Themes Developed from Embedded Software Verification and Model Checking Tools for Information Systems

Edward Escobedo
Lewis University
CPSC-59700 Week 3 Assignment
One University Parkway, Romeoville, IL 60446
edwardescobedo@lewisu.edu

***Abstract*—Model checking provides a standard method of formally verifying finite states of systems. Temporal logic formulas are used for specifications about a system, and efficient symbolic algorithms are defined to traverse the different models used by the system and verify if the specifications are correct. This allows for large states to be verified and transverse in a short period. Based on publications discussing embedded software verification and model checking tools for information systems, the concepts of new knowledge and the model tools used to conduct the model checking will be presented. Also, areas of the applications covered and how these tools are used to address verification and testing in the professional environment is discussed. This presentation will conclude with a look into the future of model checking.**

Keywords—Model Checking, System Verification Embedded Software, Specification, Temporal Logic, System, Formal Model, Explicit State, Symbolic, Logic-Based, Bounded, Constraint Satisfaction

I. Introduction

Before starting to write code for a project, there needs to be an understanding of the daunting problem of software development with its flawed design requirements. Finding flaws at the beginning of development because of flawed requirements make sense because having to get rid of bugs later in the design often accompanies a high cost to the project and its bottom line [3]. Model checking has been proven as a successful technology to help verify design and requirements of a variety embedded systems. This has become the most successful approach that has emerged for verifying requirements. A basic run down consists of a model checking tool accepting system requirements or designs called models and property, which is a specification, that the program is expected to satisfy. The tool will then output a yes if the model satisfies the specifications and will generate a counterexample otherwise. The counterexample is just a justification of why the model did not meet the specification. History shows that during the 1980's the prevailing paradigm for verification was manual proving theoretic reasoning using axioms and interface rules generated by following systems. The need to develop tools to encompass to avoid the difficulties with manual, deductive proofs is the major factor in the development of model checking [1]. This paper provides an overview of the model checking process and the different types and applications. Concepts and the various checking tools, areas of application and its present use in the professional environment will be addressed.

II. Concepts of Model Checking

A. System Validation

Systems that are based on information processing is providing a critical service to its users. As information technology is increasingly becoming more and more into our daily lives, the reliability of this information processing is an essential factor in the design process. Usually, the design process starts with a requirements analysis. After this process delineated after several design phases, a prototype is obtained. Th process that a design will satisfy its requirements is referred to as system validation. Two destined techniques of validation often include peer review and testing. These processes are completely a manual activity. Peer reviews are usually conducted by a team of developers not involved in the design process. Testing can be considered an operational standard in which checking whether a given product specifications are realized to conform to its original abstract. In most design efforts, more time and effort is in the validation of the system than on construction. Model checking uses algorithms and is executed by computer tools to verify the correctness of systems. This automates the process and allows users to input a description of the specification and leaves the verification to the machine. This lets users locate errors and repair the prototype before moving on. There are two approaches to model checking logic-based approach or and behavior-based.

The logic based approach is made by capturing the desired state behavior of the system using a set of properties in either temporal or modal logic. A system is considered correct with respect these requirements when it satisfies these properties for a given initial state. The *behavior-based* approach both the desired and possible behaviors gave the same notation and equivalent relations are used as criteria for correctness. The two model checking concepts used are considered automated techniques that systematically checks if the property holds for the given initial state of that model [2].

B. *Model Checking Benefits and Limitations*

The advantages of model checking begin with its general approach with applications to software engineering, hardware verification, communication protocols, multi-agent systems and embedded systems. Examples and case studies have shown incorporating model checking in the development process does not delay the process any more than testing and simulation. For some cases, it has led to shorter design times. Also, because of its complexity and advanced techniques, model checking is able with larger state spaces [4]. Model checking can also support partial verification. This can be used to consider only a subset of all specifications. This provides efficiencies since restricting the validation to only checking the most relevant requirements.

