A Quantitative Analysis Method of Model Reliability

Yuejun Liu*
School of Computer Engineering and Science, Shanghai University, Shanghai 200444, Shanghai, China
School of Software Engineering, Anyang Normal University, Anyang455002, Henan, China
*Corresponding author (E-mail: aylyj@qq.com)

Jing Su
School of Software Engineering, Anyang Normal University, Anyang455002, Henan, China

Abstract
Requirement analysis is an essential step in software development. Its accuracy and reliability often determine the viability of entire process of development. But with no exceptions, all devices, resources, and transmissions possess imperfection in operational reliability. Therefore, deviations are inevitable in actual operation even for models that theoretically meet customers’ needs flawlessly. It is thus imperative to assess the reliability of the systematic models, so that customers can choose appropriately based on their own needs. In this paper, we propose a Markov Decision Process (MDP) and PRISM-based approach to analyze the model reliability quantitatively. It has been shown experimentally that the proposed method can assist one in decision making and early assessment of systematic model’s reliability.

Key words: Requirement Analysis, Markov Decision Process, Probability Computation Tree Logic

1. INTRODUCTION
Software is seen everywhere, playing increasingly important roles in our life nowadays (Filieri et al.,2015). Software Engineering is one of the fundamental courses for training students on software development and maintenance (Zhang,2008). In this course, the life cycle of a software product is composed of requirement analysis, system design, system implementation, and operational maintenance. Among them, requirement analysis is an essential step. Its accuracy and reliability often determine the viability of entire development process (Okuda et al.,2013). With no exceptions, all devices, resources, and transmissions possess imperfection in operational reliability. Therefore, deviations are inevitable in actual operation even for the model that theoretically meets customers’ needs flawlessly. It is thus imperative to assess the reliability of the systematic models, so that customers can choose appropriately based on their own needs.

To solve this problem, many scholars and research institutions in recent years have done some research: (Lei Z et al. 2015) proposes how to quantitatively analyze based on dynamic programming (Ghezzi and Sharifloo,2013) describes a model-based approach to assess the design solutions for software product lines, and (Li et al.,2014) presents an approach to assess maritime risk base on Bayesian networks. However, it is relatively rare to understand the quantitative analysis method of software reliability by utilizing Markov Decision Process and the PRISM. Due to lack of experience and opportunity in software development, insufficiency exists in students’ understanding of require analysis and quantitative analysis of model reliability (Liu and Su,2015). In this paper, we introduce a Probabilistic Model Checking-based approach to quantitatively study the systematic model reliability, aiming to improve students’ understanding on the discussed topic (Kwiatkowska et al.,2010).

The rest of the paper is organized as follows: Section 2 introduces the principle of Probability Model Checking, Section 3 describes basics of model construction, Section 4 discusses model property presentation, Section 5 gives an example of communication scheduler, and finally, Section 6 concludes the study, with comments on future work directions.

2. PROBABILITY MODEL CHECKING
Formal method is a well-defined mathematical method which can do logical reasoning. Due to the accuracy and consistency of mathematical language, each step describing the issue with formal method can be proved. That is to say, the correctness of system model can be ensured by using formal method (Newcombe et al.,2015). Model checking is one of formal methods, which often represents the behaviors of the system using state transition model and temporal logical formulas. It detects whether the given state transition model satisfies the specific temporal logic formula by exhaustively searching the state space. In other words, the satisfaction problem has been translated to search path problem (Wang et al.,2012). Probabilistic model checking method is an extended model checking method which describes the events with probability, so it can describe the satisfaction problem more accurately.
As shown in Figure 1, we exhaustively search execution paths and compute the successful probabilities according to the given system model and properties. The probability describes the satisfaction degree between the system model and property formulas and can guide us to choose appropriate service based on our requirements.

3. SYSTEM MODEL DESCRIPTION

Reactive System’s run is affected by random events in the surrounding environment, so it needs adjust strategies based on the current state for better running (Alur and Henzinger, 1999). In order to accurately understand and describe the random scenario system, Markov Decision Process (MDP) is introduced. MDP clearly answers how the system functions are implemented by continuous states transition. Because Markov Decision Process is a formal verification method, ambiguousness problem introduced by the non-formal or semi-formal natural language has been resolved completely. We say that MDP provides the theoretical basis for the following Model Checking (Baier and Katoen, 2008).

