Quantitative Analysis of System Based on Extended UML State Diagrams and Probabilistic Model Checking

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 probabilistic model checking is applied in software architecture, function validation and quantitative analysis for Markov process based real-time model can be automatically performed in model refinement, which will improve software quality. In this paper, the exact definitions and mapping rules between UML state diagrams and probabilistic Kripke structure semantics are proposed, as well as the general translation algorithm of formal semantics. An asynchronous parallel composited DTMC system is illustrated, the key non-function properties of system are described by PCTL, which is automatically validated and analyzed by PRISM model checker. The key system properties are also manually deduced and proved, and compared with the experiment results. The mapping rules we proposed are bi-direction, so it can be used in both forward and reverse software engineering.**

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

I. INTRODUCTION

UML (Unified Modeling Language) is a graphical object-oriented language, which comprises of class diagrams, state diagrams, sequence diagrams, component diagrams, etc. Different aspects of system such as static topology structure and dynamic behavior can be described by different UML diagrams. Presently UML has become the de factor standard modeling language in industry ([1], [2]). However, UML is a meta-model with only static semantics but without dynamic formal semantics, thus automatic verification for key system properties can't be performed.

Software procedure is usually comprised of requirement, design, modeling, development, test, deployment, etc. In early stage of system requirement, shareholders usually describe the structure, behavior and composition mechanism of system with different UML diagrams (class diagrams, state diagrams, sequence diagrams, etc.), however, key performance measures of system, such as responding time, safety, reliability, etc. are often ignored. These quantitative measures have only just start to be cared lately in phase of development or test. If some key quantitative properties of system are not consistent with requirement at that time, it will cost more and reward less to modify the almost-finish system.

In probabilistic model checking, structure and behavior of system are described by probabilistic or stochastic Kripke structure, and key system properties to be validated are described by PCTL (Probabilistic Computation Tree Logic) or CSL (Continuous Stochastic Logic), thus key quantitative measures of system related with time or probability can be automatically reasoned and deduced. Three real-time models comprised of DTMC (Discrete-Time Markov Chains), MDP (Markov Decision Processes) and CTMC (Continuous-Time Markov Chains) can be automatically analyzed in probabilistic model checking. PRISM developed by Oxford University is a probabilistic model checking tools set, which supports solving the above three Markov processes.

If probabilistic model checking can be applied in early stage of software procedure, automatic function validation and quantitative analysis can be performed to improve software quality. It is a possible direction in software design methodology. In early stage of system requirement and design, if UML diagrams are assigned with formal semantics related with time and probability, key quantitative performance measures of system can be automatically deduced and analyzed to guarantee software reliability in model refinement.

The significance and contents of this paper lie in:

(1) After the semantics of basic element "action" in UML state diagrams are changed, there are implicit mapping relationship between UML state diagrams and probabilistic Kripke structure. In this paper, we explicitly proposed the exact

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

doi:10.4304/jsw.5.7.793-800

definitions and mapping rules between UML state diagrams and probabilistic model checking, thus UML state diagrams are assigned with formal semantics.

(2) The generating algorithm of translating UML state diagrams to probabilistic Kripke structure are presented in this paper. Both single module that comprised of state initialization, variable initialization, state transmission, etc. and different composition mechanisms that comprised of several modules are formalized with dynamic semantics.

(3) After extended UML state diagrams are translated to probabilistic Kripke structure semantics, the key system properties related with time or probability are described by PCTL, and quantitative measures can be automatically validated in probabilistic model checker, thus system designer needn't manually deduce and analyze these properties. Both quantitative performance measures and function properties can be described by PCTL, thus function validation and quantitative analysis can be simultaneously performed to improve software quality.

(4) The mapping rules proposed in this paper are bi-direction, thus it can be applied in both forward and reverse software engineering. Given UML state diagrams, they can be translated to formal semantics, vice versa.