As with anything, there are some limitations to consider. The first being that it is less suited for data-intensive applications since treatment of the information usually defines infinite state spaces. Model checking is mainly used for control-intensive applications with communications between systems. Only requirements stated are checked. The completeness of the desired properties is not guaranteed. Also, as with any software product, model checking software can be unreliable [4]. The benefits and limitations listed here are not all-inclusive. The information that can be derived from this study alone is beyond the scope of this paper. This section just demonstrated the concepts associated with model checking and list some of the general benefits and limitation.

III. Model Checking Tools

The assignment correspondent listed four model checking tools to consider. Explicit state, Symbolic model, Constraint satisfaction and Bounded model checkers are the four model checkers that will be discussed in the following paragraphs.

A. *Explicit State*

Explicit-state model checkers include CADP, SPIN, and FDR 2. These checkers use an explicit representation of the system in transition according to a specification. SPIN which is a software tool to verify models of physical systems. SPIN is one of the first checkers developed in 1980. SPIN is a generic verification model that supports verification of the design of asynchronous processes. SPIN focuses on proving the correctness of process interactions. As a formal tool SPIN aims to provide the following: [2]

- An interactive, notation for specifying design choices, without must implementation detail.
- A compact and powerful tool for expressing general correctness requirement.
- Establishing logical consistency of the design choices methodology.

Another explicit state checker is the CADP model. CADP is a toolbox for the analysis of communication protocols. It is considered an on the fly verification consisting of the analysis of a finite-state system by exploring and constructing its state space. This helps for a way to fight against state explosion, by enhancing the detection of errors with large state spaces. Techniques used by the CADP protocol are based on upon on-the-fly resolution of the boolean equation.

The last explicit state model to discuss is the FDR 2. This is a refinement based software tool. Although it is considered a model checker and it is also technically a refinement checker. As mentioned with the other explicit software templates, it has gone through several release updates and is now up to FDR 3.

B. *Symbolic*

Symbolic model checking method is used to avoid building a state graph by using Boolean formulas to repressed relations and sets. A distinction of this properties is characterized by least and greatest fixed points that are verified by manipulations of these formulas using ordered binary decision diagrams. These decision diagrams help to provide a compact canonical form of Boolean Functions. Applying this idea to temporal verification, it is observed that if the set of the state can be represented by a vector of Boolean variables, the set of states is then represented by a Boolean function which will return a true statement for all states in the set. In this way, the model checking algorithm can be developed which uses OBDD's that represent the sets and relations. This technique is symbolic model checking since symbolic variables are created to reflect the components of the program state rather the numerical values. Symbolic model checking is used to verify various regularly structured systems with many states automatically.

NuSMV is a simple symbolic model checker and is designed to be an open model checker which makes is readily available for verification of designs, a testbed for formal verification techniques, a tool used custom verification and employed in other areas. This model was developed as a joint project between Carnegie Mellon University and Istituto per la Ricerca Scientifica e Tecnologica.

C. *Bounded*

Model checking is the most widely used model. The design that is going to be verified is modeled as a finite state machine, and the formalized specification is written as temporal logic. The reachable state for the design is then transverse to verify its properties. Should the property fail, then a counterexample is created in the form of the sequence of the states. The properties are classified to safety and lines reports. The former declares what should not happen; the last describes what should eventually occur. The basic idea of bounded model checking is to consider only a finite prefix of a path to be witnessed to an existing model checking problem.

D. *Constraint Satisfaction*

Lastly, constraint satisfaction model checking uses programming logic to verify formulas. Constraints must be must be satisfied at different points of transactions and states. Based on the specified limitations, the constraint checker identifies instances where the constraints are not satisfied. One of the popular constraint satisfaction tools is the ProB model checking tool. The tool is used for B method. Its animation features allow the users confidence in the specifications, there is no ambiguity between the right values for the operation or choice variables. ProB is a model and constraint-based checker which allows for the use to detect various errors in B Specifications. The B method or specification is a collection of mathematically based techniques for specification, which is designed and implemented for software components. This is one of the few formal software development methods that supports a complete software lifecycle from specifications, refinement, implementation code, and maintenance [4].

