Application of Model Checking in Software Requirements Analysis

Yuejun Liu[1, 2]

[1]*School of Software Engineering, Anyang Normal University, Anyang 455002, Henan, China*
[2]*School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China*

Jing Su

School of Software Engineering, Anyang Normal University, Anyang 455002, Henan, China

Abstract
Aiming at the obscure questions in requirements analysis of the Software Engineering course, we propose a method using model checking technology to validate the user-established model in the early stage for better understanding the technology of software requirements analysis. Model Checking is mainly used to verify whether the model is consistent with the users' needs, i.e., whether the model given by requirements analysis meets the users' needs. In this article, we present an example-based scheduler to illustrate the Model Checking. With PRISM tool, it has shown us, clearly and intuitively, that the established model meets the user's needs, through which the understanding and knowledge of the requirements analysis has been enhanced. At the end, the potential significance of applying Model Checking in other courses is also discussed.
Key words: REQUIREMENTS ANALYSIS, FORMAL METHOD, MODEL CHECKING, RESOURCE TRANSITION SYSTEM, COMPUTATION TREE LOGIC.

1. Introduction

Software is the driving engine of modern society [1]. Today software is ubiquitous and plays an important role everywhere. Software engineering is a course to guide the development and maintenance of computer software [2], so it is obvious that students majoring in computer science must expertly master software engineering knowledge, which will play a positive role for future software development.

In software engineering course, a software life cycle mode is generally divided into four phases: requirements analysis, system design, system implementation, and operation and maintenance [2]. Among of them, the requirements analysis phase is the basis of the whole of system development [3]. The correctness and intelligibility of system requirements analysis are closely related to the developing speed and quality of the entire software system [4]. Because of limited teaching hours, shortage of development opportunities and experiences and few relationships between software engineering course and other courses, students cannot understand adequate

the contents of requirements analysis[5] [6]. To solve this problem, different scholars put forward different solutions in recent years: [7] describes a framework for applying the constructivist learning theory in the field of object-oriented software engineering. In[8], the authors proposed a cross-course design for teaching software engineering, and [9] indicates visualization-based projects teaching is popular with students. However, it is rare to enhance knowledge and understanding of software requirements analysis by utilizing Model Checking method. In this paper, we propose a new approach, based on Model Checking, to facilitate students more specifically to understand the contents of requirements analysis.

The rest of the paper is organized as follows: Section 2 introduces formal method and Model Checking. Section 3 describes how to constructing model. Section 4 discusses properties and validation. Section 5 illustrates an example of communication scheduler. Finally, in Section 6 we provide conclusions and future work directions.

2. Formal method and Model Checking

In traditional software development non-formal and semi-formal natural languages are used to describe users' requirements and protocols between developers and users, frequently. Non-formal and semi-formal natural languages are intuitive and easy to communication, but their semantic are ambiguous, prone to ambiguity and uncertainty. To this end, we need a formal method to describe the requirement problems, which is a well-defined mathematical method. Mathematical language is precise, consistency and fits logical reasoning, so people describe the requirement problems using formal methods to ensure the accuracy of requirements analysis. At the same time, every step of the reasoning process using mathematical rules can be proved, so the correctness verification of requirement description also can be guaranteed [10].

Model Checking as one of formal methods is mainly used to verify whether the model is consistent with the users' needs, i.e., whether the model given by requirements analysis meets the users' needs. The essence of the Model Checking is to explore all states of the system model in an exhaustive search manner for detecting whether the given system model and the users' needs (i.e., specific properties) are consistent, where the search process must be done in limited time and limited memory space.

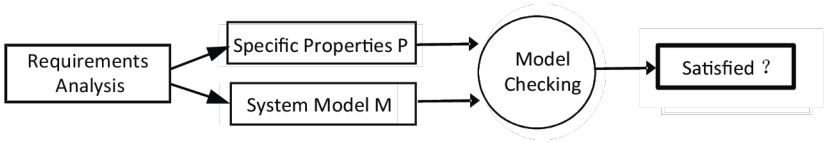

Figure 1. Model Checking

As shown in Figure 1, the developers firstly construct the system model M and system properties (i.e., specific properties) P according to the requirements analysis, then we search execution paths in the state space exhaustively. If we find a path started at an initial state, it shows the specific properties P are satisfied by the model M; otherwise, a counter-example is found. By analyzing the counter-example, the failure state can be located and it can guide us to further modify and optimize the model M [11].