The structure of this paper is: in section 2, the related works are presented; in section 3, the exact definitions and mapping rules between UML state diagrams and probabilistic Kripke structure are proposed, as well as the translation algorithm; in section 4, a asynchronous parallel composited distributed system is illustrated to show how quantitative analysis can be automatically performed in model checker after UML state diagrams are assigned with formal semantics, and the key quantitative measures are also manually deduced in theory; conclusion and future work are presented in section 5.

II. Related works

In [7], the mapping rules from probabilistic pi-calculus to probabilistic model checking are presented based on operational semantics, thus automatic translation can be performed. Process algebra is based on LTS (Label Transition System) structure, probabilistic model checking is based on probabilistic/stochastic Kripke structure, explicit mapping rules between the above two mathematics models are found up in [7]. Inspired by [7], internal state transmissions of object are described by UML state diagrams, which can be implicitly mapped to probabilistic Kripke structure. In this paper, the mapping rules and translation algorithm are presented, quantitative analysis for UML state diagrams can be automatically performed based on probabilistic Kripke structure semantics.

In [8] and [9], UML state diagrams and activity diagrams are translated to PEPA (Performance Evaluation Process Algebra) code, thus key quantitative properties such as steady-state probability, state or transition reward, etc. can be automatically analyzed. PEPA based on probabilistic LTS structure is in higher abstract level than probabilistic model checking, so it has one-to-one mapping relationship with UML state diagrams. However, probabilistic Kripke structure is in lower abstract level. In this paper, the mapping relationship from basic elements and composition mechanism of UML state diagrams to probabilistic Kripke structure is presented. The key quantitative properties are described by PCTL in probabilistic model checking, where range of validation is wider than PEPA.

In [10], firstly UML state diagrams are extended with time and probability, secondly translated to probabilistic time automata, thirdly translated to probabilistic model checking code, so the probabilistic temporal properties related with reliability can be automatically analyzed. However, the translation algorithm in [10] isn't described by exact mathematics model, neither specified with definitions and mapping rules, so the correctness of translation can't be proved by structure induction method.

In [11], P-statechart is UML state diagrams extended with probability. Model behaviors are described by P-statechart, models composition are described by UML collaboration diagrams, so validation and analysis of system can be automatically performed after formalization of UML diagrams. However, for the most important semantics translation, only state, guard and event in UML state diagrams are mapped to probabilistic model checking. Composition mechanisms are not referred in [11], and the mapping rules are not described by exact mathematic rules, so it is unable to direct the development of translation tools.

III. Probabilistic Kripke structure semantics of extended UML state diagrams

A. System initialization and labels of state

Definition 1: labels of state. It is defined as a function: $S \rightarrow 2^{AP}$, where S represents source state, AP represents a set of atomic propositions. In UML state diagrams, a system state is related with a set of labels, where the value of label is true in the state, otherwise is false.

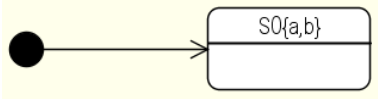

Figure 1. System initialization

In Fig. 1, the solid circle represents starting point that points to initial state, S_0 represents name of a state, a and b represent label names in state S_0. When system states are generalized, each state will be assigned with a

unique number, and the value of each label in system is assigned with true or false in system initialization.

Rule 1. Each state in UML state diagrams is assigned with a unique number, and the number of state in PRISM is recorded by an enumeration variable. The labels in state are mapped to a bool-type variable in PRISM, which is set true or false according to whether the label holds true in the state or not.

From Deifintion 1 and Rule 1, we can get the PRISM code in Fig. 1 as follows:

```
s : [0..n] init 0;
a : bool init true;
b : bool init true;
```

B. Sequential transition of state

Definition 2: Sequential transition. It is abstracted as a function: **S** × event × guard × Dist(S) → T, where S represents source state, T represents object state, "event" represents triggering event, "guard" represents guard condition, and Disg(S) represents the occurence probability of triggering event. In UML state diagrams, given the guard conditon holds true and triggering event is triggered, source state transmits to target state with probability p.

UML state diagrams comprised of five elements: source state, target state, event, guard condition and action.

