Performance Analysis of System Model Based on UML State Diagrams and Continuous-time Markov Chains

Yefei Zhao
Department of Computer Science, East China Normal University, Shanghai, China
Email: derekzhaoecnu@gmail.com

Zongyuan Yang, Jinkui Xie and Qiang Liu
Department of Computer Science, East China Normal University, Shanghai, China
Email: yzyuan@cs.ecnu.edu.cn, jkxie@cs.ecnu.edu.cn, lqiangecnu@gmail.com

***Abstract*—If software architecture is assigned with formal semantics, then automatic verification and validation can be performed during the process of model refinement. In this paper, we emphasized on the formal semantics of UML state diagrams oriented performance analysis. The exact definitions of the basic elements and composition mechanism of UML state diagrams are proposed, UML state diagrams is abstracted as a multi-tuple, CTMC models are abstracted as stochastic Kripke structure, mapping rules between the above two mathematics models are proposed, furthermore the corresponding formal semantics are generated. Finally, an asynchronous parallel composition queuing network is presented to illustrate how the theory is applied to formalize UML state diagrams. The key properties of system are manually deduced and validated. The results are analyzed and compared with the automatic executing results through model checker, which validated the practicability and validity of the theory.**

***Index Terms*—UML state diagrams, Markov process, CTMC, Probabilistic model checking, Software assurance**

I. INTRODUCTION

With the maturity of software component technology and the occurrence of CBSE (Component Based Software Engineering), CBSE becomes the mainstream software development technology. Presently shareholders always focus on software component in implementation phase and only care about functional property of system model. However, if some key performance properties of system are lately found not meet the requirement in the phase of development, it will cost more and reward less to modify the almost-finish system.

In [1], software component in design phase is described design component. Static topology structure and dynamic behavior of design component can be presented with UML diagrams. If UML diagrams are assigned with formal semantics oriented performance analysis, then system model can be automatically reasoned and analyzed key performance properties in the phase of requirement and design, thus the quantitative measure of key properties can be attained.

UML(Unified Modeling Language) is an object-oriented modeling language, which includes several sub-diagrams to describe different aspects of a system. Presently UML has become the de factor standard modeling language in industry ([2], [3]). UML state diagrams describes the responds to events and transitions between states of system model in its life cycle, thus is suitable to specify the structure, behavier and composition mechanism of software component. However, UML is a meta-model with only static semantics but without dynamic formal semantics, thus automatic analysis and verification can't be performed.

The research work to assign UML diagrams with formal semantics comprises of two categories: function verification and performance analysis. Firstly for function verification, UML diagrams is assigned with LTS (Label Transition System) semantics in [6], [7] and [11] or Kripke structure semantics in [8] and [12], thus the key system properties of system, such as atomic, liveliness, consistency and so on, can be automatically reasoned and verified by process algebras or model checking. Secondly for performance analysis, UML diagrams is assigned with formal semantics extended with time or probability, such as probabilistic Kripke structure semantics in [10] and [13] or stochastic LTS semantics in [9], thus the key quantitative performance measure can be automatically reasoned and solved by probabilistic model checking and stochastic process algebras.

In [4], probabilistic model checking is proposed by Marta Kwiakowska, Gethin Norman and Dave Parker from Oxford University. Probabilistic model checking extends the classical model checking with time and probability: system states of discrete-time (continuous-

Project number: No. 60703004, No. 20060269002, No. 09JC1405000, No. 09ZR1409500 and No. 2009054.
Corresponding author: Zongyuan Yang.

doi:10.4304/jsw.5.9.974-981

time) markov process is presented by probabilistic (stochastic) Kripke structure, while the key system properties to be verified is presented by PCTL (Probabilistic Computation Tree Logic) or CSL (Continuous Stochastic Logic), finally probabilistic model checker automatically reasons and analyze the property to determine whether it holds or get the quantitative performance measure through the method of exhausting system states. Presently probabilistic model checking has been widely used in probabilistic network protocol analysis, security-critical system validation, and so on. However, probabilistic model checking is program-level and difficult to understand, furthermore, the system designer is required to grasp some mathematics knowledge. To combine probabilistic model checking with graphical UML meta-model is a possible research direction in future.

The significance and content of this paper lie in:

(1) Probabilistic model checking is based on probabilistic Kripke structure and stochastic Kripke structure which is low in abstract level, while UML meta-model is the de factor standard in industry, thus easily accepted by most people because of its graphical and direct feature. If UML is assigned with formal semantics of probabilistic model checking, the complex mathematics knowledge in model checking theory can be hidden, thus graphical UML diagrams can replace mathematics model to perform automatic validation and analysis.