IV. Application

Due to our reliance on functioning Information and Communications Technology has rapidly increased to where it is a necessity, not a luxury. The complexity of these information systems is engulfed in out daily life via the Internet. The reliability of these systems is a key issue in the system design process. System verification techniques need to be applied to both hardware and software to verify system specifications prescribe what the system has to do. Verification is obtained once the system satisfies all specification properties. Then the real model checking is conducted. This an independent algorithmic approach where the validity of the application is checked in all states.

A variety of different programs can be used for model checking. The application can be very complex if not understood correctly and what is needed to verify the system. Tools such as SPIN, CADP, NuSMV and ProB support temporal languages for satisfaction for property satisfaction. ALLOY and FDR2 use the same language but also for model and property specification.

Due to the compound nature of embedded software, it is must harder to detect defects which can lead to life-threatening situations, delays, and huge budget costs and insufficient productivity. There are all types of embedded software. Examples include cell phones, energy generation, home appliances and an automotive component such as such antilock brakes. With this comes some difficulties such as bridging the gap between legacy code and formal specification, verification of a real-time operating system, verifying concurrent embedded programs and certifying the tool that carries out the verification.

The application of the audit of embedded software should include the following procedures and process:

- Validate the Business and Test Requirments – review business requirements and understand the test requirements.
- Test Requirments Gathering – Review technical requirements, capture functional and non-functional requirements.
- Test Strategy – Design review, test strategy, and test plan development.
- Test Execution and Reporting – Integration testing, system testing, defect analysis, regression testing.
- Release testing – Acceptance Testing and sign-off.

Following this conventional process will aid in the application of the different model testing tools available. Embedded testing is difficult so standardizing how to apply the verification tool allows for users to spend the time analyzing the results instead of trying to determine what was configured incorrectly or missed during the planning stages of testing.

V. conclusion

Model checking did not arise just out of anywhere. Logical errors in software and communication protocols have become an increasingly significant. This can delay releasing a new product into the market or case failure of critical devices already in use. Many of these systems are viewed as having a finite number of states. The efficient verification procedure is needed to ensure product design. With the Internet, all systems and applications are integrated and rely on information from each other. This can bring down entire enclaves. A perfect example occurred in San Diego, CA in 2011 when an electrical employee in Arizona improperly implemented a device which brought down the whole electrical grid here in San Diego. It shut down the entire city. The everyday infrastructure we rely on such has street lights, gas for vehicles, grocery stores were out for 7 hours. Although an incontinence, this is a perfect example of embedded systems and the need for model checking.

References

[1] Emmerson, A. (2006, Fall). The Beginnings of Model Checking. Retrieved April 17, 2017, from http://www.citationmachine.net/apa/cite-a-website/manual

[2] Holzmann, G. (1997, May 5). The Model Checker SPIN. Retrieved from http://spinroot.com/spin/Doc/ieee97.pdf

[3] Katoen, J. (1998, September). Concepts, Algorithms, and Tools for Model Checking. Retrieved from http://cialdea.dia.uniroma3.it/teaching/logica/materiale/katoen.pdf

[4] Leitem, M. (2017, Summer). The B Method Formal Specification. Retrieved from www.iup.edu/WorkArea/DownloadAsset.aspx?id=80867

[5] Palshikar, G. (2004, February 14). An introduction to model checking. Retrieved April 18,2017, from http://www.embedded.com/design/prototyping-and development/4024929/An-introduction-to-model-checking.

[6] Ogawa, H. (2008, December 03). Model Checking Process with Goal-Oriented Requirements Analysis. Retrieved from http://ieeexplore.ieee.org/document/4724569/