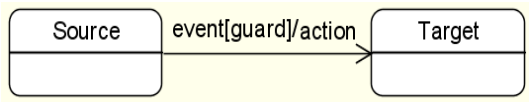

Figure 2. Squential transition

Each system state is assigned with a unique number after generalization, and "event" represents the triggering event. In DTMC, all the selection transitions are probabilistic, so event is NULL; in MDP, non-deterministic selection transitions exists, non-deterministic is solved by event.

Rule 2. Source state and target state in UML state diagrams are respectively mapped to an enumeration variable in PRISM, the value of enumeration variable is determined by generalized system variable. Event, guard and action in UML state diagrams are respectively mapped to event, guard and probability in PRISM.

"guard" represents guard condition, which is necessary for occurrence of transition. "guard" is bool expression of system variable. The semantics of "action" is extended as the occurrence probability of transition. In DTMC, for a same source state, the probability summation of all transitions equals to 1. In MDP, for a same sate, the probability summation of all transitions for a same event equals to 1.

From Definition 2 and Rule 2, we can get the PRISM code in Fig. 2 as follows:

```
s : [Source, Target] init Source;
[event] guard -> p: (s'=Target);
```

C. Internal transition of state

Definition 3: Internal transition. It is abstracted as a function: **S** × event × guard × Dist(S) → S, where S represents source state, other definitions of variable is the same as sequential transition. In UML state diagrams, given triggering event occurs and guard condition holds true, source state transmits to itself with probability p.

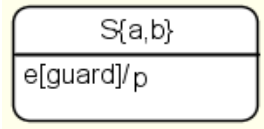

Figure 3. Internal transition

From Definition 3 and Rule 2, we can get the PRISM code in Fig. 3 as follows:

```
s : [Source, Target] init Source;
a : bool init true;
b : bool init true;
[event] guard -> p : (s'=Source);
```

D. Selection transition of state

Definition 4: Selection transition. It is abstracted as a function: $(S \times 2^{guard \times Dist(s)} \rightarrow T) \wedge (\sum_{i \in \{1..n\}} Dist(S)_i = 1)$, where S represents source state, $2^{guard \times Dist(s)}$ represents several partial orders of guard condition and probability distribution exist, $\sum_{i \in \{1..n\}} Dist(S)_i = 1$ is a constraint, which represents probability summation must equal to 1. In UML state diagrams, a state may have several successive states, and the mapping rules of state, labels, guard condition, event and probability is the same as sequential transition's.

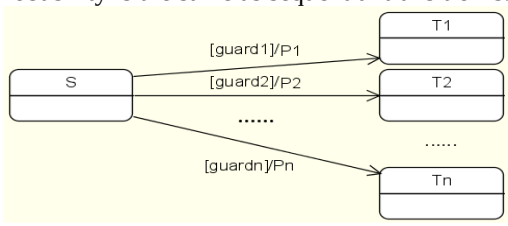

Figure 4. Selection transition in DTMC

According to different guard conditions, source state transmits to respective target state with a different probability. A constraint must be satisfied: $\sum_{i \in \{1..n\}} P_i = 1$

Rule 3. In UML state diagrams, a selection transition is comprised of several sequential transitions, which are mapped to several statements of sequential transition in PRISM. At the same time, a constraint must be satisfied: $S_j \Rightarrow \sum_{i \in \{1..n\}} P_i = 1$, which represents that the probability summation of selection transitions for the same source state must equal to 1.

From Definition 4 and Rule 3, we can get the PRISM code in Fig. 4 as follows:

s : [S0,T1,T2,...,Tn] init S0;
guard1 -> P1 : (s'=T1);
guard2 -> P2 : (s'=T2);
.....
guardn -> Pn : (s'=Tn);

Definition 5: Non-deterministic selection transition. It is abstracted as a function: $(S \times 2^{event \times guard \times Dist(s)} \rightarrow T) \wedge (event_i \Rightarrow \sum_{i \in \{1..n\}} Dist(S)_i = 1)$, where the definitions of variable are the same as selection transition'. According to different triggering event, a source state respectively transmits to different target state with a different probability, and the probability summation for a same triggering event must equal to 1.