Definition 1: Markov Decision Process (MDP): a Markov Decision Process is a tuple $M = (S, Act, P, \rightarrow, I_0, \text{AP}, L)$, where

- $S$ is a set of states,
- $Act$ is a set of actions,
- $P: S \times Act \times S \rightarrow [0,1]$ is the transition probability function such that for all states $s \in S$ and actions $a \in Act$:
  $$\sum_{s' \in S} P(s, a, s') = 1,$$
- $\rightarrow \subseteq S \times S$ is a transition relation between states,
- $I_0 \subseteq S$ is a set of initial states:
  $$\sum_{s' \in I_0} P(\pi_0, s') = 1,$$
- $\text{AP}$ is a set of atomic propositions,
- $L: S \rightarrow 2^{\text{AP}}$ is a label function. Each state will be marked as the atomic propositions when the atomic propositions values are true at the state.

MDP describes a Cartesian product space of states. In order to travel all the states in limited time and limited space, we usually require the set of states $S$, the set of actions $Act$ and the set of atomic propositions $\text{AP}$ are all countable or limited. MDP may be seen as an underlying digraph: states are represented by nodes and transitions are represented by directed edges between nodes; an action identifier on the edge shows that the transition is activated by this action (an action identifier may be omitted when it is the only one originating from the state); a probability identifier on the edge represents the probability of the transition. In this way, Probability Model Checking of MDP evolves into a new searching path issue. The result path comes from the initial state and walks along edges of the digraph one after another, which satisfies the corresponding scheduler actions and transition probabilities.

In MDP, we assume that the successor and probability distribution don’t depend on previous states and actions but only depend on the current state and the scheduler action. It is known as the memoryless property which simplifies the complexity of issues (Katoen et al., 2011; Kai et al., 2015).

4. PROPERTY PRESENTATION

The “Properties must be satisfied with the model” is a basic requirement for users. When describing properties of system we should implement the consistency and unambiguous of the function and time (Gao et al., 2013). Computing Tree Logic (CTL) is an important branching temporal logic with a discrete notion of time and can better express the system property formulas. Because CTL can only give qualitative result “True” or “False”, in order to describe the properties quantitatively, Probability Computation Tree Logic (PCTL) is introduced.
Definition 2: Probability Computation Tree Logic (PCTL) formula: a PCTL formula is constructed through using the following forms for limited times (Liu and Su, 2015):

- Proposition constants [true, false] and atomic proposition variable p are PCTL state formulas.
- If ϕ₁, ϕ₂ are PCTL formulas individually, then ϕ₁ ∨ ϕ₂, ϕ₁ ∧ ϕ₂, ϕ₁(ϕ₂) are all PCTL state formulas, and
- If ϕ₁, ϕ₂ are PCTL state formulas individually, Xϕ₁, ϕ₁Uϕ₂ and ϕ₁U≤mϕ₂ are also PCTL path formulas.

Where, ϕ is a PCTL path formula and m ≥ 1 is a natural number for step length defining the U (Until) operation. The probabilistic operator ¬J represents a lower bound and/or upper bound of the probability formula ϕ, where ϕ ∈ {<, ≤, >, ≥, = } and J ∈ [0,1]. When J=1 or J=0, P_ϕ ( expreses that ϕ is true or false, which represents the quality property. When 0<J<1, P_ϕ (indicates that ϕ holds within the probability bounds given by J, which represents the quantity property. In addition, the operator P_ϕ ( represents the maximum (minimum) probability when ϕ holds (Baier and Katoen, 2008).

According to PCTL definition, the following common formulas are explained:

(1) ϕ represents the current state satisfies the atomic proposition p;
(2) ϕ₁ ∧ ϕ₂ indicates that ϕ₁ and ϕ₂ are all true at the current state;
(3) ϕ₁ U≤mϕ₂ shows along the path from the initial state ϕ₂ will be satisfied at most m transitions and ϕ₁ has been always satisfied before ϕ₂;
(4) P_ϕ ( represents the current state satisfies the formula ϕ which probability is constrained by ¬ J ;
(5) P_ϕ (true U ϕ₂) returns the maximum probability value when formula ϕ is true firstly;
(6) P_ϕ (true U≤m ϕ₂) returns the maximum probability value when formula ϕ is true within a maximum of m transitions.