(2) In this paper, we emphasize on the exact definitions of basic elements and composition mechanism of UML state diagrams, as well as the mapping rules between UML state diagrams and stochastic Kripke structure. UML state diagrams is abstracted as a multi-tuple, while CTMC models are abstracted as stochastic Kripke structure. The transformation algorithm and mapping rules between the above two mathematics models are presented, which makes the theory be more general in applications, finally CTMC formal semantics are generated.

(3) Once UML state diagrams is assigned with CTMC formal semantics, the key system properties to be verified are presented with CSL, probabilistic model checker automatically reasons and verifies the CSL formula to get the quantitative performance measure. Not only quantitative measure but also functional property can be described by CSL formula, thus function validation and performance analysis can be performed simultaneously to improve software quality.

The structure of this paper is as follows: In section 2, the abstracting representation of UML state diagrams CTMC models are presented, the mapping rules between the above two mathematics models are found up, the exact definitions of basic elements and composition mechanism of UML state diagrams are also presented. In section 3, an asynchronous parallel composition CTMC system comprised of two models is presented to illustrate how to apply the theory to formalize UML state diagrams, the key performance properties are manually deduced and solved, the manual results are analyzed and compared with automatically executing results from probabilistic model checker.

II. FORMAL SEMANTICS OF UML STATE DIAGRAMS

A. Abstract representation of UML state diagrams

UML state diagrams is abstracted as a multi-tuple <S, S_{init}, E, G, A, S_F, L, T>, S represents the set of system states, G represents the set of guard conditions, A represents the set of actions, S_F represents the set of final states, L is a labeled function: $s \rightarrow 2^{AP}$, where $s \in S \wedge val(AP)=true$, function L takes input as state s and returns the set of atomic propositions that hold true in state s. T is a transition function: $s \times e \times g \times a \rightarrow s'$, where $s \in S \wedge e \in E \wedge val(g)=true \wedge a \in A \wedge s' \in S$, which represents: the current system state is s, if event e occurs and the value of guard condition g is true, then action a is executed and system transforms to state s'.

In this paper, we emphasize on the formal semantics of UML state diagrams oriented performance analysis, there are implicit mapping relationships between UML state diagrams and CTMC semantics.

B. Abstract representation of CTMC

In CTMC, each transition of system states relates with a rate, the cumulative distribution function of system states is exponential with parameter rate, thus CTMC is suitable to model markov process with time. For more detailed information, please refer to [16].

CTMC is abstracted as stochastic Kripke structure, which represents as a multi-tuple <M, V, E', G', R, V_s, T'>, where M represents module, a CTMC system comprises of one or more modules assembled by synchronous or asynchronous parallel composition, V represents the set of variables, E' represents the set of events, G' represents the set of guard conditions, R represents the set of rates, V_s represents the set of variables that record system state, T' represents the transition of system variable value.

PRISM input language is the abstract language-level representation of stochastic Kripke structure. For more detailed information, please refer to [5].

Rule 1. $(\sum L(s) \Rightarrow V) \wedge (E \Rightarrow E') \wedge (G \Rightarrow G') \wedge (A \Rightarrow R) \wedge (T \Rightarrow T') \wedge (num(s) \Rightarrow v_s$

"$(\mathbf{E \Rightarrow E')} \wedge \mathbf{(G \Rightarrow G')} \wedge \mathbf{(T \Rightarrow T')}$" represents event E, guard condition G and transition T in UML state diagrams are respectively mapped to event E', guard condition G' and transition T' in stochastic Kripke structure, which doesn't change in semantics.

"**($\sum$L(s)$\Rightarrow$V)**" represents the union of set of labels in state s, which is mapped to the set of variables V in stochastic Kripke structure. "**(A $\Rightarrow$ R)**" represents: in CTMC, the distribution of the executing time for action is exponential with parameter rate, thus the semantics of action (A) in UML state diagrams changes and is mapped to rate (R) in stochastic Kripke structure. "**(num(s)$\Rightarrow v_s$)**" represents that each state s in UML state diagrams is mapped to a unique number v_s, where $v_s \in V_s$.

C. System initialization

System initialization comprises of initialization of variable value and state value.

Definition 1 Variable initialization. Variable initialization in UML state diagrams is represented as: for all s $\in S_{init} \Rightarrow$ (val(L(s))=true), where for all s$\in S_{init}$, the value of labels related with state s is set true, otherwise is set false.

Rule 2. $\because$ ($\sum$L(s) $\Rightarrow$ V) (Rule 1), $\therefore$ (for all v $\in \sum$L(s) $\Rightarrow$ init(v) := true) $\wedge$ (for all v$\notin \sum$L(s)$\Rightarrow$ init(v) := false)

Rule 3. Initialization of state value in UML state diagrams is mapped to such semantics in stochastic Kripke structure as follows: for all s$\in S_{init} \Rightarrow$((init(s) := v_s $\wedge$ ($v_s \in V_s$)), where for each state s belongs to the set of initial states, the initial value of s is set as true and v_s belongs to the set of system variables V_s.