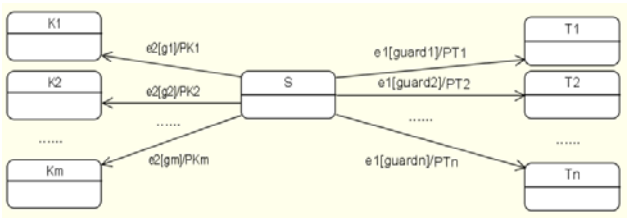

Figure 5. Non-deterministic selection transition in MDP

Rule 4. In UML state diagrams, according to different triggering event, a source state non-deterministic transmits to different target state, and they are mapped to several statements of sequential transition in PRISM. A constraint must be satisfied: $S_j \wedge event_k \Rightarrow \sum_{i \in \{1..n\}} P_i = 1$, where the probability summation must equal to 1 for the same source state.

From Definition 5 and Rule 4, we can get the PRISM code in Fig. 5 as follows:

s : [S0,T1,T2,...Tn,K1,K2,...Km];
[e1] guard1 -> PT1 : (s'=T1);
[e1] guard2 -> PT2 : (s'=T2);
.....
[e1] guardn -> PTn : (s'=Tn);
[e2] g1 -> PK1 : (s'=K1);
[e2] g2 -> PK2 : (s'=K2);
.....
[e2] gm -> PKm : (s'=Km);

E. Module declaration

Definition 6: Module declaration. Module is the smallest entity in probabilistic model checking, it is comprised of state initialization, state labels initialization, and several sequential (non-deterministic) selection (selection) transitions of state.

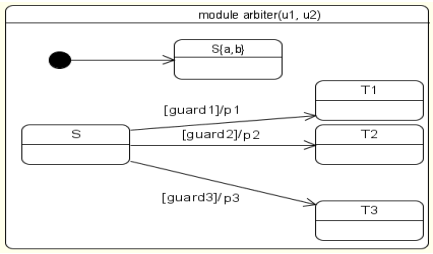

Figure 6. Module declaration

Rule 5. In UML state diagrams, a nest state comprised of state initialization and several sequential or selection transitions are mapped to a statement of module declaration prefixed with keyword "module" in PRISM, and the syntax should conform to above BNF paradigm.

From Definition 6 and Rule 5, we can get the PRISM code in Fig. 6 as follows:

Module arbiter
s : [0..N] init 0;

F. Modules composition

Modules composition is comprised of synchronous parallel composition, asynchronous parallel composition, and constraint parallel composition. Inter-module synchronous can be realized by event. Synchronous parallel composition is default in probabilistic model checking.

Definition 7: Synchronous parallel composition. Synchronous parallel composited modules parallel execute on all the same actions.

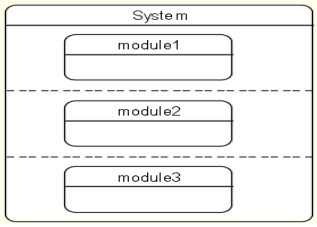

Figure 7. Synchronous parallel composition

Rule 6. In UML state diagrams, synchronous parallel composited modules are mapped to a statement prefixed by keyword "||" and should conform to above BNF paradigm in PRISM.

From Definition 7 and Rule 6, we can get the PRISM code in Fig. 7 as follows:

system
module1 || module2 || module3
endsystem

Definition 8: Asynchronous parallel composition. Complete interleaving semantics rather than synchronous relationship are generated in asynchronous parallel composited modules.

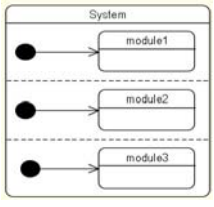

Figure 8. Asynchronous parallel composition

Rule 7. In UML state diagrams, asynchronous parallel composited modules are mapped to a statement prefixed by keyword "|||" and should conform to above BNF paradigm in PRISM.

