A Model Abstraction Technique for Probabilistic Real-Time Systems Based on CEGAR for Timed Automata

Takeshi Nagaoka†, Akihiko Ito†, Toshiaki Tanaka†, Kozo Okano†, and Shinji Kusumoto†

†Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1–5, Suita City, Osaka, 565-0871, Japan

Abstract - Model checking techniques are considered as promising techniques for verification of information systems due to their ability of exhaustive checking. Well-known state explosion, however, might occur in model checking of large systems. Such explosion severely limits the scalability of model checking. In order to avoid it, several abstraction techniques have been proposed. Some of them are based on CounterExample-Guided Abstraction Refinement (CEGAR) technique proposed by E. Clarke *et al.*

This paper proposes a reachability analysis technique for probabilistic timed automata. In the technique, we abstract time attributes of probabilistic timed automata by applying our abstraction refinement technique for timed automata proposed in our previous work. Then, we apply probabilistic model checking to the generated abstract model which is just a markov decision process (MDP) with no time attributes. This paper also provides some experimental results on applying our method to IEEE 1394, FireWire protocol. Experimental results show our algorithm can reduce the number of states and total execution time dramatically compared to one of existing approaches.

Keywords: Probabilistic Timed Automaton, CEGAR, Model Checking, Real-time System, Formal Verification

1 INTRODUCTION

Model checking[1] techniques are considered as promising techniques for verification of information systems due to their ability of exhaustive checking. For verification of real-time systems such as embedded systems, timed automata are often used. On the other hand, probabilistic model checking[2]–[4] can evaluate performance, dependability and stability of information processing systems with random behaviors. In recent years, probabilistic models with real-time behaviors, called probabilistic timed automata (PTA) attract attentions. As well as traditional model checking techniques, however, state explosion is thought to be a major hurdle for verification of probabilistic timed automata.

Clarke *et al.* proposed an abstraction technique called CEGAR (CounterExample-Guided Abstraction Refinement)[5] shown in Fig. 1. In the CEGAR technique, we use a counter example (CE) produced by a model checker as a guide to refine abstracted models. A general CEGAR technique consists of several steps. First, it abstracts the original model (the obtained model is called abstract model) and performs model checking on the abstract model. Next, if a CE is found, it checks whether the CE is feasible on the concrete model or not. If the CE is spurious, it refines the abstract model. The last step is repeated until the valid output is obtained. In the CEGAR loop, an abstract model must satisfy the following property: if the abstract model satisfies a given specification, the concrete model also satisfies it.

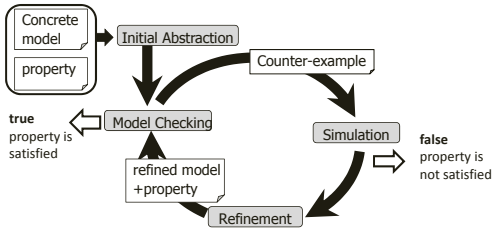

Figure 1: A General CEGAR Technique

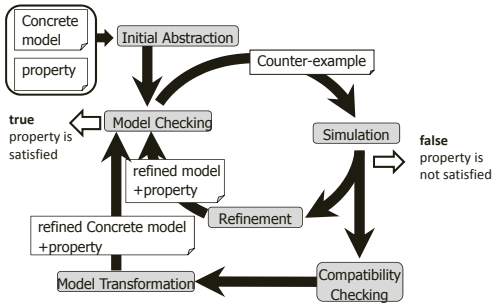

Figure 2: Our CEGAR Technique for Reachability Analysis of a Probabilistic Timed Automaton

In Paper[6], we have proposed an abstraction algorithm for timed automata based on CEGAR. In this algorithm, we generate finite transition systems as abstract models where all time attributes are removed. The refinement modifies the transition relations of the abstract model so that the model behaves correctly even if we don't consider the clock constraints.

This paper proposes a reachability analysis technique for probabilistic timed automata. In the technique, we abstract time attributes of probabilistic timed automata by applying our abstraction technique for timed automata proposed in Paper[6]. Then, we apply probabilistic model checking to the generated abstract model which is just a markov decision process (MDP) with no time attributes. The probabilistic model checking algorithm calculates a summation of occurrence probability of all paths which reach to a target state for reachability analysis. For probabilistic timed automata, however, we have to consider required clock constraints for such paths, and choose the paths whose required constraints are compatible. Since our abstract model does not consider the clock

constraints, we add a new flow where we check whether all paths used for probability calculation are compatible. Also, if they are not compatible, we transform the model so that we do not accept such incompatible paths simultaneously. The proposed procedure for the probabilistic timed automata is shown in Fig. 2.

This paper also provides some experimental results on applying our method to some examples. Experimental results show our algorithm can reduce the number of states and total execution time dramatically compared to one of existing approaches.

Several papers including Paper[3] have proposed probabilistic model checking algorithms. These algorithms, however, don't provide CEs when properties are not satisfied. Our proposed method provides a CE as a set of paths based on k-shortest paths search. This is a major contribution of our method. The proposed method also performs model checking considering compatibility problem. Few approaches resolve the compatibility problem. Paper [16] resolves the compatibility problem in a similar way to us. It, however, uses another approach (, which is based on a natural technique called predicate abstraction of clocks constraints) to abstract the models and the paper doesn't perform evaluation while our approach uses a quite simple abstraction technique, which remeves all of clock attributes, and this paper also shows the efficiency via performing experiments.

The organization of the rest paper is as follows. Sec.2 provides some definitions and lemmas as preliminaries. Sec.3 describes our proposed abstraction technique for the probabilistic timed automaton. Sec.4 gives some experimental results. Finally, Sec.5 concludes the paper and gives future works.

2 PRELIMINARY

This section gives some definitions about models used in this paper and also describes a general CEGAR technique.

2.1 Clock and Zone

Let C be a finite set of clock variables which take non-negative real values ($\mathbb{R}_{\geq 0}$). A map $\nu : C \to \mathbb{R}_{\geq 0}$ is called a clock assignment. The set of all clock assignments is denoted by $\mathbb{R}^C_{\geq 0}$. For any $\nu \in \mathbb{R}^C_{\geq 0}$ and $d \in \mathbb{R}_{\geq 0}$ we use $(\nu + d)$ to denote the clock assignment defined as $(\nu+d)(x) = \nu(x)+d$ for all $x \in C$. Also, we use $r(\nu)$ to denote the clock assignment obtained from ν by resetting all of the clocks in $r \subseteq C$ to zero.