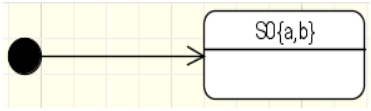

Figure 1. System initialization

For more detailed information about PRISM, PCTL and CSL, please refer to [15].

In Fig. 1, "s_0" represents that system is in state s_0, "{a, b}" represents the set of labels which hold true in state s_0. Given $\sum$L(s) = {a, b, c, d} $\wedge$ $\sum$num(s)={0,..,N}.

From definition 1, rule 2 and rule 3, we can get the PRISM source code in Fig.1 as follows:

```
s : [0..N] init 0;
a : bool init true;   b : bool init true;
c : bool init false;  d : bool init false;
```

D. Sequential transition

Definition 2 Sequential transition. In UML state diagrams, sequential transition is expressed as T_{seq}: s × e × g × a → t, where (s ∈ S) ∧ (e∈E) ∧ (g∈G) ∧ (a∈A) ∧ (t∈S)

Rule 4. $\because$ (a$\Rightarrow$r) $\wedge$ (e$\Rightarrow e'$) $\wedge$ (g$\Rightarrow g'$) (Rule 1), $\therefore$ sequential transition is expressed as T_{seq}' in stochastic Kripke structure:

$e' \times g' \rightarrow$ r × (next(v) := ?(v ∈ L(t)) × (next(s) := num(t)), which represents: given event e' occurs and guard condition g' holds true, if v∈L(t), then the value of variable v in next state is set as true with rate r, otherwise is set as false with the same rate, the number of system state is assigned with num(t), which represents the number of target state t.

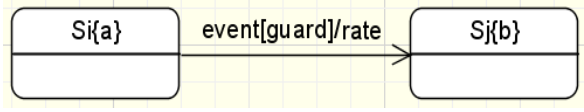

Figure 2. Sequential transition

In Fig. 2, s_i and s_j respectively represents the ith and jth system state, which is source or target state of sequential transition. "event" represents event, "guard" represents guard condition, the semantics of "rate" in UML state diagrams is action, however, it changes in stochastic Kripke structure and transforms to rate.

From definition 2 and rule 4, we can get the PRISM source code in Fig.2 as follows:

```
[event] (guard & s=i) → rate : (s'=j) & (b'=true) & (a'=false);
```

(b'=true) and (a'=false) represent that the variable changed in next state is assigned with new value.

Definition 3 Internal transition. In UML state diagrams, internal transition is expressed as T_{int}: s × e × g × a → s, where source state and target state are the same state, others are the same as sequential transition.

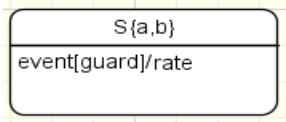

Figure 3. Internal transition

In Fig. 3, system state transforms to itself, so no variable value changes. From definition 3 and rule 4, we can get the PRISM source code in Fig.3 as follows:

```
[event] (guard & s=i) → rate : (s'=i);
```

E. Selection transition

Definition 4 Selection transition. In UML state diagrams, selection transition is expressed as T_{sel}: s × e × $\sum$($g_i \times a_i$) → t_i, which represents: for some source state s, if event e occurs, under different guard condition g_i, specific action a_i is executed, then system transforms to state t_i.

Rule 5. $\because$ (e$\Rightarrow e'$) $\wedge$ ($g_i \Rightarrow g'_i$) $\wedge$ ($a_i \Rightarrow r_i$) (Rule 1), $\therefore$ sequential transition is expressed as T_{sel}' in

stochastic Kripke structure: $\sum(e' \times g'_i \rightarrow r_i \times (v' := ?(v \in L(t_i)) \times (s' := num(t_i)))$

"$(v' =? (v \in L(t_i))$" represents that: all the variables whose value changed in target state t_i are assigned with new value "$? (v \in L(t_i)$".

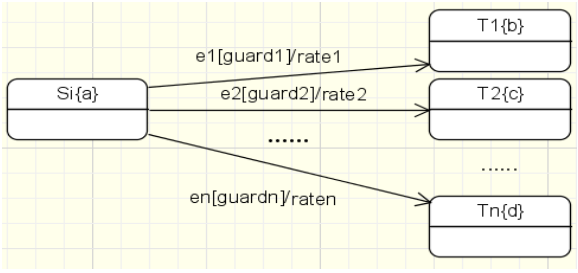

Figure 4. Selection transition

In Fig. 4, for a same source state, under different event, guard condition and rate, system will transform to respective target state.

