Verification Framework for Detecting Safety Violations in UML Statecharts

C.M. Prashanth
Dept. of Computer Engg,
N.I.T.K, Surathkal,
INDIA-575 025.
prashanth_bcs@yahoo.co.in

Dr. K.Chandrashekar Shet
Dept. of Computer Engg,
N.I.T.K, Surathkal,
INDIA- 575 025.
kcshet@nitk.ac.in

Janees Elamkulam
IBM India Ltd,
Airport Road, Bangalore,
INDIA -560 017.
janees.ek@in.ibm.com

Abstract

The model based development is a widely accepted phenomenon to build dependable software. This has lead to development of tools which can generate deployable code from the model. Hence, ensuring the correctness of such models becomes extremely important. Model checking technique can be applied to detect specification violations in such models at the early stage of development life cycle. In practice, such validations are done using off-the-shelf model checkers. This technique though popular has a drawback that, model should be described in the native language of the model checker. In this paper, we propose a framework for the verification of the dynamic behavior of reactive systems modeled using UML (Unified Modeling Language) statechart diagrams. The model is translated to an intermediate representation by parsing the information embedded behind the UML statecharts, this intermediate representation is used for checking the safety violations. Verification framework proposed is scalable to complex systems.

1. Introduction

The development of dependable software has been the major goal for the advent of software engineering discipline. The traditional way of verifying software systems is through human inspection, simulation, and testing. Though these methods are cost effective, unfortunately these approaches provide no guarantee about the quality of the software. Human inspection is limited by the abilities of the reviewers, simulation and testing can only explore a minuscule fraction of the state space of any software system. Model driven software development has been a prominent means to enhance the understandability of the system's structure and behavior. It has prompted industries to develop tools which can generate the code in high level languages like C, C++ or JAVA from the model. IBM's Rational Rose RT [1] is one such tool for the development of embedded real time systems. As deployable binaries are generated from the model, ensuring model's correctness becomes highly essential. The commonly used formal model verification technique is model checking. Model checking [2] is a pragmatic technique that, given a finite-state model of a system and a logical property (expected system property), systematically checks whether model holds the property or not. If the model does not hold the expected property, an error trace (also called as counter example) is generated. The original model can be refined by leveraging information generated by the counter example; this approach is known as counter example guided model refinement [3]. Several model checking tools like SPIN (Simple Promela INterpreter) [4], SMV (Symbolic Model Verifier) [5], BLAST (Berkeley Lazy Abstraction software verification Tool) [6] and RuleBase [7] are in existence. The major drawback of using model checking tools for verification is that, they expect system to be modeled using their proprietary input language. The input languages of most of these tools are text based and lacks advantages of visual representation. Numerous researchers have tried to address this issue. We have reviewed some of the papers [8] [9] [10] and found that, though they suggest modeling the dynamic behavior of the system using UML(Unified Modeling Language) statechart diagrams (provides visual representation to the models), subsequently these statechart diagrams are translated to the input language of the model checker before verification. The translation process removes the abstraction of the models and exponentially increases the state space of the complex systems. This could lead to state-explosion-problem [11]. We in this paper present a verification frame work which avoids the usage of off-the-shelf-model-checker and translation of UML statechart models to input language of the model checker and hence the state explosion is minimized.

In the next section, we give the necessary introduction to model checking techniques and in section 3 proposed model verification framework &

978-0-7695-3136-6/08 $25.00 © 2008 IEEE
DOI 10.1109/AMS.2008.9

methodology are discussed. We draw conclusions in section 4.

2. Preliminaries

2.1. Explicit state model checking

Explicit state model checking is systematic and exhaustive searching of error states in a state space graph using graph search algorithms like DFS and BFS. A basic algorithm is that the checker starts from an initial state and recursively generates successive system states by executing the events of the system. States are stored in a hash table to ensure each state is explored at most once. The process continues either until all reachable states are explored or the checker runs out of resources. SPIN [4], developed at Bell Labs, is a widely used explicit state model checker. This expects that the model to be verified is constructed using PROMELA (Process Meta Language), which is the input language for SPIN. However, one of the main problems associated with explicit sate model checking is that of state-space explosion. The number of states for a complex software system is potentially exponential with respect to the number of processes in the system. As a result full verification is often impossible. Therefore, several techniques are used to reduce the memory required to store the state information or the number of states/paths explored. Some of the techniques are described below.