Definition 2.1 (Differential Inequalities on C)**.** Syntax and semantics of a differential inequality E on a finite set C of clocks is given as follows:
$E ::= x - y \sim a \mid x \sim a$,
where $x, y \in C$, a is a literal of a real number constant, and $\sim \in \{\leq, \geq, <, >\}$. Semantics of a differential inequality is the same as the ordinal inequality.

Definition 2.2 (Clock Constraints on C)**.** Clock constraints $c(C)$ on a finite set C of clocks is defined as follows:
A differential inequality in on C is an element of $c(C)$.

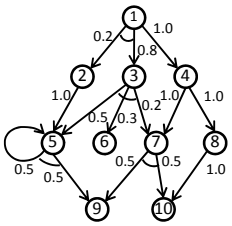

Figure 3: An Example of an MDP

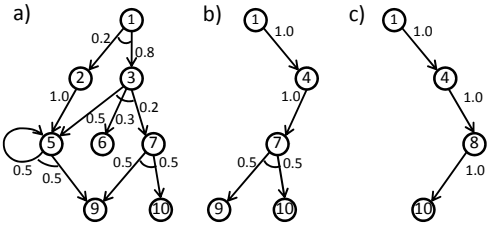

Figure 4: Examples of Adversaries

Let in_1 and in_2 be elements of $c(C)$, $in_1 \wedge in_2$ is also an element of $c(C)$.

A zone $D \in c(C)$ is described as a product of finite differential inequalities on clock set C, which represents a set of clock assignments that satisfy all the inequalities. In this paper, we treat a zone D as a set of clock assignments $\nu \in \mathbb{R}^C_{\geq 0}$ (For a zone D, $\nu \in D$ means the assignment ν satisfies all the inequalities in D).

2.2 Probability Distribution

A discrete probability distribution on a finite set Q is given as the function $\mu : Q \to [0, 1]$ such that $\Sigma_{q \in Q}\mu(q) = 1$. Also, $support(\mu)$ is a subset of Q such that $\forall q \in support(\mu).\mu(q) > 0$ holds.

2.3 Markov Decision Process

A Markov Decision Process (MDP)[7] is a markov chain with non-deterministic choices.

Definition 2.3 (Markov Decision Process)**.** A markov decision process MDP is 3-tuple $(S, s_0, Steps)$, where
S : a finite set of states;
$s_0 \in S$: an initial state; and
$Steps \subseteq S \times A \times Dist(S)$: a probabilistic transition relation where $Dist(S)$ is a probability distribution over S.

In our reachability analysis procedure, we transform a given PTA into a finite MDP, and perform probabilistic verification based on the Value Iteration[8] technique.

Figure 3 shows an example of an MDP. In the figure, probability distributions are associated with transitions. In the figure, transitions which belong to the same distribution are connected with a small arc at their source points. The MDP has several non-deterministic choices at the state 1 and 4. For example, at the state 1, we have two choices; 1) the control moves to the state 2 with the probability 0.2 and to the state

3 with the probability 0.8, 2) the control moves to the state 4 with the probability 1.0.

2.3.1 Adversary

An MDP has non-deterministic transitions called action. To resolve the non-determinism, an adversary is used. The adversary requires a finite path on an MDP, and decides a transition to be chosen at the next step.

Figure 4 shows examples of resolving the non-determinism of the MDP shown in Fig. 3 by some adversaries. Figure 4. a) is the case where we choose the action which moves to the state 2 or state 3 at the initial state 1. On the other hand, b) and c) are the cases where we choose the action which moves to the state 4 at the initial state 1. In the case of b), we choose the action which moves to the state 7 when we move from the state 1 to state 4. Also, in the case of c), we choose the action which moves to the state 8 in the same trace.

Here, if we want to obtain the reachability probability from the state 1 to the state 10, under the adversary of a), we can obtain the probability 0.08 ($= 0.8 \times 0.2 \times 0.5$), which is the minimum reachability probability. On the other hand, under the adversary of c), we can obtain the probability 1.0 ($= 1.0 \times 1.0 \times 1.0$), which is the maximum reachability probability.

2.3.2 Value Iteration

A representative technique of model checking for an MDP is Value Iteration[8]. The Value Iteration technique can obtain both of maximum and minimum probabilities of reachability and safety properties, respectively. At each state, Value Iteration can select an appropriate action according to the property to be checked. Therefore, the technique can produce the adversary as well as the probability.

2.4 Timed Automaton

Definition 2.4 (Timed Automaton)**.** A timed automaton $\mathscr{A}$ is a 6-tuple (A, L, l_0, C, I, T), where
A: a finite set of actions;
L: a finite set of locations;
$l_0 \in L$: an initial location;
C: a finite set of clocks;
$I \subset (L \to c(C))$: a mapping from locations to clock constraints, called a location invariant; and
$T \subset L \times A \times c(C) \times \mathscr{R} \times L$,
where $c(C)$ is a clock constraint, called guards;
$\mathscr{R} = 2^C$: a set of clocks to reset.

A transition $t = (l_1, a, g, r, l_2) \in T$ is denoted by $l_1 \xrightarrow{a,g,r} l_2$. A map $\nu : C \to \mathbb{R}_{\geq 0}$ is called a clock assignment. We define $(\nu + d)(x) = \nu(x) + d$ for $d \in \mathbb{R}_{\geq 0}$. $r(\nu) = \nu[x \mapsto 0], x \in r$, where $\nu[x \mapsto 0]$ means the valuation that maps x into zero, is also defined for $r \in 2^C$.

Definition 2.5 (Semantics of a Timed Automaton)**.** Given a timed automaton $\mathscr{A} = (A, L, l_0, C, I, T)$, let $S \subseteq L \times \mathbb{R}^C_{\geq 0}$ be a set of whole states of $\mathscr{A}$. The initial state of $\mathscr{A}$ shall be given as $(l_0, 0^C) \in S$.