From definition 4 and rule 5, we can get the PRISM source code in Fig.4 as follows:

[e1] (guard1 & s=i) → rate1 : (s'=t1) & (a'=false) & (b'=true);

[e2] (guard2 & s=i) → rate2 : (s'=t1) & (a'=false) & (c'=true);

……

[en] (guardn & s=i) → raten : (s'=t1) & (a'=false) & (d'=true);

In CTMC models, there isn't the constraint that the probability summation of all transitions from a source state for specific event should always be 1, so there isn't also non-determination selection transition.

F. Module declaration and module renaming

Definition 5 Model declaration. In UML diagrams, model declaration is presented with a name of complex state prefixed with keyword "module", the complex state comprises of a system initialization, several sequential transitions, internal transitions, selection transitions.

Rule 6. "module CSName" $\Rightarrow$ m $\wedge$ m$\in$M, which means: a complex state CSName prefixed with keyword "module" in UML state diagrams is mapped to an element m of set M in stochastic Kripke structure.

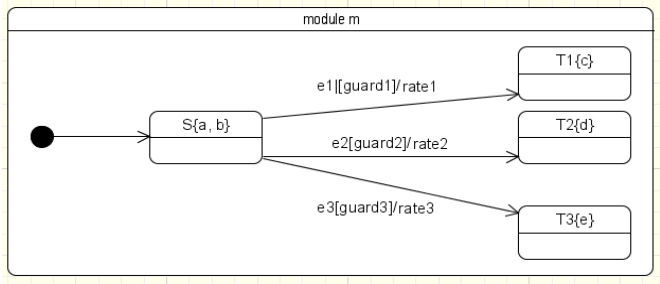

Figure 5. Module declaration

In Fig. 5, "m" is name of a complex state name, which comprises of a system initialization and a selection transition.

From definition 5 and rule 6, we can get the PRISM source code in Fig.5 as follows:

module m

s : [0..N] init 0;

endmodule

Rule 7. In UML state diagrams, given a defined module, if another module is all the same as the defined module except for variable name, then a new module can be defined by module renaming, which is mapped to such stochastic Kripke structure semantics as: $m_i = m_j$ [$\prod (v_k = v_k')$ $\wedge$ { m_i , m_j } $\subseteq$ M $\wedge$ $(\sum v_k \cup \sum v_k') \subseteq$ V]

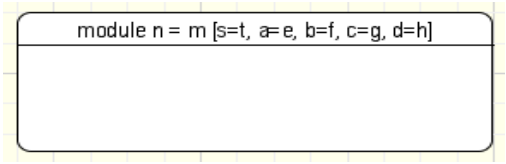

Figure 6. Module renaming

From rule 7, we can get the PRISM source code of model renaming in Fig.6 as follows:

module

n = m [s=t, a=e, b=f, c=g, d=h]

endmodule

G. Synchronous parallel composition and asynchronous parallel composition

Definition 6 Synchronous parallel composition. In UML state diagrams, a complex state prefixed with keyword "system" comprises of several module declarations, different modules synchronously execute on the same actions.

Rule 8. Synchronous parallel composition is mapped to such semantics in stochastic Kripke structure as: $m_1 \| m_2 \| \ldots \| m_n \wedge \{m_1, m_2, \ldots m_n\} \subseteq$ M

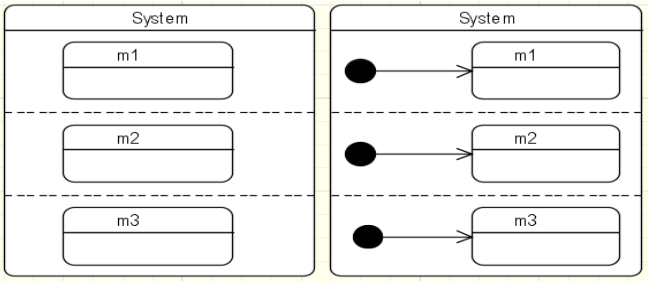

Figure 7. Synchronous parallel and Asynchronous parallel

From definition 6 and rule 8, we can get the PRISM source code in Fig.7 as follows:

system

$m_1 \| m_2 \| \ldots \| m_n$

endsystem

Definition 7 Asynchronous parallel composition. In UML state diagrams, a complex state prefixed with keyword "system" comprises of several module declarations, each module has its own initial state, different modules asynchronously execute on all actions.

Rule 9. Asynchronous parallel composition is mapped to such semantics in stochastic Kripke structure as: $m_1 ||| m_2 |||\ldots||| m_n \ \wedge \{m_1, m_2, \ldots m_n\} \subseteq \mathrm{M}$

From definition 7 and rule 9, we can get the PRISM source code in Fig.7 as follows:

system

$m_1 || m_2 ||\ldots|| m_n$

Endsystem

Besides above two composition mechanisms, there is still constraint parallel composition, for more detailed information, please refer to [15].