Abstraction [12] is one of the most important techniques for tackling state explosion problem. The idea is to check if a property holds on an abstract system, and then based on this result infer if the same property hold on the concrete system. An abstraction can be formed by defining a mapping between concrete states and abstract states such that the abstract transition system allows more behaviors than the concrete system but has fewer states. Verification task is performed on the abstract transition system, which has fewer states than the concrete transition system. The second way of dealing with the state explosion problem is to reduce the number of concrete states that need to be searched during the state space exploration. There are two main techniques that use this approach, namely, the symmetry reduction technique [13] and the partial order reduction technique [14]. Both of these techniques induce an equivalence relation on states of the system and perform analysis without exploring the states which have an equivalent state that has already been explored (i.e., during the state space exploration if we come to a state that has an equivalent state that has already been explored, then the current state in not explored).

.

2.2. Symbolic model checking

Symbolic model checking algorithms use special data structures such as BDD (Binary Decision Diagram) for the representation of state transition graph, as it allows efficient manipulation of boolean formulae. BDDs are directed acyclic graph obtained by removing the isomorphic sub trees or redundant vertices from binary decision trees [15]. In Symbolic model checking technique, states in state transition graph is symbolically represented by boolean formulae rather than keeping explicit list of states itself to avoid the state explosion problem. An SMV [5] system, developed by Mc Millan, is a BDD based model checker used for checking the model for properties expressed in temporal logic CTL (Computation Tree Logic). The SMV expects the model to be verified is constructed in its own input language. The underlying principle of SMV is depicted in the Fig.1.

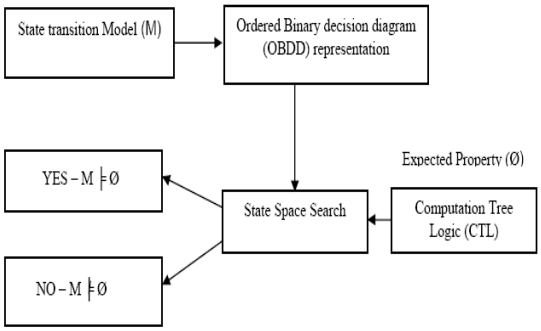

Figure 1. Engineering of Symbolic model verification

2.3. BMC-Bounded model checking

The symbolic model checking with BDDs have been successful formal verification technique till the late 90's. The major problem of this method is that BDDs may grow exponentially and amount of available memory restricts the size of the system that can be verified efficiently. The Bounded model checking technique [16] avoids the usage of BDD and has become a competing technique to BDD-based model checking. The bounded model checking was proposed by Biere et.al in 1999 [17] and the technique is shown in the Fig. 2. The basic idea here is to search for a property in executions whose length is bounded by some integer R. If no bug is found, then increases the value of R by one until a bug is found, problem becomes intractable or some pre-known upper bound is reached. The BMC problem can be reduced to propositional satisfiability problem and can be solved

by SAT methods (beyond the scope of this paper) rather than BDDs. NuSMV2 [18] incorporates BMC techniques in SMV and users can use both techniques (SMV & BMC) for verification.

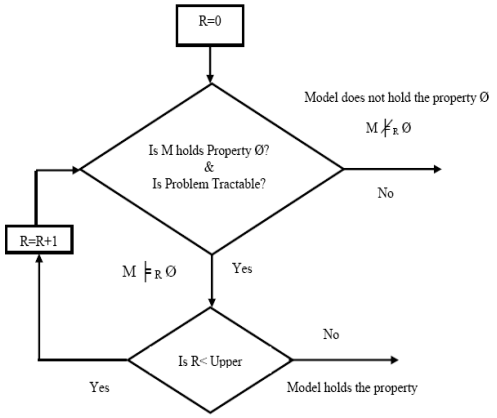

Figure 2. Engineering of Bounded model checking

2.4. UML statechart diagrams

The Unified Modeling Language (UML) is a general purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of the software system. The UML specification consists of two interrelated parts:

- UML semantics[1]: A meta model that specifies the abstract syntax and semantics of the UML object modeling components
- UML Notation: A graphic notation for the visual representation of the UML semantics