5. QUANTITATIVE ANALYSIS AND EXPERIMENTS

Probabilistic model checker PRISM is a tool for modeling, simulation and formal verification developed by M. Kwiatkowska at the University of Oxford (Hinton et al., 2006; Kwiatkowska et al., 2011). PRISM consists of modeling editor and analog detector: Modeling editor can create the system model being described as PRISM language; analog detector can display analog animation scenes, graphical statistic results and the reliability of the specific properties. By PRISM model checker, the reliability of the model can be computed, which shows how user requirements have been satisfied and enhances our confidence. The basic grammatical structure of PRISM is as follows:

[action] guard -> prob₁ : update₁ + prob₂ : update₂ + ... + probₙ : updateₙ.

In this structure, action is a mark which indicates all the statements with the same one mark will be executed at the same time; guard is a Boolean condition which shows whether the statement is executed, i.e., when the guard condition is true, the statement will execute; ”prob₁ : update₁” represents the system will execute command update₁ with the probability prob₁. In addition, the following formula holds: prob₁ + prob₂ + ... + probₙ = 1. For example, a PRISM sentence is [] x = 0 -> 0.8; (x = 0) + 0.2; (x = 1).

The following example in detail describes how we analyze and compute the reliability of the system model. Assume that there are two network terminals x and y which all connect to the external network through one channel and only one terminal is permitted to transmit messages at one time. If the terminal x priority is higher than the terminal y, please design a scheduler to ensure that the two terminals can continuously access to external resources through the channel.

According to the example, we divide each terminal process into three states: free_state, trying_state and sending_state. The three states indicate respectively that the terminal is free-running, applying for the channel and sending messages along the channel. Because only one terminal is permitted to transmit messages at one time, we must assign transmission time to each terminal according to the priority. Because the network transition and the hardware are not absolutely reliable, we assume that the transition probability from one state to another state is 0.99 and the transition probability of one state to its own is 0.999. Next, the system model can be abstracted as follows: Figure 2 represents all the transitions and probabilities of terminal x, which x=0, x=1, x=2 and x=3 represent terminal process x is in free_state, trying_state, sending_state and failed_state respectively. The meaning of Figure 3 is as same as Figure 2. Figure 4 shows the scheduler strategy according to the resource channel. The resource channel has three states: idle_state (i.e., r=0), x_state (i.e., r=1) and y_state (i.e., r=2). The three states indicate respectively that the resource channel is free, occupied by the terminal x, occupied by the terminal y. The state r=3 represents failed scheduler. For simplicity, we assume that the transition probabilities of terminal x and y are respectively 0.558 and 0.441, which indicates that the priority of x is higher than y. Likewise, we assume that the failed transition probability is 0.001. A common machine with
Intel(R) Core(TM) i7-4702MQ CPU 2.20GHz 2.19GHz and 4Gb of RAM, the windows8-64bit operating system and PRISM-4.3 are the execution environment. The corresponding PRISM procedures are as follows(Kwiatkowska et al.,2006):

\[
\begin{align*}
x &= 0 \\
x &= 1 \\
x &= 2 \\
x &= 3 \\
r &= 2, 0.999 \\
r &= 0.99, 0.001 \\
r &= 1, 0.99 \\
r &= 0.99, 0.01 \\
r &= 1, 0.01 \\
r &= 2, 0.01 \\
\end{align*}
\]

\[
\begin{align*}
y &= 0 \\
y &= 1 \\
y &= 2 \\
y &= 3 \\
r &= 1, 0.999 \\
r &= 0.99, 0.001 \\
r &= 2, 0.99 \\
r &= 0.99, 0.01 \\
r &= 2, 0.01 \\
\end{align*}
\]

\[
\begin{align*}
r &= 0, 0.001 \\
r &= 0.999, 0.001 \\
r &= 2, 0.001 \\
r &= 0.999, 0.01 \\
r &= 2, 0.01 \\
\end{align*}
\]