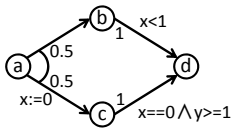

Figure 5: An Example of a PTA

For a transition $l_1 \xrightarrow{a,g,r} l_2 \ (\in T)$, the following two transitions are semantically defined. The former one is called an action transition, while the latter one is called a delay transition.

$$\frac{l_1 \xrightarrow{a,g,r} l_2, g(\nu), I(l_2)(r(\nu))}{(l_1, \nu) \overset{a}{\Rightarrow} (l_2, r(\nu))}, \qquad \frac{\forall d' \leq d \quad I(l_1)(\nu + d')}{(l_1, \nu) \overset{d}{\Rightarrow} (l_1, \nu + d)}$$

Definition 2.6 (A Semantic Model of a Timed Automaton)**.** For timed automaton $\mathscr{A} = (A, L, l_0, C, I, T)$, an infinite transition system is defined according to the semantics of $\mathscr{A}$, where the model begins with the initial state.

2.5 Probabilistic Timed Automaton

A PTA is a kind of a timed automaton extended with probabilistic behavior. Therefore, using the PTA, we can evaluate quantitative properties such as performance of information systems based on the probabilistic model checking technique. In the PTA, a set of probabilistic distributions is used instead of a set T of discrete transitions on the timed automaton.

Definition 2.7 (Probabilistic Timed Automaton)**.** A probabilistic timed automaton PTA is a 6-tuple $(A, L, l_0, C, I, prov)$, where
A: a finite set of actions;
L: a finite set of locations;
$l_0 \in L$: an initial location;
C: a finite set of clocks;
$I \subset (L \to c(C))$: a location invariant; and
$prob \subseteq L \times A \times c(C) \times Dist(2^C \times L)$: a finite set of probabilistic transition relations, where $c(C)$ represents a guard condition, and $Dist(2^C \times L)$ represents a finite set of probability distributions p. The Distribution $p(r, l) \in Dist(2^C \times L)$ represents the probability of resetting clock variables in r and also moving to the location l;

Figure 5 shows an example of a PTA. In the figure, from the location a, it moves to the location b with the probability 0.5 and also moves to the location c letting the value of the clock x reset to zero with the probability 0.5. Both of the arcs starting location a are connected with a small arc at their source points, which represents that they belong to the same probability distribution.

Definition 2.8 (Transitions of a Probabilistic Timed Automaton)**.** For $PTA = (A, L, l_0, C, I, prov)$, 6-tuple (l, a, g, p, r, l') represents a transition generated by a probabilistic distribution $(l, a, g, p) \in prob$ such that $p(r, l') > 0$.

Definition 2.9 (Semantics of a Probabilistic Timed Automaton)**.** Semantics of a probabilistic timed automaton $PTA = (A, L, l_0, C, I, prob)$ is given as a timed probabilistic system $TPS_{PTA} = (S, s_0, TSteps)$ where,

- $S \subseteq L \times \mathbb{R}^C$;
- $s_0 = (l_0, 0^C)$; and
- $TSteps \subseteq S \times A \cup \mathbb{R}_{\geq 0} \times Dist(S)$ is composed of action transitions and delay transitions.

  a) action transition
  if $a \in A$ and there exists $(l, a, g, p) \in prob$ such that $g(\nu)$ and $I(l')(r(\nu))$ for all $(r, l') \in support(p)$, $((l, \nu), a, \mu) \in TSteps$ where for all $(l', \nu') \in S$

$$\mu(l', \nu') = \sum_{r \subseteq C \wedge \nu' = r(\nu)} p(r, l').$$

  b) delay transition
  if $d \in \mathbb{R}_{\geq 0}$, and for all $d' \leq d$, $I(l)(\nu + d')$, $((l, \nu), d, \mu) \in TSteps$ where $\mu(l, \nu + d) = 1$.

The concrete delay in the delay transition can be decided non-deterministically on the semantics of a probabilistic timed automaton as well as those of a timed automaton.

In this paper, using a location l and a zone D, we describe a set of semantic states as $(l, D) = \{(l, \nu) \mid \nu \in D\}$.

A probabilistic timed automaton is said to be well-formed if a probabilistic edge can be taken whenever it is enabled[2]. Formally, a probabilistic timed automaton $PTA = (A, L, l_0, C, I, prob)$ is well-formed if

$$\forall (l, g, p) \in prob.\, \forall \nu \in \mathbb{R}^C_{\geq 0}.\, (g(\nu)) \rightarrow \forall (r, l) \in support(p).\, I(l)(r(\nu)).$$

In this paper, we assume that a given PTA is well-formed.

Definition 2.10 (Path on a Timed Probabilistic System)**.** A path ω with length of n on a timed probabilistic system $TPS_{PTA} = (S, s_0, TSteps)$ is denoted as follows.

$$\omega = (l_0, \nu_0) \xrightarrow{d_0, \mu_0} (l_1, \nu_1) \xrightarrow{d_1, \mu_1} \cdots \xrightarrow{d_{n-1}, \mu_{n-1}} (l_n, \nu_n)$$

, where $(l_0, \nu_0) = s_0$, $(l_i, \nu_i) \in S$ for $0 \leq i \leq n$ and $((l_i, \nu_i), d_i, \mu) \in TSteps \wedge ((l_i, \nu_i + d_i), 0, \mu_i) \in TSteps \wedge (l_{i+1}, \nu_{i+1}) \in support(\mu_i)$ for $0 \leq i \leq n-1$.

For model checking of a probabilistic timed automaton, we extract a number of paths and calculate a summation of their occurrence probabilities in order to check the probability of satisfying a given property. The important point is that we have to choose a set of paths which are compatible with respect to time elapsing.