The UML has a collection of graphical notations each with a well defined semantics. It allows construction of several diagrams using these notations and relationship among them. These diagrams aid to visualize a system from different perspectives. The UML includes class diagram, sequence diagrams and statechart diagrams that can be used to specify both structural (class diagrams) and dynamic (sequence and statechart diagrams) views of software systems.
The UML statechart diagrams are used for behavioral modeling of the software and greatly increase the understanding of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go undetected. The statecharts were first introduced by David Harel in 1987 [20] as a visual formalism for complex reactive systems. The primary motivation behind this model was to overcome the limitations inherent in conventional state transition diagrams (or state diagrams for short) to describe the complex systems. State diagrams are directed graphs, with nodes denoting states, and arrows denoting the transitions. The UML statechart diagrams extend state diagrams to include notions of hierarchy (ability to cluster many states into a super state) and concurrency (orthogonality). Statechart shows sequence of states that an object goes through during its life cycle in response to stimuli, together with its responses and actions. A state in a statechart is represented by rounded rectangle and can be recursively decomposed into exclusive states (OR-state) or concurrent states (AND state). A simple transition may have a triggering event (whose occurrence cause the transition to take place), an enabling guard condition (which must be true for the transition to be taken), and output event and actions, all of which are optional. When a transition in a statechart is triggered (i.e. an event is received and guard condition becomes true), the object leaves its current state, initiates the action(s) for that transition and enters a new state. Any internal or external event is broadcasted to all states of all objects in the system. Transitions between concurrent states are not allowed, but synchronization and information exchange is possible through events. The transitions are represented by directed arrows with labels showing triggering event guard condition and actions (optional). An initial state is shown as a small solid filled circle and a final state is shown as a circle surrounding a small solid filled circle. The final state represents the completion of activity in the enclosing state.

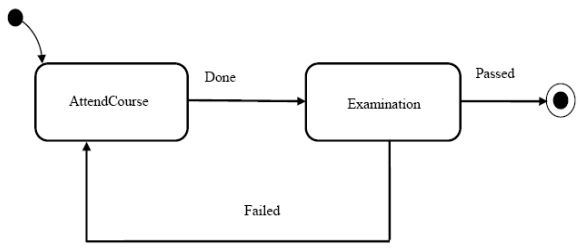

Figure 3 a. Abstraction of "Training system"

The Fig.3a is an abstraction of hypothetical "Training system, modeled using UML statechart diagram. The Figures 3b and 3c shows expansion of AttendCourse and Examination states of Fig. 3a into concurrent sub states (AND states) and sequential sub states (OR states) respectively. Visually AND state is depicted by splitting the state using dashed lines as shown in Fig. 3b. The Fig.3d shows completely expanded/flattened view of the abstract model, called global representation (without hierarchy).

[1] This is not discussed in this paper and interested readers can refer to [19].

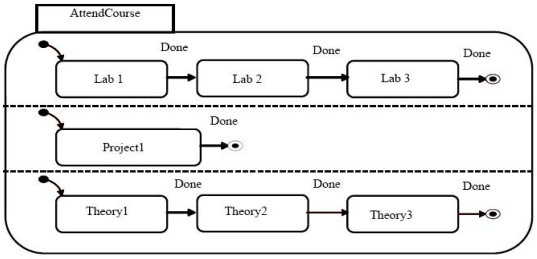

Figure 3 b. Concurrent refinement

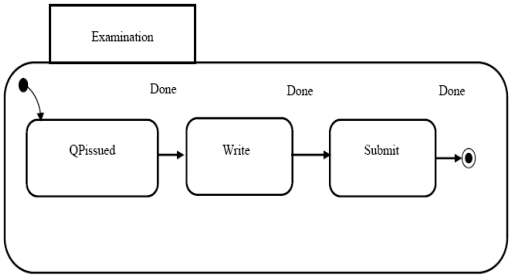

Fig 3 c. Sequential refinement

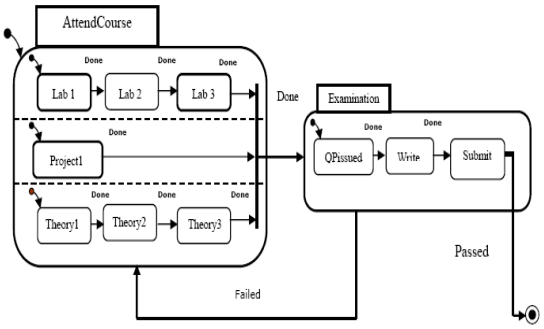

Figure 3 d. Global representation

3. Verification Framework

In this section, we present a framework, which supports the construction of automated safety violation analysis tools for UML statechart diagrams. This can be integrated with industrial CASE tools (which generate code from the models) for verifying models

3.1. Methodology

A widely known approach for verifying the complex systems is, by modeling them in the input language (mostly text-based) of the off-the-shelf model checker and passing them on to model checker. The property expected is specified in temporal logic. Subsequently, the need of visual formalism to the models was realized and UML statecharts were used for modeling dynamic behavior of the system. The verification of such models is done by first representing the UML statecharts in Extended Hierarchical Automata (EHA) [10] [21] [22] and then mapping to input language of the model checker. This approach is well received and successful for less complex systems. As the complexity of the system grows, this technique of flattening (removal of abstraction) the original model during verification would lead to "state-explosion" and hence aborts the verification process.