**Figure 2.** Model of the terminal x  **Figure 3.** Model of the terminal y

**Figure 4.** Model of the channel r

```mdp
module Mx
  x : [0..3] init 0;
  // Initialization
  [] x=0 -> 0.99:(x'=1)+0.01:(x'=3);
  // Enter into the trying section with probability 0.99
  [] x=1 & r=1-> 0.99:(x'=2)+0.01:(x'=3);
  // Enter into the sending section with probability 0.99
  [] x=1 & r=2-> 0.999:(x'=1)+0.001:(x'=3);
  // Waiting in the trying section with probability 0.999
  [] x=2 -> 0.99:(x'=0)+0.01:(x'=3);
  // Return to the free section with probability 0.99
  [] x=3 ->0.99: (x'=0)+0.01:(x'=3);
  //Return to the free section with probability 1
Endmodule

module My
  y : [0..3] init 0;
  [] y=0 -> 0.99:(y'=1)+0.01:(y'=3);
  [] y=1 & r=2-> 0.99:(y'=2)+0.01:(y'=3);
  [] y=1 & r=1-> 0.999:(y'=1)+0.001:(y'=3);
  [] y=2 -> 0.99:(y'=0)+0.01:(y'=3);
  [] y=3 ->0.99:(y'=0)+0.01:(y'=3);
Endmodule
```
module Mr
r : [0..3] init 0;
[ ] r=0 -> 0.558:(r'=1)+0.441:(r'=2)+0.001:(r'=3);
[ ] r=1 -> 0.99:(r'=0)+0.01:(r'=3);
[ ] r=2 -> 0.99:(r'=0)+0.01:(r'=3);
[ ] r=3 -> 0.99:(r'=0)+0.01:(r'=3);
Endmodule

During Model Checking, people usually focus on four typical properties (Kupferman and Vardi, 2001): reachability, safety, liveness and fairness. The reachability represents something will happen; the safety indicates bad something will never appear; the liveness shows good something will eventually happen; and the fairness shows something will occur infinitely often.

Now we study these issues using the quantitative analysis method: (1) How much is the probability of the property formula $P_{\text{max}} = \exists x (x = 2)$ when the execution step $m$ of $x$ is 5, 10, 20 or 40 respectively? (2) How much is the probability of the property formula $P_{\text{max}} = \exists y (y = 2)$ when the execution step $n$ of $y$ is 5, 10, 20 or 40 respectively? (3) How much is the probability of the property formula $P_{\text{max}} = x = 2 \land y = 2$ when the network terminal $x$ and $y$ send messages concurrently successfully at one time? Probabilistic model checker PRISM shows the simulate results as follows:

Figure 5 shows that the probabilities of property formulas $P_{\text{max}} = \exists x (x = 2)$ and $P_{\text{max}} = \exists y (y = 2)$ continuously increase along with the execution steps. After 40 steps, we can say both the property formulas have almost 100% opportunities to send messages. Figure 6 shows that the opportunities which the terminal $x$ sends messages are always much more than the terminal $y$. This phenomenon is determined by the schedule strategies and is more obvious during the first 40 steps. Therefore, the terminal $x$ will be chosen to send messages when users have more strict time requirements (Cheng and Liu, 2015). Of course, the higher reliability you get, the more fees you pay.

Figure 7 indicates that the probability of property formula $P_{\text{max}} = x = 2 \land y = 2$ is 0. That is to say, the network terminal $x$ and $y$ cannot send messages concurrently by the one channel $r$, which ensures that the data transfer does not conflict.
We simulate the first 100 execution steps of system model. The results are illustrated in Figure 8 which shows that the terminal x and y can all enter into the state 2 to send messages and the terminal x has more opportunities to send messages than y. Likewise, the results can be represented by the channel r.

6. CONCLUSIONS

Due to the imperfection of reliability from all devices, resources, and transmissions, deviations from theoretical predictions are inevitable in actual model operation. To better assess the reliability of requirement analysis models, we introduce a Markov Decision Process (MDP) and PRISM-based approach to analyze the model reliability quantitatively. It has been shown experimentally that the proposed method can assist one in decision making and early assessment of systematic model’s reliability. However, it must be noted that only simple temporal definition, i.e., step size, has been considered in the study. Therefore, future works will appreciate more rigorous time confinements to be employed in the study.

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