3. Constructing model

The primary task of requirements analysis is to answer "what the system must do?" [2], so we should first exactly understand users' needs. Next, we need construct the model of users' needs by abstraction techniques, which can help us to understand and master the requirements of users. In the following, we introduce the resource transition system (RTS) to construct the system model [11] [12].

Definition 1 Resource Transition System (RTS): A resource transition system RTS is a tuple (S, Act, R, VAR, $\rightarrow$, s_0, g_0) where

S is a set of states,

Act is a set of actions,

R is a set of resources,

VAR: R $\times$ Act $\rightarrow$ R is a function of resources R,

$\rightarrow \in$ S $\times$ Con(R) $\times$ Act $\times$ S is a transition relation, and it describes the requirements for resources,

$s_0 \in$ S is a set of initial states, and

g_0 is a set of initial conditions.

Resource transition system (RTS) extends Transition System (TS) upon resources. Typically, the set of S, R and Act are finite in RTS. It ensures that the state space is finite, thus Model Checking can be completed within limited time.

Resource transition system may be seen as an underlying digraph: states are represented by nodes, and transitions are represented by directed edges between nodes. In this way, Model Checking of resource transition system evolves into a new one which can search the special path by walking along edges of the digraph one after another and edges corresponding to the transitions of RTS. If we could find a path, it illustrates that specific properties P can be satisfied by

the model M.

4. Describing properties and validation

In order to verify whether the model has completed "what system must be done", we need to describe the users' requirements (i.e., the function that must be done by system) without any ambiguity [13]. Properties can do it well, which are generally described as computation tree logic formula. Computation tree logic (CTL) is a branching time logic, originally be put forward by Clarke and Emerson in the early 1980s.Its model of time is a tree-like structure in which different paths present various future. Any one of paths might be the actual path that is realized [11].

Definition 2 Computation Tree Logic (CTL) formula: CTL formulas are constructed through using the following forms for limited times:

Proposition constants {true, false} and atomic proposition variable p are CTL formulas,

If Φ, φ are CTL formulas individually, $\neg \Phi$, $\Phi \wedge \varphi$, $\Phi \vee \varphi$ and $\Phi \rightarrow \varphi$ are all CTL formulas, and

If Φ, φ are CTL formulas individually, AX Φ, EX Φ, AF Φ, EF Φ, AG Φ, EG Φ, A[Φ U φ] and E[Φ U φ] are also CTL formulas.

Computation tree logic formulas describe that each of the CTL temporal connectives is a pair of symbols: the first of the pair may be A (i.e., along all paths) or E (i.e., at least there exists one path); and the second one of the pair may be X (i.e., next state), F (i.e., some future states), G (i.e., all future states) or U (i.e., until), respectively. For example, EF (q), GF (q) and A [Φ Uφ] are all legal CTL formulas.

CTL formulas are generally interpreted over the states and paths of a TS or RTS. The following formulas are explained: EF(q) means there is surely a reachable state which satisfies q in the future; GF(q) represents formula q can be satisfied infinitely often; and AG(p→E[p U q] represents from all reachable states satisfying p it always satisfies formula p until reaching a state satisfying q. But, EFG (q) and EF [p U q] are not CTL formulas [14] [15].

PRISM is a Model Checking tool developed by the M. Kwiatkowska. It uses a variant of BDDs or MTBDD storage model to enable the compact representation of transition matrices, and supports both qualitative and quantitative properties Model Checking, such as LTL, CTL, PTC, and so on [16]. By PRISM model checker, we can answer whether users' requirements have been satisfied and confirm whether the system model is correct. This process could further raise our confidence.

To verify the system model, people usually focus on four typical properties: reachability is that "something will happen in the future"; safety is that "bad things will never happen" [17]; liveness is that "good things will eventually happen"; and fairness is that "something will occur infinitely often".

5. Experiments

In order to enhance students' understanding of the requirements analysis, the following example in detail describes how Model Checking applies to requirements analysis of software engineering. Assume that there are two sets of network terminals, which are all connected to external network through one channel, but only one terminal is permitted to transmit messages at one time. Please design a scheduler to ensure that the two terminals can continuously access to external resources through the channel.

According to the example described, we divide each terminal process into three states: free section, trying section and critical section. Free section means network terminal process is in free-running. Trying section means the terminal is trying to transmit messages and critical section represents network terminal process is transmitting messages along the channel. Because only one network terminal process is allowed in the critical section at one time, we have to determine whether the other one is in the critical section (i.e., the process is transmitting messages). IF a terminal process is transmitting messages in critical section, the trying network terminal process is not permitted to enter but only continuously to wait. The established model based on the resource transition system is shown in Figure 2. Left side illustrates terminal process x, which x=0, x=1 and x=2 represent terminal process is in free section, trying section and critical section respectively. The meaning of right side is the same as left side.