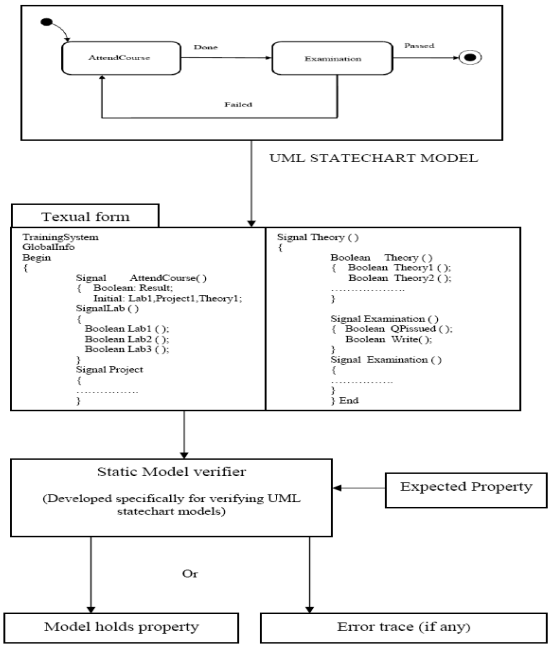

Figure 4. Verification model

The proposed framework for verification of systems does not use off-the-shelf model checker. The Fig. 4 depicts the methodology. The logics of the UML statechart diagram are captured in a textual format and then the same is converted to state space graph. Unlike most of the model checkers, here the data structure preserves the abstraction and whole state space graph is not brought into memory at once. The verification is done iteratively so that the state explosion problem discussed earlier in the paper would be minimized.

3.2. Architecture

We present the architecture of the verification environment in this section. The UML model verifier shown in Fig. 5, automatically searches the complete

set of states of the state space for an incorrect behavior and out puts error trace if any. There are four principal components, viz.,

- The UML editor
- Model compiler
- Property extractor
- Checker

The UML editor allows creating both abstract and concrete statechart models of the reactive systems. It supports all UML visual notations and semantics to capture all important design decisions. The user is prompted to enter relevant information as (s) he creates the model. This .information is later used by "model compiler" for constructing intermediate representation of the model.

The Model Compiler reads the UML statechart model and generates an intermediate representation of the same by parsing the UML statechart. This intermediate textual representation is the translated to state space graph.

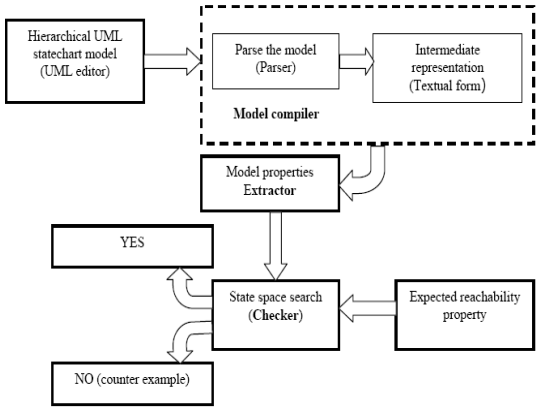

Figure 5. High-level design

The Property Extractor extracts model properties such as set of valid states, transitions and events from the textual form and the state space graph. It also gets safety property to be verified from the external world**.**

The Checker searches the state space iteratively for the safety violations. It outputs "Yes", if the specified safety property holds, otherwise outputs "No" and the error trace (counter example). As this is the core part of the verifier we discuss in detail the technological aspects in the next section.

3.3 Checker Algorithm

Detection of bad states in huge state space graph becomes hard at times. This is due to the limited availability of resources, memory in particular. Therefore devising a memory efficient algorithm is indispensable.

An iterative search approach is presented below:

Step 1: Start from the abstract level
Step 2: Let S represents set of reachable sates (given by property extractor)
Step 3: Let Ø be the expected property (let Ø be the safety violation or bad state)
Step 4: Let I be the set of initial states (given by property extractor)
Step 5: Find out set s which can be reached in one step from the current state
Step 6: Search the resulting s to find Ø
Step 7: Iterate the steps 5 and 6 until no new states are visited or Ø is found.
Step 8: Deepen/expand the part of the model and iterate the steps 2 through 7, until all the parts of the model is exhaustively searched
Step 9: If Ø is not found output "Yes" otherwise output "No" and path from the initial state.