III. AUTOMATIC ANALYSIS AND VALIDITY VERIFICATION

A. Formal CTMC semantics of UML state diagrams

In this section, a queuing network queue size is presented, which illustrates how to apply above theory to generate formal CTMC semantics of UML state diagrams.

The UML state diagrams in Fig. 8 represents a queuing network comprised of two asynchronous composition parallel modules from the aspect of static topology structure and dynamic behavior. The first module "jobs" is a client-server queuing network, "client" sends request with rate 3, "server" provides service with rate 5 and size of the queuing network is 3. If no request is unprocessed, then system is labeled with "empty", otherwise if more than 3 requests is unprocessed, then system is labeled with "full". The second module "queue" is the renaming module of "jobs" (definition 6 and rule 8). The whole system is asynchronous parallel composition of module "jobs" and "queue".

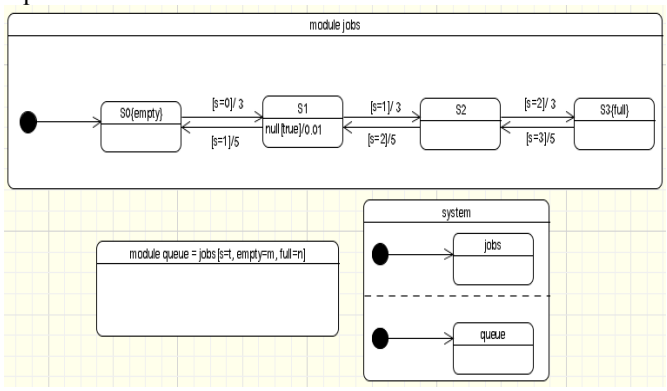

Figure 8. An asynchronous parallel composition CTMC system

From the basic elements and composition mechanism of UML state diagrams, as well as exact definition and mapping rules between UML state diagrams and stochastic Kripke structure, we can get the PRISM source code in Fig. 8 as follows:

```
ctmc
const int t;
module jobs
  s : [0..3] init 0;
  empty : bool init true;
  full : bool init false;
  [] s=0 -> 3 : (s'=1) & (empty'=false);
  [] s=1 -> 3 : (s'=2) + 5 : (s'=0) & (empty'=true);
  [] s=2 -> 3 : (s'=3) & (full'=true) + 5 : (s'=1);
  [] s=3 -> 5 : (s'=2) & (full'=false);
endmodule
module
  queue = jobs[ s=p, empty=m, full=n ]
endmodule
system jobs ||| queue endsystem
```

The above PRISM source code can be attained by the definitions and mapping rules in section 2. In the second line, an int type variable t is manually added to express time interval, which will be used in simulation experiment for key system property in section 3.2.

B. Automatic analysis and verification of key system properties

PRISM is a probabilistic model checking tool sets proposed by Marta Kwiakowska etc. from Oxford University. The concepts of time and probability are introduced into classical model checking framework so that DTMC, MDP and CTMC models can be processed in probabilistic model checking. In DTMC and MDP models, the key system properties to be verified is described with PCTL, in CTMC models, the key system properties to be analyzed is described with CSL. In PRISM, the method of exhausting system states is used to realize automatic verification and analysis and get quantitative performance measure.

For more detailed information about the syntax and semantics of CSL, please refer to [16]. The key system properties to be analyzed are described with CSL formula as follows:

Definition 8 System steady-state solution. It is defined as the probability of being in a state in the long-run.

CSL formula: S=? [full] S<0.1 [full]
S=? [s=0] S=? [s=1]
S=?[s=2] S=? [s=3]

"S=? [full]" represents the steady-state probability in case system fulfils the label "full". "S=? [s=0]" represents the steady-state probability in case system fulfils state s=0, likewise, for s=1, s=2 and s=3. Only if s=3, label "full" holds true, so "S=? [full]" ≡ "S=? [s=3]".

Definition 9 Reachable transient probability. It is defined as the transient probability in case that system starts a certain state, after some time intervals, reaches another state or fulfils some guard condition.

CSL formula:
P=? [true U[0,3] s=1 {s=0}]
P=? [true U[0,7.5] full {s=0}]
P=? [true U[0,7.5] full {s=1}]
P=? [true U[0,7.5] full {s=2}]
P=? [true U[0,7.5] full {s=3}]

"P=? [true U[0,3] s=1 {s=0}]" represents the transient probability in case that system starts from state s=0, after [0, 3] time intervals, reaches the state s=1. "P=? [true U[0,7.5] full {s=0}]" represents the transient probability in case that system starts from state s=0, after

[0, 7.5] time intervals, reaches the state that fulfils "full" is true. Likewise, for s=1, s=2 and s=3.