Because reactive system is always running and the running results are nondeterministic, which is influenced by surrounding environment, students have some difficulties to understand it. In the following, we validate the model obtained by requirement analyzing and explicitly give its correctness for better understanding. The execution environment is a common machine with Intel(R) Core(TM) i7-4702MQ CPU 2.20GHz 2.19GHz and 4Gb of RAM. The operating system is windows 8, 64bit and PRISM is 4.2 Beta1. We construct the program model based on the resource transition system shown in Figure 2, which is illustrated by table 1 more in details [18][19][20].

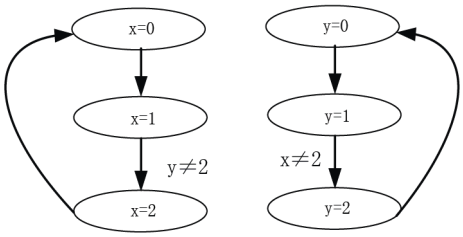

Figure 2. RTS model

Table 1. States of figure 2

ID	States	Meaning
1	x=0	x in free section
2	x=1	x in trying section
3	x=2	x in critical section
4	y=0	y in free section
5	y=1	y in trying section
6	y=2	y in critical section

The corresponding procedures are as follows:

```
// Process scheduler
mdp
module M1
   x : [0..2] init 0;            // Initialization
   [] x=0 -> (x'=1);
   [] x=1 & y!=2-> (x'=2);     // Enter into the critical
section
   [] x=2 -> (x'=0);            //Return to the free section
endmodule

module M2
   y : [0..2] init 0;
   [] y=0 -> (y'=1);
   [] y=1 & x!=2-> (y'=2);
   [] y=2 ->(y'=0);
Endmodule
```

Below the three specific properties will be verified [21]: (1) Pmax=? [F x=2], which returns the probability that process x eventually send messages successfully in the critical state x=2; (2) Pmax=? [F y=2], which returns the probability that process y eventually send messages successfully in the critical state y=2; (3) Pmax=? [F x=2 & y=2], which calculates the probability that two processes send messages successfully at one time. The simulation results are shown in Figure 3. We find the probability of properties (1) and (2) are all 1 or 100%, which ensure every terminal can successfully send messages, i.e., reachability. The probability of properties (3) is 0, which indicates two terminals cannot send messages concurrently, i.e., safety.

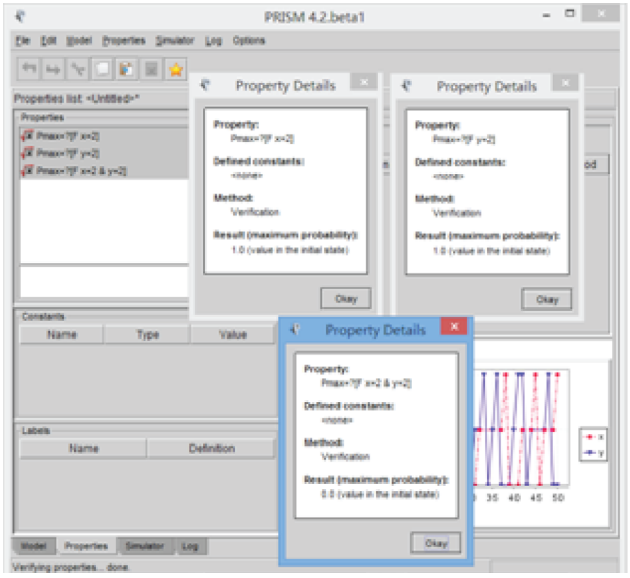

Figure 3. Simulation results

In order to verify the property, which the two terminals processes can continuously send messages via one channel respectively, we simulate the execution of the prior 50 steps by PRISM. The result is illustrated in the Figure 4. The processes x and y can all enter into the state of the critical section, i.e., the communication channel can be obtained by every one for sending messages. It indicates that every process will occur infinitely often, i.e., fairness.