From Definition 8 and Rule 7, we can get the PRISM code in Fig. 8 as follows:

```
system
  module1 ||| module2 ||| module3
endsystem
```

Besides synchronous and asynchronous parallel compositions, there are still constraint parallel composition and action-hidden parallel composition in probabilistic model checking. We won't discuss them because of page limitation.

G. Generation algorithm of probabilistic Kripke structure semantics

In this section, we will propose the algorithm of translating UML state diagrams to probabilistic Kripke structure semantics. The basic idea is: Firstly system states are circularly exhausted, and each state is assigned with a unique number; secondly each module in system is circularly found and translated to PRISM code according to above definitions and rules; finally formal semantics are generated according to different composition mechanism of modules. The java-like source code of the algorithm is as follows:

```
i := 0;
while((s=StateSet.NextElement()) != NULL)
    // each system state is assigned with an unique
number
{ s.num := i++; }
while((m=ModuleSet.NextElement()) != NULL)
    // module declaration are translated to the PRISM
code
{  m.StateInit();  m.VarInit();
    // state and variable initialization
   m.SeqTrans();  m.InternalTrans();
    // state transition are translated to PRISM code
    m.SelTrans();  m.MdpSelTrans();
}
// modules composition are translated to PRISM code
If(System.CompMod = 0) then System.SynComp();
    // synchronous, asynchronous and constraint parallel
    // composition
Else if(System.CompMod = 1) then
System.AsynComp();
Else if(System.CompMod = 2) then
System.ConstraComp();
```

Ⅳ. AUTOMATIC VALIDATION AND ANALYSIS BASED ON PROBABILISTIC KRIPKE STRUCTURE

A. Probabilistic Kripke structure semantics of UML state diagrams

In this section, a distributed system comprised of two asynchronous parallel composited modules is illustrated to show how to model a time-probability related system with extended UML state diagrams and probabilistic model checking, and function validation and performance analysis are automatically performed. An asynchronous parallel composited DTMC system is presented in Fig. 9 ([6]). Module "process1" is a critical resource requesting example: the initial state of system s_0 transmits to "try" state s_1 with probability 1, s_1 respectively transmits to itself, "fail" state s_2 and "succ" state s_3 with probability of 0.01, 0.01 and 0.98, s_2 and s_3 respective transmits to s_0 and s_3 itself. Module "process2" is a die example: initial state T_0 respectively transmits to "heads" state T_1 and "tails" state T_2 with probability of 0.5 and 0.5, and state T_1 and T_2 return to initial state T_0 with probability 1. Module "process1" and "process2" are asynchronous parallel composited, so complete interleaving semantics are generated.