Definition 10 Transient simulation probability. The value range and variation steps of a variable can be specified, then PRISM can automatically draw the graph about transient probability simulation solution.

CSL fomula: P=? [true U[0,t] s=3]

"P=? [true U[0,t] s=3]" represent the probability in case that system reaches state s=3 in 0~t time intervals, where t is an integer variable, the value range and steps of t can be specified by user, then PRISM can automatically draw the simulation probability of the CSL formula in graph.

Software and hardware environment of the experiment are: Windows XP, Pentium 2.4G, 1G memory, probabilistic model checker PRISM 3.2. The CTMC formal semantics of the UML state diagrams in Fig. 8 is represented as the PRISM source code described in section 3.1, the key system properties to be analyzed is described with above CSL formula. The experiment result is shown in Fig. 9 as follows:

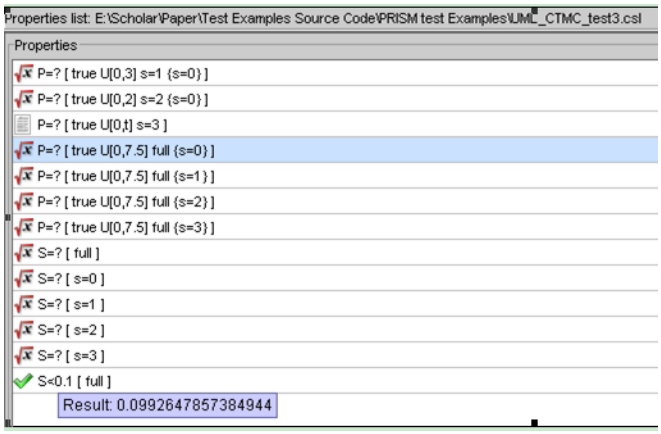

Figure 9. Automatic experiment result

Fig. 9 shows the automatic experiment result in PRISM, we extract the result shown in Tab. 1.

TABLE 1. AUTOMATIC EXPERIMENT RESULT OF KEY PROPERTIES

Property	CSL formula	Result
Steady-state solution	S=? [full]	0.09926
	S<0.1 [full]	true
	S=? [s=0]	0.4595
	S=? [s=1]	0.2757
	S=? [s=2]	0.1654
	S=? [s=3]	0.09926
Reachable transient probability	P=? [true U[0,3] s=1 {s=0}]	0.9998
	P=? [true U[0,7.5] full {s=0}]	0.9255
	P=? [true U[0,7.5] full {s=1}]	0.9344
	P=? [true U[0,7.5] full {s=2}]	0.9571
	P=? [true U[0,7.5] full {s=3}]	1.0

C. Manually computation and analysis of key system properties

(1). Steady-state solution

In [16], the algorithm and steps to solve steady-state probability of CTMC models are proposed. First, we generate the rate transition matrix Q. Second, global balance equation and normalized condition equation are listed. Third, steady-state probability is solved. We manually solve the steady-state probability of the CTMC models in Fig. 8 as follows:

Let π is a row vector, which represents the steady-state solution, $\pi = (p_1, p_2, p_3, p_4)$. From the CTMC model in Fig. 8, we can get the rate transition matrix

$$Q=\begin{bmatrix} -3 & 3 & 0 & 0 \\ 5 & -8 & 3 & 0 \\ 0 & 5 & -8 & 3 \\ 0 & 0 & 5 & -5 \end{bmatrix}$$

From global balance equation $\pi \times Q = 0$, we have:

$$(p_1, p_2, p_3, p_4) \times \begin{bmatrix} -3 & 3 & 0 & 0 \\ 5 & -8 & 3 & 0 \\ 0 & 5 & -8 & 3 \\ 0 & 0 & 5 & -5 \end{bmatrix} = 0 \Rightarrow$$

$$\begin{cases} -1.5p_1 + 2.5p_2 = 0 \\ 1.5p_1 - 4p_2 + 2.5p_3 = 0 \\ 1.5p_2 - 4p_3 + 2.5p_4 = 0 \\ 1.5p_3 - 2.5p_4 = 0 \end{cases},$$

The solution is:

$$p_1 = \frac{125}{27} p_4,\ p_2 = \frac{25}{9} p_4,\ p_3 = \frac{5}{3} p_4 \text{ -- (1)}$$

From normalized condition equation $\sum_{i=0}^{n} \pi_i = 1$, $\therefore$ $p_1 + p_2 + p_3 + p_4 = 1$, p_1, p_2 and p_3 are substituted for the above formulas, we can get the solution:

$p_1 = 0.4595$, $p_2 = 0.2757$, $p_3 = 0.1654$, $p_4 = 0.0992$

The above manual result is consistent with the automatic experiment result from PRISM, which proved that the theory is practicability and validity for steady-state solution.