Lemma 2.1 (Compatibility of two paths)**.** If two paths $\omega^\alpha = (l_0^\alpha, \nu_0^\alpha) \xrightarrow{d_0^\alpha, \mu_0^\alpha} (l_1^\alpha, \nu_1^\alpha) \xrightarrow{d_1^\alpha, \mu_1^\alpha} \cdots \xrightarrow{d_{n-1}^\alpha, \mu_{n-1}^\alpha} (l_n^\alpha, \nu_n^\alpha)$ and $\omega^\beta = (l_0^\beta, \nu_0^\beta) \xrightarrow{d_0^\beta, \mu_0^\beta} (l_1^\beta, \nu_1^\beta) \xrightarrow{d_1^\beta, \mu_1^\beta} \cdots \xrightarrow{d_{m-1}^\beta, \mu_{m-1}^\beta} (l_m^\beta, \nu_m^\beta)$ on a timed probabilistic system TPS_{PTA} satisfy the following predicate $isCompatible$, then ω^α and ω^β are said to be compatible.

$$isCompatible(\omega^\alpha, \omega^\beta) = \begin{cases} \text{true}, & \text{if } \forall i < min(n, m).\, l_i^\alpha = l_i^\beta \wedge d_i^\alpha = d_i^\beta \\ & \text{or there exists } i < min(n, m) \text{ such that} \\ & l_i^\alpha \neq l_i^\beta \wedge d_i^\alpha = d_i^\beta \wedge \\ & \forall j < i.\, l_j^\alpha = l_j^\beta \wedge d_j^\alpha = d_j^\beta \\ \text{false}, & \text{otherwise.} \end{cases}$$

Above predicate $isCompatible$ stands for that two paths are compatible if and only if one path is a prefix of the other, or same amount of delay is executed in both paths at the branching point of them.

Lemma 2.2 (Compatibility of a set of paths)**.** If a set Ω of paths on a timed probabilistic system TPS_{PTA} satisfies the following predicate $isCompatible$, then all of the paths over Ω are said to be compatible.

$$isCompatible(\Omega) = \begin{cases} \text{true}, & \text{if } \forall i \leq min(\Omega) \bigwedge_{\substack{\omega^\alpha, \omega^\beta \in \Omega \\ \wedge \omega^\alpha \neq \omega^\beta}} (l_i^\alpha = l_i^\beta \wedge d_i^\alpha = d_i^\beta) \\ & \text{or there exists } i \leq min(\Omega) \text{ such that} \\ & \bigwedge_{\substack{\omega^\alpha \omega^\beta \in \Omega \\ \wedge \omega^\alpha \neq \omega^\beta}} (l_i^\alpha \neq l_i^\beta \wedge d_i^\alpha = d_i^\beta \wedge \bigwedge_{j \leq i} (l_j^\alpha = l_j^\beta \wedge d_j^\alpha = d_j^\beta)), \\ & \text{and also } \bigwedge_{\substack{\Omega' \in 2^\Omega \wedge \\ \Omega' \neq \Omega \wedge |\Omega'| \leq 2}} isCompatible(\Omega') \\ \text{false}, & \text{otherwise.} \end{cases}$$

In Lemma2.2, we give the predicate $isCompatible$ for a set Ω of paths on a timed probabilistic system. In the lemma, we let paths in Ω be compatible if there is no contradiction with respect to time elapsing at the branching point of all the paths in Ω, and also if the compatibility is kept for every subset of Ω which contains more than two paths.

Next, we give a simple example of a pair of paths which does not satisfy the compatibility. In the Fig. 5, paths from the location a to d are given as $\omega^\alpha = (a, x = 0 \wedge y = 0) \xrightarrow{0, 0.5} (b, x = 0 \wedge y = 0) \xrightarrow{0, 1.0} (d, x = 0 \wedge y = 0)$ which reaches to d through b, and $\omega^\beta = (a, x = 0 \wedge y = 0) \xrightarrow{1, 0.5} (c, x = 0 \wedge y = 1) \xrightarrow{0, 1.0} (d, x = 0 \wedge y = 1)$ which reaches to d through c. In the path ω^α, we are required to let delay at the location a be less than one unit of time because of the guarded condition $x < 1$ of the transition between b and d. On the other hand, in the path ω^β, we are required to let delay at a be grater than or equal one unit of time because of the condition $x == 0 \wedge y \geq 1$ of the transition between c and d. Like the path ω^α and ω^β, if the required conditions of time elapsing at the branching point are contradict, we cannot use such paths simultaneously in the probability calculation.

2.6 CounterExample-Guided Abstraction Refinement

2.6.1 General CEGAR Technique

Since model abstraction sometimes over-approximates an original model, we may obtain spurious CEs which are infeasible

on the original model. Paper [5] gives an abstraction refinement framework called CEGAR (CounterExample-Guided Abstraction Refinement) (Fig. 1).

In the algorithm, at the first step (called Initial Abstraction), it generates an initial abstract model. Next, it performs model checking on the abstract model. In this step, if the model checker reports that the model satisfies a given specification, we can conclude that the original model also satisfies the specification, because the abstract model is an over-approximation of the original model. If the model checker reports that the model does not satisfy the specification, however, we have to check whether the CE detected is spurious or not in the next step (called Simulation). In the Simulation step, if we find that the CE is valid, we stop the loop. Otherwise, we have to refine the abstract model to eliminate the spurious CE, and repeat these steps until valid output is obtained.

2.6.2 CEGAR Technique for a Timed Automaton

In Paper[6], we have proposed the abstraction refinement technique for a timed automaton based on the framework of CEGAR. In this approach, we remove all the clock attributes from a timed automaton. If a spurious CE is detected by model checking on an abstract model, we transform the transition relation on the abstract model so that the model behaves correctly even if we don't consider the clock constraints. Such transformation obviously represents the difference of behavior caused by the clock attributes. Therefore, the finite number of application of the refinement algorithm enables us to check the given property without the clock attributes. Since our approach does not restore the clock attributes at the refinement step, the abstract model is always a finite transition system without the clock attributes.

3 PROPOSED APPROACH

In this section, we will present our abstraction refinement technique for a probabilistic timed automaton. In the technique, we use the abstraction refinement technique for a timed automaton proposed in Paper[6]. Though the probability calculated on the abstract model may be spurious because the abstract model has no time attributes, the finite number of applications of the refinement algorithm enables us to obtain correct results on the abstract model. In addition, we resolve the compatibility problem shown in Sec.2.5 by performing a backward simulation technique and generating additional location to distinguish the required condition for every incompatible path. Figure 2 shows our abstraction refinement framework. As shown in the figure, we add another flow where we resolve the compatibility problem.

Our abstraction requires a probabilistic timed automaton PTA and a property to be checked as its inputs. The property is limited by the PCTL formula $P_{<p}[\text{true } \mathbf{U}\ err]$. The formula represents a property that the probability of reaching to states where err (which means an error condition in general) is satisfied, is less than p.