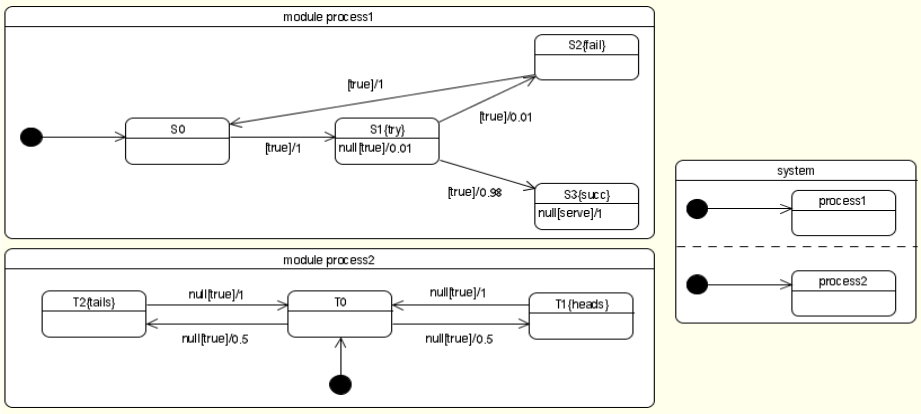

Figure 9. Asynchronous parallel composited DTMC system

From the definitions and rules in section 3, and the translating algorithm in section 3.7, we can get the PRISM code in Fig. 9 as follows:

```
dtmc
module process1
  s : [0..3] init 0;
```

```
  try : bool init false;
  succ : bool init false;
  fail : bool init false;
  [] s=0 -> 1 : (s'=1) & (try'=true);
  [] s=1 -> 0.01 : (s'=1) + 0.01 : (s'=2) & (try'=false) &
(fail'=true) + 0.98 : (s'=3) & (succ'=true) & (try'=false);
  [] s=2 -> 1 : (s'=0) & (fail'=false);
  [] s=3 -> 1 : (s'=3) & (succ'=true);
endmodule
module process2
  t : [0..2] init 0;
  heads : bool init false;
  tails : bool init false;
  [] t=0 -> 0.5 : (t'=1) & (heads'=true) + 0.5 : (t'=2) &
(tails'=true);
  [] t=1 -> 1 : (t'=0) & (heads'=false);
  [] t=2 -> 1 : (t'=0) & (tails'=false);
endmodule
system process1 ||| process2 endsystem
```

B. PCTL representation of key system properties and automatic validation

Key system properties are comprised of safety, fairness, liveliness, etc. Both function properties and quantitative measures can be automatically validated and deduced in probabilistic model checking. System properties of DTMC and MDP models are described by PCTL, and properties of CTMC models are described by CSL. The distributed system in Fig. 9 is based on DTMC. For more syntax and semantics of PCTL, please refer to [5] and [6].

Definition 9: Safety. System is in either "try" state or "succ" state in Fig. 9.

PCTL formulas: label "safe" = try | succ;

P>=0.99 [F "safe"]

P>=0.99 [G "safe"]

The semantics of the above PCTL formula are: the probability of either satisfying safety state in the future or globally satisfying safety state is more than 0.99.

Definition 10: Liveliness. Given system in "try" state, system can definitely arrive at "succ" state in the future in Fig. 9.

PCTL formulas:

P=? [try U succ {s=0}] P=? [try U succ {s=1}]

P=? [try U succ {s=2}] P=? [try U succ {s=3}]

P>=1 [F heads=true] P>=1 [F tails=true]

The semantics of the above PCTL formulas are: while system in different active state, the probability of satisfying "try" until "succ" becomes true is described by "P=? [try U succ {s=0/1/2/3}]". Starting from the initial state, the probability of satisfying "heads" or "tails" in the future is more than 1, which is described by "P>=1 [F tails/heads=true]".

Definition 11: Next-step liveliness. While system in different state, liveliness is satisfied in the next step rather than next several steps.

PCTL formulas: label "active" = !try | succ;

P=? [X "active" {s=0}] P=? [X "active" {s=1}]

P=? [X "active" {s=2}] P=? [X "active" {s=3}]

The semantics of the above PCTL formulas are: while system in different state, the probability of satisfying next-step liveliness is described by "P=? [X "active" {s=0/1/2/3}]".

Software and hardware environment of the experiment are: Windows XP, Pentium 2.4G, 1G memory, probabilistic model checker PRISM 3.2 ([5], [6]). The experiment result is shown in Fig. 10 as follows:

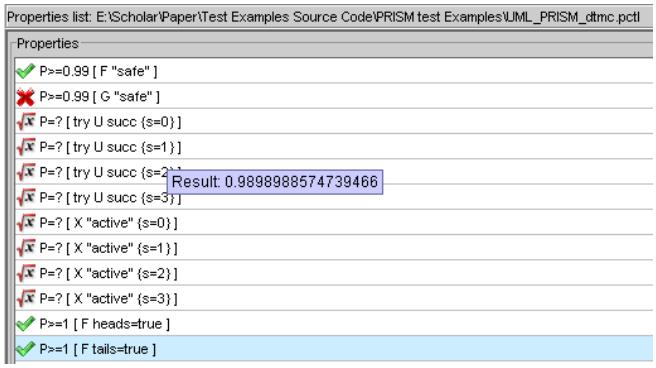

Figure 10. Automatic validation result in PRISM

Automatic validation results from PRISM model checker in Fig. 10 are extracted as Tab.1 as follows:

TABLE I. AUTOMATIC VALIDATION RESULTS OF PROPERTIES

PCTL formula	Result
P>=0.99 [F "safe"]	true
P>=0.99 [G "safe"]	false
P=? [try U succ {s=0}]	0
P=? [try U succ {s=1}]	0.989898
P=? [try U succ {s=2}]	0
P=? [try U succ {s=3}]	1
P=? [X "active" {s=0}]	0.5
P=? [X "active" {s=1}]	0.495
P=? [X "active" {s=2}]	1
P=? [X "active" {s=3}]	1
P>=1 [F heads=true]	true
P>=1 [F tails=true]	true

C. Deduction procedure of key system properties in theory

Next the above key properties are deduced and analyzed in theory.

(1) Safety

The result of PCTL formula "P>=0.99 [F "safe"]" is false, which represents the probability of satisfying safety state is more than 0.99.

For trace "$S_0 \to S_1$", $\because$ (Dist($S_0 \to S_1$) = 1) >= 0.99 $\wedge$ L(S_1) = {try}, $\therefore$ the result should be true.

The result of PCTL formula "P>=0.99 [G "safe"]" is false, which represents the probability of always satisfying safety state is less than 0.99.

For trace "$S_0 \to S_1 \to S_3$", $\because$ (Dist($S_0 \to S_1 \to S_3$) = 0.98) < 0.99 $\wedge$ L(S_3) = {succ}, $\therefore$ the result should be false.

(2) Liveliness

The result of PCTL formula "P=? [try U succ {s=0}]" is 0, which represents that: starting from initial state s_0, the probability of "try" holding true until "succ" becomes true is zero.

∵ L(S_0)={NULL} $\Rightarrow$ (try=false), ∴ we can draw the conclusion that: P=0.

The result and deduction procedure of PCTL formula "P=? [try U succ {s=2}]" are similar to formula "P=? [try U succ {s=0}]".

The result of PCTL formula "P=? [try U succ {s=1}]" is 0.989898, which represents that: starting from the initial state s_1, the probability of "try" holding true until "succ" becomes true is 0.989898.

∵ P(try U succ) = 1 – P(!(try U succ)) = 1 – (0.01 + 0.01×0.01 + 0.01×0.01×0.01 + ...) = 1 - $\sum_{i=1}^{\infty}(0.01)^i$ = $\frac{98}{99}$ = 0.989898, ∴ we can draw the above conclusion.

The result of formula "P=? [try U succ {s=3}]" is 1, which represents that: starting from the initial state s_3, the probability of "try" holding true until "succ" becomes true is 1.

∵ L(S_3)={succ} $\Rightarrow$ ((try U succ) = true), ∴ we can draw the conclusion: P=1.

The result of formula "P>=1 [F heads=true]" is true, which represents that: starting from the initial state T_0, the probability of reaching the state where "heads" eventually holds true is more than 1.

∵ P(F heads=true) = P(!(F heads=true)) = 1 – $(0.5)^{\infty}$ = 1, ∴ we can draw the above conclusion. The result and deduction procedure of formula "P>=1 [F tails=true]" is similar to "P>=1 [F heads=true]".

(3) Next-step liveliness

The result of formula "P=? [X "active" {s=0}]" is 0.5, which represents that: starting from the initial state s_0, the probability of reaching the state where "active" holds true in the next step is 0.5. Because process1 ||| process2, which represents process 1 and process 2 are asynchronous parallel composited, the two modules are scheduled with the same probability, so the probability is 0.5.

∵ If process1 is scheduled, then (Dist($S_0 \rightarrow S_1$)=1 ∧ L(S_1) = {try}) $\Rightarrow$ P(X "active")=0; if process1 isn't scheduled, then (Dist(S_0)=1 ∧ L(S_0) = {NULL}) $\Rightarrow$ P(X "active")=1; from full probability formula, P(X "active" {s=0}) = 0.5×0 + 0.5×1 = 0.5, ∴ we can draw the above conclusion.

