, ProB, and Spin. These model checkers represent the main classes of model checkers which are listed as explicit state, symbolic, bounded and constraint satisfaction. They are presented and compared within a case study example of a library system. The author's main focus is to answer three questions:

1) Is the modeling language of the model checker adapted for the specification of IS models?
2) Is the property specification language adapted to specify IS properties?
3) Is the model checker capable of checking a sufficient number of instances of IS entities?

The structure of [2] is focused on the control of an information system or, in other words, determining the sequence of actions which the information system should accept.

III. Model Checking Tools

A. IESV

The verification tools which are presented in [1] include theorem proving and model checking. In theorem proving, verification is accomplished through mathematical proofs which derive a theorem. This approach involves two different disciplines: mathematics and logic. The main benefit of theorem proving is that unlike in model checking, the prover does not need to check a program's state space in order to verify certain properties. It is, however, more time consuming as the prover needs to think about the proof step to apply and then execute this

step. Following this, the prover needs to give the new proof state as output and await the next proof steps. Because of this, theorem proving is not generally implemented in industry while model checking is.

In model checking, the main idea is to prove whether a given model satisfies a certain specification property. The system is generally modeled by a Kripke Structure and proving certain properties is performed by determining truth formulas in system states. It is mentioned that traditional model checking is comprised of three sets: defining a formal model of the system, providing a particular system property, and invoking the model checking tool. As mentioned previously, Kripke Structures are used as a basis of representing the models. In Kripke Structures, every state is labeled with Boolean variables which evaluate the expressions in that state. The expressions, in turn, correlate to certain properties of the system. Computation Tree Logic, or CTL, is used to unwind Kripke Structures and visualize the resulting computation paths. CTL combines two kinds of operators: linear temporal logic and branching-time logic. In linear temporal logic, operators are used to describe events along a single computation path. In branching –time logic, operators are used to quantify over a set of states that are successors of a current state.

B. CMCTIS

The model checking tools presented in [2] for validation are ALLOY, CADP, FDR2, NUSMV, PROB, and SPIN. ALLOY is a symbolic model checker which utilizes first-order logic. CADP is described as a rich and modular toolbox. For the paper, LOTOS-NT is used to specify models and XTL is utilized to specify properties. FDR2 is an explicit state model checker which can check refinement, deadlocks, livelocks, and determinacy of process expressions. NUSMV is a model checker which is based on Symbolic Model Verifier (SMV) software. It verifies temporal logic properties in finite state systems, implementing a symbolic representation of the specification in order to check a model against a property. PROB is described as a model checking and animation tool for the B Method – a method of formal software development. The final model checking tool is SPIN which implements LTL model checking. This method computes transitions for each property needing verification.

IV. APPLICATIONS

A. IESV

One of the applications presented in [1] refers to a coffee vending machine. In this model, the Kripke structure is further examined where a user inserts a coin and chooses their coffee, the coffee is then brewed after a valid selection is made, and the user is able to abort the process before the final selection is made. The Kripke structure is displayed as three states where each one represents a step in the process. This example is also applied to Computational Tree Logic (CTL).

A downside of model checking is discussed where State Space Explosions may occur in embedded system verification. Because Kripke structures can become extremely large, it may be impossible to fully explore the entire state space due to time and memory being limited.

The other application which is discussed in [1] is model checking applied to microcontroller assembly code. In this verification approach, the model is derived from the source code. It is well known that with embedded systems, there is heavy usage of microcontrollers and C code. Because of this, model checking of the code itself is not enough to provide a good verification of the systems as a whole. With the implementation of model checking assembly code, the checker is able to work with code that is extremely close to the hardware. Even though using this process requires a whole model of the microcontroller (due to the software and hardware being so intimate), there is still the capability of targeting every piece of code and checking the system entirely.

B. CMCTIS

The main application of the concepts presented in [2] relates to a library system. In this system, there two types of entities: books and members. From these two entities, there are ten actions possible. Books can be acquired and discarded. Members can join, leave, lend, renew, return (a book), reserve (a book), cancel, and take (a book home). From this, there are 15 different properties that can be verified using model checkers. It is then mentioned that generally within information systems, there are two different types of properties distinguished. The first is the liveness property which represents the “live” system which is not stuck in deadlock or livelock. Liveness properties tend to describe sufficient conditions to enable an action. The second of the properties is the safety property which describes “necessary conditions to enable an action, or what a user is not allowed to do with the system at a given point [2]”.

In the analysis of this library system case study, the results of the study are displayed for the reader.