In model checking techniques, several properties presented in CTL[9], LTL[10], and others would be checked in general. The typical properties, however, are safety and progress. The reachability analysis is the primitive procedure for safety checking, thus model checking problems on several important properties represented in CTL could be reduced into the reachability analysis problem. Therefore, the reachability analysis is important problem. On the other hand, the limitation of the properties that we can check derives from the abstraction technique proposed in Paper[6]. Since the technique of Paper[6] focuses on properties of reachability, in this paper we also focus on reachability properties only.

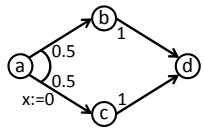

Figure 6: An Initial Abstract Model

3.1 Initial Abstraction

The initial abstraction removes all the clock attributes from a given probabilistic timed automaton as well as the technique in Paper[6]. The generated abstract model over-approximates the original probabilistic timed automaton. Also, the abstract model is just an MDP without time attributes.

Definition 3.1 (Abstract Model)**.** For a given probabilistic timed automaton $PTA = (A, L, l_0, C, I, prob)$, a markov decision process $M\hat{D}P_{PTA} = (\hat{S}, \hat{s}_0, \hat{Steps})$ is produced as its abstract model, where

- $\hat{S} = L$
- $\hat{s}_0 = l_0$
- $\hat{Steps} = \{\ (s, a, p) \mid (s, a, g, p) \in prob\ \}$

Figure 6 shows an initial abstract model for the PTA shown in Fig. 5 As shown in the figure, the abstract model is just an MDP where all of the clock constraints are removed though we keep a set of clock reset as a label of transitions.

3.2 Model Checking

In model checking, we apply Value Iteration[8] into the markov decision process obtained by abstraction and calculate a maximum reachability probability. Also, it decides an action to be chosen at every state as an adversary. If the obtained probability is less than p, we can terminate the CEGAR loop and conclude that the property is satisfied.

Although Value Iteration can calculate a maximum reachability probability, it cannot produce concrete paths used for the probability calculation. To obtain the concrete paths, we use an approach proposed in Paper[11] which can produce CE paths for PCTL formulas. The approach translates a probabilistic automaton into a weighted digraph. And we can obtain at most k paths by performing k-shortest paths search on the graph.

Definition 3.2 (Path on the Abstract Model)**.** A path $\hat{\omega}$ on an abstract model $M\hat{D}P_{PTA} = (\hat{S}, \hat{s}_0, \hat{Steps})$ for $PTA = (A, L, l_0, C, I, prob)$ is given as follows,

$$\hat{\omega} = \hat{s}_0 \overset{a_0,p_0,r_0}{\longrightarrow} \hat{s}_1 \overset{a_1,p_1,r_1}{\longrightarrow} \cdots \overset{a_{n-1},p_{n-1},r_{n-1}}{\longrightarrow} \hat{s}_n$$

, where $\hat{s}_i \in \hat{S}$ for $0 \leq i \leq n$ and $(\hat{s}_i, a_i, p_i) \in \hat{Steps} \wedge (r_i, \hat{s}_{i+1}) \in support(p_i)$ for $0 \leq i \leq n-1$.

As defined in Def. 3.2, we associate a set r of clock reset with a path on an abstract model in order to show the difference of r over the probabilistic distribution p.

For the abstract model shown in Fig. 6, Value Iteration outputs 1.0 as the probability that it reaches to the location d from the location a. On the other hand, k-shortest paths search ($k \geq 2$) detects two paths $\hat{\omega}^\alpha = a \overset{\tau,0.5,\{\}}{\longrightarrow} b \overset{\tau,1.0,\{\}}{\longrightarrow} d$ and $\hat{\omega}^\beta = a \overset{\tau,0.5,\{x:=0\}}{\longrightarrow} c \overset{\tau,1.0,\{\}}{\longrightarrow} d$, where τ represents a label for transitions with no label in the figure.

3.3 Simulation

Simulation checks whether all the paths obtained by k-shortest paths search are feasible or not on the original probabilistic timed automaton. We use the simulation algorithm proposed in Paper[6] where we use some operations of DBM (Difference Bound Matrix)[12] to obtain zones which are reachable from the initial state. If there is at least one path which is infeasible on the original PTA, we proceed to the abstraction refinement step.

Figure 7 shows the simulation results for two paths $\hat{\omega}^\alpha$ and $\hat{\omega}^\beta$. Simulation concludes that the two paths are feasible on the original PTA.

3.4 Abstraction Refinement

In this step, we refine the abstract model so that the given spurious CE also becomes infeasible on the refined abstract model. We can use the algorithm proposed in Paper[6]. Since the algorithm of Paper[6] performs some operations on transitions of a timed automaton, we replace such operations by those on probability distributions of a probabilistic timed automaton.

3.5 Compatibility Checking

When all the paths obtained by k-shortest paths search are feasible and a summation of occurrence probabilities of them is greater than p, we also have to check whether all the paths are compatible or not. In this compatibility checking step, at each location of the paths, we have to obtain a condition (zone) which is reachable from the initial state and also reachable to the last state along with the path. Next, we check the compatibility of such conditions among all paths. To obtain such conditions, we have to perform both forward simulation shown in Sec. 3.3 and backward simulation for each path, and merge the results. For the result of forward simulation, we can reuse the result obtained in the Simulation step. Then we check the compatibility based on Lemma 2.2.