This search algorithm is memory efficient as it does not keep the entire state-space of the model in the memory instead iteratively expands the model.

4. Conclusions

Most of the existing approaches translate UML statechart model into text based modeling language which can then be verified using off-the-shelf model checker. This translation process removes the salient features of the statecharts like hierarchy or abstraction. In other words, flattening of the statecharts leads to large state space requirement and makes verification approach not scalable to complex systems. The proposed iterative verification technique does not keep the entire state space of the UML statechart models in the memory. Hence, system with large state space can be handled efficiently. The usage of off-the-shelf model checker for verification of UML statechart models is avoided here; therefore no translation of UML models to the proprietary language of the model checker is necessary.

5. References

[1] IBM rational rose real time http://www.ibm.com/developerworks/ rational/library/797.html, visited on 01/12/2007
[2] Edmund M. Clarke,Jr., Orna Grumberg and Doron A. Peled , Model Checking (The MIT press, 1999)
[3] Edmund M Clarke, Ansgar Fehnker, et.al.: Abstraction and Counterexample refinement fin model checking of Hybrid Systems, Vol.14, No 4, International journal of foundations of computer science, (2003), 583-604
[4] Gerard J. Holzmann, The Model Checker Spin, IEEE Trans. on Software Engineering, Vol. 23, No. 5, (1997), 279-295
[5] Kenneth L. Mc. Millan, Symbolic Model Checking: An

approach to the state explosion problem, (Ph.D thesis submitted to Carnegie Mellon University (CMU), (1992)
[6] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Grégoire Sutre, Software Verification with BLAST, *Electronic Editions* (Springer LINK), 235-239
[7] I. Beer, S. Ben-David, C. Eisner and Landvar : RuleBase-an industry-oriented formal verification tool, Proceedings of 33rd Design Automation Conference (DAC), Association for Computing Machinery Inc.,(1996), 655-660.
[8] Johan Lilus, Ivan pores paltor, vUML: A tool for verifying UML models, Proceedings of the 14th IEEE international conference on automated software engineering,(1999) 225-228
[9] E. Clarke and W. Heinle, Modular translation of statecharts to SMV, Technical report, school of computer science, Carnegie Mellon University, Pittsburgh, (2000)
[10] Diego Latella, Istvan Majzik and Mieke Massink, Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model checker, Formal Aspects of Computing, volume 11(6), (1999),637-664
[11] Valmari,A.: The State explosion Problem, Lectures on Petri Nets I: Basic Models, LNCS 1491, Springer-Verlag (1998) 429-528
[12] Edmund M Clarke, Orna Grumberg & David E. Long, Model checking and Abstraction , Proc. of the 19th ACM SIGPLAN-SIGACT symposium on principles of programming languages Albuquerque, New Mexico, United States (1992),343 – 354
[13] E. A. Emerson and A. P. Sistla, Symmetry and model checking, In Proc. of the International Conf. on Computer Aided Verification, LNCS 697, ,Elounda, Greece, (1993). 463-478
[14] E. Clarke, O. Grumberg, M. Minea, D. Peled. State-Space Reduction using Partial-Ordering Techniques, STTT 2(3), (1999), 279-287
[15] Randal E. Bryant, Graph based algorithms for Boolean function manipulation, IEEE Trans. on Computers Vol. C-35 No. 8, (1986),677 – 691
[16] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, Yunshan Zhu, The Bounded Model Checking, In highly dependable software, Volume 58, Advances in computers, Academic press 2003
[17] A. Biere, A. Cimatti et.al., Symbolic model checking without BDDs, Proc. of the worksphop on Tools and Algorithms for the construction and Analysis of Systems, LNCS, Springer-verlag, (1999)
[18] NuSMV2: a new symbolic model checker, Visited on 16/08/2007 http://nusmv.irst.itc.it/
[19] Rational software, IBM, Microsoft et al, UML semantics OMG document:visited on 02/09/2007. ftp://ftp.omg.org/pub/docs/ad/97-08-04.pdf
[20] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science Computer Programming 8, (1987), 231-274.
[21] G.J. Holzmann et.al., Implementing statecharts in PROMELA/SPIN, proc. workshop on industrial strength formal specification techniques WIFT'98, USA, IEEE computer society, (1998).
[22] Adam Darvas et.al., Verification of UML statechart models of embedded systems 5th IEEE design & diagnostics of electronic circuits and systems workshop, (2002),70 -77