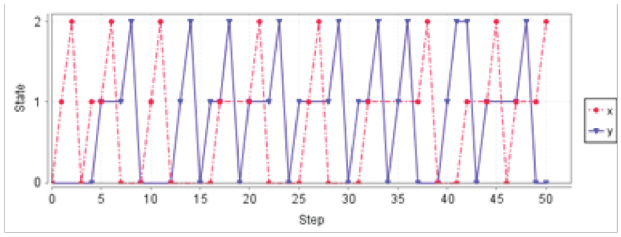

Figure 4. Simulation of first 50 steps

6. Conclusions

The contents of requirements analysis in Software Engineering is hard for students to comprehend due to the obscure concepts and limited experiences. PRISM tool intuitively shows us whether the model meets users' needs, which enhances more accurate understanding and mastery to requirements analysis. Model Checking technology is mainly used to verify the reliability in earlier of the software life cycle, and we plan to apply it to embedded systems design and hardware design teaching for content understood.

Acknowledgements

This work was partly supported by Soft Science Projects of Henan Province (132400410249), Natural Science Research Program of the Department of Education of Henan Province (12A520001) and High Reliability Software Research Team of Henan Province (B20141741).

References

1. Filieri A, Tamburrelli G, Ghezzi C. (2015) Supporting Self-adaptation via Quantitative Verification and Sensitivity Analysis at Run Time. *IEEE Transaction on software engineering,* DOI:10.1109/TSE.2015.2421318.
2. Zhang Haipan. (2008) *Introduction to Software Engineering*. Tsinghua University Press: Beijing.
3. Miao, M., & Weber, G. (2013). User Requirements Analysis in the Context of Multimodal Applications for Blind Users. *Mensch & Computer Workshop,* p.p. 283-286.
4. Okuda H, Ogata S, Matsuura S. (2013) Experimental development based on mapping rule between requirements analysis model and web framework specific design model. *SpringerPlus*, 2(1), p.p.1-8.
5. Dalal N. (2012) Teaching Tip Using Rapid Game Prototyping for Exploring Requirements Discovery and Modeling. *Journal of Information Systems Education*, 23(4), p.p.341.
6. Bareiša E, Karčiauskas E, Limanauskienė V, et al. (2005) Software Engineering Process and its Improvement in the Academy. *Information Technology and Control,* 34(1), p.p.63-70.
7. Hadjerrouit, S. (2003, November). Toward a Constructivist Approach to Web-based Instruction in Software Engineering. *In World conference on E-learning in coprporate, government, healthcare, and higher education (ELEARN)*, Phoenix, Arizona.
8. von Wangenheim, C. G., & Hauck, J. C. R. (2010). Teaching software process improvement and assessment. *In 17th European Systems and Software Process Improvement and Innovation Conference,* Grenoble Institute of Technology, p.p.25-34.
9. Müller C, Reina G, Burch M, et al. (2012) Large-Scale Visualization Projects for Teaching Software Engineering. *Computer Graphics and Applications,* 32(4), p.p.14-19.
10. Newcombe C, Rath T, Zhang F, et al. (2015) How Amazon web services uses formal methods. *Communications of the ACM,* 58(4), p.p.66-73.
11. Baier C, Katoen J P. (2008) *Principles of model checking*. MIT press: Cambridge.
12. Berendsen, J., Jansen, D. N., & Vaandrager, F. (2010) Fortuna: Model checking priced probabilistic timed automata. *Proc. of IEEE 2010 Seventh International Conference on the Quantitative Evaluation of Systems*, p.p.273-281.
13. Gao H, Miao H, Zeng H. (2013) Predictive Web Service Monitoring using Probabilistic Model Checking. *Appl. Math*, 7(1L), p.p.139-148.
14. Hinton, A., Kwiatkowska, M., Norman, G., & Parker, D. (2006). PRISM: A tool for automatic verification of probabilistic systems. *In Tools and Algorithms for the Construction and Analysis of Systems,* Springer,Berlin Heidelberg, p.p.441-444.
15. Baier C, Haverkort B, Hermanns H, et al. (2003) Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6), p.p.524-541.
16. Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. *In Computer aided verification*, Springer Berlin Heidelberg, p.p.585-591.
17. Kupferman O, Vardi M Y. (2001) Model checking of safety properties. *Formal Methods in System Design*, 19(3), p.p.291-314.
18. Antoniotti M, Policriti A, Ugel N, et al. (2003) Model building and model checking for biochemical processes. *Cell biochemistry and biophysics,* 38(3), p.p.271-286.
19. Alur R, Henzinger T A. (1999) Reactive modules. *Formal Methods in System Design*, 15(1), p.p.7-48.
20. Fokkink W, Pang J. (2005) Simplifying Itai-Rodeh leader election for anonymous rings. *Electronic Notes in Theoretical Computer Science*, 128(6), p.p.53-68.
21. Kwiatkowska M, Norman G, Parker D. (2006) Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science,* 153(2), p.p.5-31.