Algorithm 1 BackwardSimulation(PTA, ω)

```
1: /* PTA = (A, L, l_0, C, I, prob)
      ω̂ = ŝ_0 -a_0,p_0,r_0-> ŝ_1 -a_1,p_1,r_1-> ... -a_{n-1},p_{n-1},r_{n-1}-> ŝ_n */
2: D^ω̂_{b,n} := I(ŝ_n)
3: for i := n − 1 downto 0 do
4:     D^ω̂_{b,i} := D^ω̂_{b,i+1}
5:     D^ω̂_{b,i} := down(D^ω̂_{b,i})              /* reverse the time elapse */
6:     D^ω̂_{b,i} := and(D^ω̂_{b,i}, I(ŝ_{i+1}))
7:     D^ω̂_{b,i} := free(D^ω̂_{b,i}, r_i)         /* remove all constraints on r_i */
8:     D^ω̂_{b,i} := and(D^ω̂_{b,i}, g_i)          /* (ŝ_i, a_i, g_i, p_i) ∈ prob */
9:     D^ω̂_{b,i} := and(D^ω̂_{b,i}, I(ŝ_i))
10: end for
11: return D^ω̂_b
```

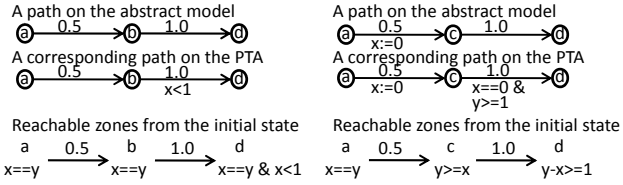

Figure 7: Simulation Results for a Set of Paths

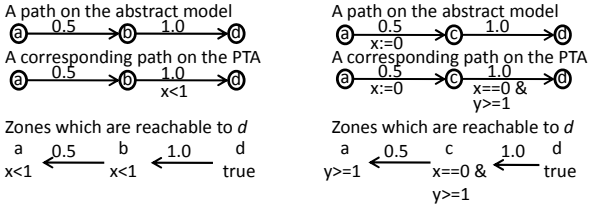

Figure 8: Results of Backward Simulation for a Set of Paths

3.5.1 Backward Simulation

Algorithm 1 implements the backward simulation. Functions and, $free$, $down$ used in the algorithm are operation functions on a zone, and are defined in Paper[12]. Formally, for a zone D, a constraint c, and a set r of clock reset, those functions are defined as follows; $and(D, c) = \{u \mid u \in D \wedge u \in c\}$, $free(D, r) = \{u \mid r(u) \in D\}$, and $down(D) = \{u \mid u + d \in D \wedge d \in \mathbb{R}_{\geq 0}\}$

Figure 8 shows results of backward simulation for two paths $\hat{\omega}^\alpha$ and $\hat{\omega}^\beta$ detected in Sec. 3.2.

3.5.2 Determination of Compatibility

In this step, we check compatibility of the set $\hat{\Omega}$ of paths on the abstract model using the required conditions obtained by both of forward and backward simulation. Algorithm 2 checks the compatibility of $\hat{\Omega}$ using the Algorithm 3.

Algorithm 3 first checks whether the required conditions of the i-th locations for each path are compatible or not ($l2$-$l8$) using the results of forward and backward simulation. Next,

Algorithm 2 IsCompatible(PTA, $\hat{\Omega}$, D_f, D_b)

```
1: /* PTA = (A, L, l_0, C, I, prob), Ω̂ is a set of abstract paths,
      and D_f and D_b are sets of forward and backward simulation
      results for each path ω̂ ∈ Ω̂,respectively. */
2: return CompatibleCheck(PTA, Ω̂, D_f, D_b, 0)
```

Algorithm 3 CompatibleCheck(PTA, $\hat{\Omega}$, D_f, D_b, i)

1: $D' := true$
2: **foreach** $\hat{\omega} \in \hat{\Omega}$ such that $length(\hat{\omega}) \geq i$ **do**
3: $D^{\hat{\omega}}_{c,i} := D^{\hat{\omega}}_{f,i} \cap D^{\hat{\omega}}_{b,i}$
4: $D' := D' \cap D^{\hat{\omega}}_{c,i}$
5: **if** $D' = \emptyset$ **then**
6: **return** $false$
7: **end if**
8: **end for**
9: $S^{\hat{\Omega}}_{i+1} := SplitPathSet(\hat{\Omega}, i+1)$
10: /* split $\hat{\Omega}$ into a set of its subsets without overlap with respect to the i+1-th location and clock reset for each path in $\hat{\Omega}$ */
11: **foreach** $\hat{\Omega}' \in S^{\hat{\Omega}}_{i+1}$ such that $|\hat{\Omega}'| \geq 2$ **do**
12: **if** $CompatibleCheck(PTA, \hat{\Omega}', D, i+1) = false$ **then**
13: **return** $false$
14: **end if**
15: **end for**
16: **return** $true$

Algorithm 4 SplitPathSet($\hat{\Omega}$, i)

1: $S := \emptyset$
2: **foreach** $\hat{\omega} \in \hat{\Omega}$ **do**
3: /* $\hat{\omega} = \hat{s}_0 \overset{a_0,p_0,r_0}{\rightarrow} \hat{s}_1 \overset{a_1,p_1,r_1}{\rightarrow} \cdots \overset{a_{n-1},p_{n-1},r_{n-1}}{\rightarrow} \hat{s}_n$ */
4: **if** $\hat{\Omega}_{r_{i-1},\hat{s}_i} \notin S$ **then**
5: $\hat{\Omega}_{r_{i-1},\hat{s}_i} := \{\hat{\omega}\}$
6: $S := S \cup \hat{\Omega}_{r_{i-1},\hat{s}_i}$
7: **else**
8: $\hat{\Omega}_{r_{i-1},\hat{s}_i} := \hat{\Omega}_{r_{i-1},\hat{s}_i} \cup \{\hat{\omega}\}$
9: **end if**
10: **end for**
11: **return** S

the algorithm divides $\hat{\Omega}$ into some subsets of it based on the (i+1)-th locations and the set of clock reset for each path (l9). Then, it checks the compatibility for the following sequences of paths by applying the algorithm into the divided subsets recursively (l11-l15). Although the predicate $isCompatible$ in the Lemma 2.2 checks the compatibility for each subset of Ω, the algorithm omit redundant checks by dividing Ω based on the branches of the paths.

For the path $\hat{\omega}^{\alpha}$ in Sec. 3.2, zones at a which is reachable from initial state and which can move to d are given as $D^{\hat{\omega}^{\alpha}}_{f,0} = (x == y)$, and $D^{\hat{\omega}^{\alpha}}_{b,0} = (x < 1)$, respectively. Also, a zone of the product of them is given as $D^{\hat{\omega}^{\alpha}}_{c,0} = (x == y \wedge x < 1)$. Similarly, for the path $\hat{\omega}^{\beta}$, the product zone is given as $D^{\hat{\omega}^{\beta}}_{c,0} = (x == y \wedge y > 1)$. Since $D^{\hat{\omega}^{\alpha}}_{c,0}$ and $D^{\hat{\omega}^{\beta}}_{c,0}$ contradict each other, we can conclude that the paths $\hat{\omega}^{\alpha}$ and $\hat{\omega}^{\beta}$ are incompatible each other.