(2). Reachable transient probability

In [16], Each CTMC model has an embedded DTMC model.

$$P^{emb(C)}(s,s') = \begin{cases} R(s,s')/E(s), if\,(E(s) > 0) \\ 1, if\,(E(s) = 0)\ \&\ (s = s') \\ 0, otherwise \end{cases},$$

From [16], the probability that CTMC model transforms from s_i to s_j in certain time interval $[t_1, t_2]$ can be solved by such formula as follows:

$$\Pr(C(s_i, [t_1, t_2], s_j)) = \Pr(C(s_i)) \times P^{emb(C)}(s_i, s_j) \times (e^{-E(s_i)\times t_1} - e^{-E(s_j)\times t_2})$$

For CSL formula "P=? [true U[0,3] s=1 {s=0}]", substitutes terms for corresponding data in the above formula, we get the solution:

$$\Pr(C(s=0, [0, 3], s=1)) = \Pr(C(s=0)) \times P^{emb(C)}(s=0, s=1) \times (e^{-E(s=0)\times 0} - e^{-E(s=0)\times 3})$$

$$= 1 \times 1 \times (e^{-3\times 0} - e^{-3\times 3}) = 1 - e^{-9} = 0.9998$$

The above manual solution is consistent with the automatic experiment result (0.9998) from PRISM, which proved that the theory is practicability and validity for reachable transient probability.

(3). Transient simulation probability

For CSL formula "P=? [true U[0,t] s=3]", the value range of variable t is specified as [0, 10] and the steps is set as 1, PRISM automatically drew the graph about simulation solution in Fig. 10 as follows:

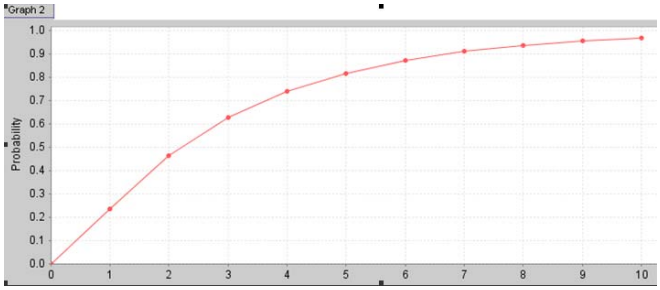

Figure 10. Transient simulation solution of CTMC model

In Fig. 10, probability p=0 in case time t=0, probability $p \approx 0.95$ in case time t=10. With the increase of time t, probability p becomes larger and larger. The probability that CTMC model transform from state s=0 to state s=3 becomes larger and larger with time increase, which is consistent with experience expectation, thus the practicability and validity of the theory is validated.

IV. Conclusion and Future Work

In this paper, we emphasized on the formal semantics of UML state diagrams oriented performance analysis. UML state diagrams is abstracted as a multi-tuple, CTMC model is abstracted as stochastic Kripke structure, the mapping rules between the above two mathematics models are found up, we also proposed the exact definitions of basic elements and composition mechanism of UML state diagrams. Finally an asynchronous parallel composition CTMC system comprised of two modules is presented to illustrate how to apply the above in performance analysis of UML state diagrams. The manual solution is consistent with the automatic experiment result from PRISM, which proved the practicability and validity of the theory.

In previous research work, we proposed that UML diagrams can be assigned with pi-calculus semantics in [11], Kripke structure semantics in [12] and probabilistic Kripke structure semantics in [13], thus formal function validation and performance analysis can be automatically performed during the process of model refinement.

Possible future work: according to the theory, we will develop a set of automatic tool sets which can formalize UML state diagrams with stochastic Kripke structure semantics. A possible practical route: Poseidon for UML → XMI text format → Java DOM (Docuement Object Model) parser → PRISM input code.

Presently probabilistic model checking can only process Markov process based system model, given current system state, probabilistic model checker can automatically reason and analyze system state in the future. If probabilistic model checking can be extended with additional operator about time and probability to apply Bayes formula, given current system state, then probabilistic model checker can automatically reason system state in the past, which will extend the reasoning range and ability of probabilistic model checking.

Acknowledgment

The work was supported by the National Natural Science Foundation of China under Grant No. 60703004, the National Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20060269002, Key Project of Basic Research of Shanghai under Grant No. 09JC1405000, Natural Science Foundation of Shanghai under Gran No. 09ZR1409500, and PhD Program Scholarship Fund of ECNU 2007 under Grant No. 2009054.