- All model checkers, excluding NUSMV and LOTOS-NT, supported abstraction over entity instances. This is a feature which allows parameterization of the number of instances for each entity and association.
- All model checkers supported representation of entity and association structures.
- All model checkers also supported representation of information systems scenarios.
- NUSMV, LOTOS-NT, and SPIN were the only model checkers to lack abstraction over entity instances, which allows for abstraction from entity instances using quantification on variables.
- PROB and ALLOY were the simplest to use for accessing states and events.
- ALLOY proved to be the most efficient model checker for execution time and number of entity instances.
- Tools support was available for all of the model checkers.

In the end, the study showed that a model checker for information systems needs to be polyvalent, or having many functions, and it should support states and events. The author finds that PROB seems to be the best fit for model checking information systems.

C. *Proposed Application*

Of the two sources covered in this paper, [1] can more closely be applied to my work activities. I am an engineer in the Crystal Growth Department for Detectors within Siemens Medical Solutions. In my department, we grow NaI(TI) scintillation crystals for use with SPECT scanners. The crystals begin as raw salt material and are brought through a number of steps, or stages, including drying, heating, and cooling down. The raw material is then placed into a furnace for the actual growth process. Here it is heated at extreme temperatures until crystallization occurs and the salt forms into a solid ingot, or crystal block. After this crystallization occurs, the ingot needs to be cut, pressed, and shaped before it's ready to be formed into a plate size, framed, and installed in a scanning device.

All of these processes are automated to some degree and many of them possess embedded systems as a form of control. With the current advances in technology, one of my main goals is to expand the use of embedded control systems within the department in order to replace some of the older analog controls. With the expansion of embedded control systems, it would be extremely beneficial to implement the concepts presented in [1]. Utilizing Kripke Structures, model checking can be performed on many of the systems used in the scintillation crystal manufacturing process. An example of this could pertain to the saw that is used to slice the crystal ingot after growth. As of now the saw is controlled with the use of analog components but the implementation of an embedded system would surely make it more efficient at cutting and increase longevity of the electronics. Implementing Kripke Structures, we have a textual function description:

- After pressing "Start" button, motor begins to spin and saw begins cutting crystal. This is state 1 to state 2.
- Saw continues cutting crystal (continues in state 2).
- Saw aborts and motor turns off on three conditions (returning to state 1): if the "Off" button is pressed, if crystal is fully cut (limit sensor is tripped), or if a timer runs out.

The timer function is essential in protecting the integrity of the crystal in cases where the motor moving the saw fails and the motor turning the cutter continues. This can cause serious damage to the crystal and in many cases it would need to be discarded. A simple Kripke Structure of this system can be seen in Figure 1 below.

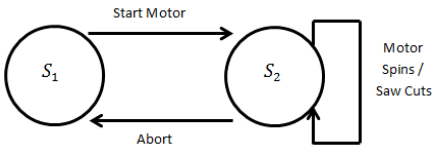

Figure 1
Kripke Structure of Crystal Saw

Recently, I have begun designing an embedded system for this saw using an Arduino development board as the microcontroller for the prototype. I immediately realized that checking the model and the intended operations was one of the most important, if not the most important steps, in the development process. In my initial attempt, I set a for loop as the driving force behind the timer or counter function.

```
if (sensorStatus==HIGH){
        for (int counter = 0; counter > 70; counter++) {
                ;
        }
}
```

I, however, quickly realized that once the loop initiated, it was difficult to leave it without using interrupts. Because my goal was to create as simple of a system as possible, I decided to create a simple counter and control it using simple if statements in lieu of loops.

```
if (sensorStatus==HIGH){
        count = counter++;
}
if (sensorStatus==LOW){
        counter = 0;
}
```

Going back to the foundations of what the system is intended to do can greatly help in solving issues in the design. I realized that instead of trying to fix the loop code, I should start over with a simple counter that counts sensor trips and work my way forward from there. Eventually, I was able to complete a fully functioning program for the saw.

As exhibited by this example, applying model checking methods to the embedded systems within the crystal manufacturing process could greatly aid in verifying that they work as intended. It is a simple and yet convenient and efficient way to break systems down to their most foundational levels and verify that they are operating properly.

V. Conclusion

As mentioned previously, verification and validation is extremely necessary and important in assuring that that a software system meets all of its specifications. These processes can help in the detection of issues early on in the development process, potentially saving future time and money. Model checking has been proven to be the most efficient way to perform these tasks as presented in [1] and [2]. No matter if it's working with embedded systems or information systems, model checking is an invaluable method of checking whether a given software system performs to the highest of its capabilities.

References

[1] T. Reinbacher, "Introduction to Embedded Software Verification", Master Embedded Systems, University of Applied Sciences Technikum Wien, 2008.

[2] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa and M. Ouenzar, "Comparison of Model Checking Tools for Information Systems", GRIL, Universite de Sherbrooke, 2010.