3.6 Model Transformation

When the compatibility check procedure decides a given set $\hat{\Omega}$ of paths is incompatible at i-th location, our proposed algorithm resolves the incompatibility by refining behaviors from the i-th location. Our algorithm uses $D^{\hat{\omega}}_{c}$ which is a product of results of forward and backward simulation for a path $\hat{\omega} \in \hat{\Omega}$. It duplicates locations which are reachable from the zone $D^{\hat{\omega}}_{c,i}$ by an action associated with the i-th distribution p_i. Also it constructs transition relations so that the transformation becomes equivalent transformation. For example, transition relations from a duplicated location are duplicated if the relations are executable from the invariant associated with the duplicated location.

Algorithm 5 TransformPTA(PTA, D_c, $\hat{\Omega}$, i)

1: $D_{complement}$:= true
2: **foreach** $\hat{\omega} \in \hat{\Omega}$ **do**
3: $L_{dup} := DuplicateLocation(PTA, \hat{\omega}, D^{\hat{\omega}}_{c,i}, i)$
4: $L := L \cup L_{dup}$
5: $prob_{dup} := DuplicateDistribution(PTA, \hat{\omega}, L_{dup}, i)$
6: $prob := prob \cup prob_{dup}$
7: $D_{complement} := D_{complement} \cap D^{\omega}_{c,i}$
8: **end for**
9: $L_{dup} := DuplicateLocation(PTA, \hat{\omega}, D_{complement}, i)$
10: $L := L \cup L_{dup}$
11: $prob_{dup} := DuplicateDistribution(PTA, \hat{\omega}, L_{dup}, i)$
12: $prob := prob \cup prob_{dup}$
13: $prob := RemoveDistribution(PTA, \hat{s}_i, p_i)$
14: /* for all path $\hat{\omega} \in \hat{\Omega}$, the i-th state $\hat{s}_i$ and i-th probability distribution is p_i */
15: **return** PTA

Algorithm 6 DuplicateLocation($PTA, \hat{\omega}, D, i$)

1: /* $PTA = (A, L, l_0, C, I, prob)$
$\hat{\omega} = \hat{s}_0 \overset{a_0,p_0,r_0}{\rightarrow} \hat{s}_1 \overset{a_1,p_1,r_1}{\rightarrow} \cdots \overset{a_{n-1},p_{n-1},r_{n-1}}{\rightarrow} \hat{s}_n$ */
2: $L_{dup} := \emptyset$
3: **foreach** $(l, r) \in L \times 2^C$ such that $p_i(l, r) > 0$ **do**
4: $(l, D) := Succ((\hat{s}_i, D), e)$
5: /* $succ$ returns a successor state set through a given edge e, and $e = (\hat{s}_i, a_i, g, p_i, r, l)$ */
6: $l_{dup} := newLocation()$
7: $I(l_{dup}) := D$
8: $L_{dup} = l_{dup}$
9: **end for**
10: **return** L_{dup}

Algorithm 5 transforms a given PTA with considering its compatibility. The algorithm calls $DuplicateLocation$ (Algorithm 6) which duplicates locations, $DuplicateDistribution$ (Algorithm 7) which duplicates probabilistic transitions, and $RemoveDistribution$ (Algorithm 9) which removes probabilistic transitions. The procedure $Succ$ in Algorithms 6 and 8 calculates a successor state set from a given state set S through a given edge $e = (l, a, g, p, r, l')$, i.e. $Succ(S, e) = \{(l', r(\nu) + d) \mid (l, \nu) \in S \wedge g(\nu) \wedge I(l')(r(\nu)) \wedge \forall d' \leq d.I(l')(r(\nu) + d')\}$

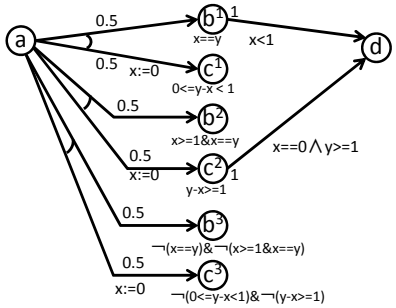

Figure 9: A Transformed PTA

Algorithm 7 DuplicateDistribution($PTA, \hat{\omega}, L_{dup}, i$)

/* $PTA = (A, L, l_0, C, I, prob)$
$\hat{\omega} = \hat{s}_0 \overset{a_0,p_0,r_0}{\rightarrow} \hat{s}_1 \overset{a_1,p_1,r_1}{\rightarrow} \cdots \overset{a_{n-1},p_{n-1},r_{n-1}}{\rightarrow} \hat{s}_n$ */
$prob_{dup} := \emptyset$
$p_{dup} := newDistribution()$
/* generate a new distribution over $L \times 2^C$ */
foreach $(l, r) \in L \times 2^C$ **do**
  $p_{dup}(l_{dup}, r) := p_i(l, r)$
  /* l_{dup} is a duplicate location of l generated by DuplicateLocation algorithm */
end for
$prob_{dup} := Prob_{dup} \cup \{(\hat{s}_i, a_i, g, p_{dup})\}$
/* $(\hat{s}_i, a_i, g, p_i) \in prob$ */
foreach $l_{dup} \in L_{dup}$ **do**
  $prob_{dup} := Prob_{dup} \cup$
  $DuplicateDistFromDupLoc(PTA, l_{dup})$
end for
return p_{dup}

Algorithm 8 DuplicateDistFromDupLoc(PTA, l_{dup})

/* $PTA = (A, L, l_0, C, I, prob)$, and let l be an original location of l_{dup} */
$prob_{dup} := \emptyset$
foreach $(l, a, g, p) \in Prob$ **do**
  $f_{dup} :=$ true, $p_{dup} := newDistribution()$
  foreach $(l', r) \in L \times 2^C$ **do**
    if $Succ((l, I(l_{dup})), e) \neq \emptyset$ **then**
      /* $e = (l, a, g, p, r, l')$ */
      $p_{dup}(l', r) = p(l, r)$
    else
      $f_{dup} :=$ false
      break
    end if
  end for
  if f_{dup} **then**
    /* duplicate the distribution if it is executable from the duplicate location */
    $Prob_{dup} := Prob_{dup} \cup \{(l, a, g, p_{dup})\}$
  end if
end for