The result of formula "P=? [X "active" {s=1}]" is 0.495, which represents that: starting from the initial state s_1, the probability of reaching the state where "active" holds true in the next step is 0.495.

∵ starting from s_1, ∴ s_1, s_2, and s_3 can be reached in the next step, and P("active"{s=1}) = 0.01×0 =0 , P("active"{s=2}) = 0.01 × 1 =0.01 , P("active"{s=3}) = 0.98×1 = 0.98. From full probability formula and equiprobability of asynchronous parallel composition, P(X "active" {s=1}) = 0.5×(0.01×0 + 0.01×1 + 0.98×1) = 0.495, ∴ we can draw the above conclusion.

The deduction procedure of the other two PCTL formulas are similar to formula "P=? [X "active" {s=1}]".

From section 4.2 and 4.3, the deduction procedure of these PCTL formulas in theory is consistent with the automatic validation result from PRISM model checker, which proves the practicability and validity of the translating algorithm from UML state diagrams to probabilistic Kripke structure semantics.

Ⅴ. CONCLUSION AND FUTURE WORK

If probabilistic model checking is applied in software architecture, function validation and quantitative analysis can be automatically performed in model refinement, which will improve software reliability. In this paper, the exact definitions and bi-direction mapping rules between UML state diagrams and probabilistic Kripke structure are proposed, as well as the translating algorithm. An asynchronous parallel composited DTMC system is illustrated to perform automatically function validation and quantitative analysis for key properties in PRISM model checker. Manual deduction results are consistent with automatic verification results in model checker, which proves the practicability and validity of the above theory. The mapping rules proposed in this paper are bi-direction, so they can be applied in both forward and reverse software engineering.

In [13] and [14], design component ([15]) is described by UML diagrams, which are respectively assigned with pi-calculus semantics and Kripke structure semantics, so function validation can be automatically performed. In this paper, we proposed a method to do automatic quantitative analysis for key properties in phase of requirement and design.

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

In current research work of assigning UML diagrams with formal semantics related with time or probability, continuous-time real-time models based on CTMC and discrete-time probabilistic models can only be separately processed in probabilistic model checking. We notice that the formal parameters and their number of DTMC, MDP and CTMC are consistent with each other. So we will propose a theory framework comprised of UML, DTMC, MDP and CTMC, which can simultaneously recognize different types of model, so

perform function validation and quantitative analysis can be automatically performed.

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] OMG. OMG Unified Modeling Language specification version 1.5, March 2003. http://www.omg.org
[2] G. Booch, J. Rumbaugh and I. Jacobson. UML notation guide, version 1.1. Rational Software Corporation, Santa Clara, CA, 1997.
[3] Michael Sharpe. General theory of Markov processes. Academic Press, San Diego, 1988.
[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] PRISM model checker, http://www.prismmodelchecker.org/tutorial/
[7] G Norman, C Palamidessi, D Parker, P Wu. Model checking the probabilistic pi-calculus. Quantitative Evaluation of Systems, 2007.
[8] D. N. Jasen, H. Hermanns, and J. P. Katoen. A Qos-oriented extension of UML state charts. LNCS 2863: 76-91.
[9] NV Haenel. User guide for the java edition of the pepa workbench. LFCS, University of Edinburgh, 2003
[10] N Addouche, C Antoine, J Montmain. Combining extended uml models and formal methods to analyze real-time systems. Lecture notes in computer science, Springer, 2005.
[11] 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.
[12] Poseidong for UML. http://www.gentleware.com/
[13] Yefei Zhao, YANG Zong-yuan, 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.
[14] Yefei Zhao, YANG Zong-yuan, 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.
[15] 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

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:

[16] 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.
[17] 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.
[18] Yefei Zhao, Zongyuan Yang, Jinkui Xie, Qiang Liu. Formal model and analysis of sliding window protocol based on NuSMV. Journal of Computers. May, 2009.
[19] 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.