References

[1] Keller, Rudolf K, Schauer Reinhard, Design components: towards software composition at the design level. In: Proceedings of the 20th International Conference on Software Engineering, 302–311 ,1998

[2] OMG. OMG Unified Modeling Language specification version 1.5, March 2003. http://www.omg.org

[3] G. Booch, J. Rumbaugh and I. Jacobson. UML notation guide, version 1.1. Rational Software Corporation, Santa Clara, CA, 1997.

[4] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 2.0: A tool for probabilistic model checking. In Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04), pages 322–323. IEEE Computer Society Press, 2004.

[5] A Hinton, M Kwiatkowska, G Norman, D Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. Lecture Notes in Computer Science, Springer, 2006.

[6] Korenblat, K., Priami, Extraction of Pi-calculus specifications from a UML sequence and state diagrams. DEGAS IST-2001-32072, Technical Report No. DIT-03-07. 2003.

[7] Vitus SWL, Julian P. Consistency checking of sequence diagrams and statechart diagrams using the π-calculus. Proc. of the 5th Int'l Conf. on Integrated Formal Methods (IFM 2005). LNCS 3771, Berlin: Springer-Verlag, 2005. 347−365.

[8] P Inverardi, H Muccini, P Pelliccione. CHARMY: an extensible tool for architectural analysis. ACM SIGSOFT Software Engineering Notes, 2005

[9] D. N. Jasen, H. Hermanns, and J. P. Katoen. A Qos-oriented extension of UML state charts. LNCS, 2003.

[10] Jansen, D.N. Probabilistic UML statecharts for specification and verification: a case study. In: Critical systems development with UML: proceedings of the UML'02 workshop, Leipzig, Germany. pp. 121-131. Technical Report. 2002.

[11] Yefei Zhao, Zongyuan Yang, Jinkui Xie. Pi-calculus based assembly mechanism of UML state diagram and Validation of model refinement. International Conference on Electronic Computer Technology (ICECT 2009). 2009.

[12] Yefei Zhao, Zongyuan Yang, Jinkui Xie. Formal semantics of UML state diagram and automatic verification Based on Kripke structure. 22nd IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2009). 2009.

[13] Yefei Zhao, Zongyuan Yang, Jinkui Xie. System performance analysis Based on extended UML state diagram and Markov process. 2009. (submitted to journal)

[14] Poseidong for UML. http://www.gentleware.com/

[15] PRISM model checker, http://www.prismmodelchecker.org/tutorial/

[16] PRISM CTMC, http://www.prismmodelchecker.org/lectures/05-ctmcs.pdf

Yefei Zhao was born in Jinlin city, China, in May, 1978; received B.S. in Computer Science from North-Eastern University, Shenyang, China; received M.S. in Computer Science from East China Normal University, Shanghai, China. His research interests include formal method and software engineering.

He worked as a software engineer in Avant, SVA and DBtel Corporation from July, 2001 to July July, 2005 in Shanghai, China. Presently he works as a PH. D. candidate in Computer Science from East China Normal University, Shanghai, China. His publications include:

1. Yefei Zhao, Zongyuan Yang, Jinkui Xie. Formal semantics of UML state diagram and automatic verification Based on Kripke structure. 22nd IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2009). May, 2009.
2. Yefei Zhao, Zongyuan Yang, Jinkui Xie. Pi-calculus based assembly mechanism of UML state diagram and Validation of model refinement. International Conference on Electronic Computer Technology (ICECT 2009). February, 2009.
3. Yefei Zhao, Zongyuan Yang, Jinkui Xie, Qiang Liu. Formal model and analysis of sliding window protocol based on NuSMV. Journal of Computers. May, 2009.
4. Qiang Liu, Zongyuan Yang, Yefei Zhao. Design Patterns in Situation Calculus. International Conference on Software Technology and Engineering (ICSTE 2009). July, 2009.

Yefei Zhao is IEEE student member, IACSIT senior member, editor and reviewer of AICIT and IACSIT and PC Member of several international conferences. His work was supported by PhD Program Scholarship Fund of ECNU 2007 (No. 2009054).

Zongyuan Yang was born in August, 1953, Shanghai, China. He is a professor and PH. D. supervisor in Computer Science of East China Normal University. His research interests include software design and method, software component and formal method.

Jinkui Xie was born in October, 1975, Guilin, China. He received B.S., M.S. and PH.D. in Shanghai Jiao Tong University. Presently he works as a teacher in Computer Science, East China Normal University. His research interests include software security, trustable computation and type theory.

Qiang Liu was born in September, 1983 in Hunan province, China. Presently he is a PH. D. candidate in Computer Science, East China Normal University. His research interests include software engineering and formal method.