Figure 9 shows the transformed PTA by applying the model transformation procedure for the paths $\hat{\omega}^\alpha$ and $\hat{\omega}^\beta$. The locations b^1 and c^1 are duplicated locations based on the path $\hat{\omega}^\alpha$ and the zone $D^{\hat{\omega}^\beta}_{c,0} = (x == y \wedge x < 1)$ on the location a. We associate invariants to b^1 and c^1 based on zones which are reachable from $D^{\hat{\omega}^\beta}_{c,0}$ through transitions from a to b, and from a to c, respectively. Also, we duplicate a transition from b to d as the transition from b^1 to d because the transition is feasible from the invariant of b^1. On the other hand, we do not duplicate a transition from c to d because the transition is not feasible from the invariant of c^1. Similarly, locations b^2 and c^2 are duplicated locations based on the path $\hat{\omega}^\beta$ and the zone $D^{\hat{\omega}^\beta}_{c,0}$. Locations b^3 and c^3 are generated as complements of the invariant associated with each duplicated location in order to preserve the equivalence.

By transforming the original PTA in such a way, if we remove all clock constraints from the model in Fig. 9, Value Iteration on its abstract model outputs 0.5 as the maximum probability.

Algorithm 9 RemoveDistribution(PTA, l, p)

/* $PTA = (A, L, l_0, C, I, prob)$, and let l be an original location of l_{dup} */
foreach (l, a, g, p) **do**
  $prob := prob \setminus \{(l, a, g, p)\}$
end for
return $prob$

4 EXPERIMENTS

We have implemented a prototype of our proposed approach with Java, and performed some experiments. Though the prototype can check the compatibility of a given set of paths, currently it cannot deal with the model transformation.

The prototype performs k-shortest paths search and simulation concurrently in order to reduce execution time. By implementing the algorithms concurrently, we have not to wait until all of k paths are detected, i.e. if a path is detected by the k-shortest paths search algorithm, we can immediately apply simulation and (if needed) abstraction refinement procedures.

Also, our prototype continues the k-shortest search algorithm when a spurious CE is detected and the refinement algorithm is applied. If other paths which do not overlap with the previous spurious CEs, are detected, we can apply simulation and refinement algorithms to it again. This helps us reduce the number of CEGAR loop.

4.1 Goals of the Experiments

In this experiment, we evaluated the performance of our proposed approach with regard to execution time, memory consumption, and qualities of obtained results. As a target for comparison, we chose the approach of Digital Clocks[3] where they approximate clock evaluations of a PTA by integer values.

4.2 Example

We used a case study of the FireWire Root Contention Protocol[13] as an example for this experiment. This case study concerns the Tree Identify Protocol of the IEEE 1394 High Performance Serial Bus (called "FireWire") which takes place when a node is added or removed from the network. In the experiment, we checked the probability that a leader is not selected within a given deadline. The probabilistic timed automaton for the example is composed of two clock variables, 11 locations, and 24 transitions.

4.3 Procedure of the Experiments

In this experiment, we checked the property that "the probability that a leader cannot be elected within a given $deadline$ is less than p." We considered three scenarios where the parameter $deadline$ is 5, 10, 20 μs, respectively. Also, for each scenario, we conducted two experiments where the value of p is 1.5 times as an approximate value of the maximum probability obtained by the Digital Clocks approach[3] and a half of it, respectively. In the proposed approach, we searched at most 5000 paths by letting the parameter k of the k-shortest

Table 1: Experimental Result

		Digital Clocks[3]				Proposed Approach				
$D(\mu s)$	p	$Result$	$Time(s)$	$State$	$MEM(MB)$	$Result$	$Time(s)$	$Loop$	$State$	$Heap(MB)$
5	1.09×10^{-1}	false	20.90	297,232	10.2	false	4.19	10	37	8.0
	3.28×10^{-1}	true	20.89	297,232	10.2	true	3.60	9	36	8.0
10	1.26×10^{-2}	false	54.80	685,232	21.7	false	8.16	19	134	8.0
	3.79×10^{-2}	true	54.82	685,232	21.7	true	6.57	15	115	8.0
20	1.85×10^{-4}	false	176.93	1,461,232	41.0	false	1186.08	47	477	64.0
	5.56×10^{-4}	true	177.46	1,461,232	41.0	true	31.32	32	435	8.0

Table 2: Analysis of Counter Example Paths

$D(\mu s)$	p	$Path$	$Probability$	$CC(ms)$
5	1.0938×10^{-1}	7	1.2500×10^{-1}	0.7
10	1.2635×10^{-2}	43	1.2695×10^{-2}	5.9
20	1.8500×10^{-4}	2534	1.8501×10^{-4}	296.9

paths search algorithm be 5000. For evaluation of existing approach, we used the probabilistic model checker PRISM[14].

The experiments were performed under Intel Core2 Duo 2.33 GHz, 2GB RAM, and Fedora 12 (64bit).

4.4 Results of the Experiments

The results are shown in Table 1. The column of D means the value of $deadline$. For each approach, columns of $Results$, $Time$, and $States$ show the results of model checking, execution time of whole process, and the number of states constructed, respectively. The column MEM in the columns of the Digital Clocks shows the memory consumption of PRISM. The columns $Loop$ and $Heap$ in the columns of the proposed approach show the number of CEGAR loops executed and the maximum heap size of the Java Virtual Machine (JVM) which executes our prototype, respectively.

Table 1 shows that for all cases we can dramatically reduce the number of states and obtain correct results. Moreover, we can reduce the execution time more than 80 percent except for the case when $deadline = 20\mu s$ and $p = 1.85 \times 10^{-4}$. In this case, however, the execution time drastically increases.

Table 2 shows the results of analysis of CE paths obtained when the results of model checking are false. The columns of $Path$, $Probability$ and CC show the number of CE paths, the summation of occurrence probability of them, and execution time for compatibility checking, respectively. For this example, the obtained sets of CE paths are compatible in every case.

4.5 Discussion

From the results shown in Table 1, we can see that our proposed approach is efficient with regard to both execution time and the number of states. Especially, the number of states decrease dramatically. The execution time is also decreased even though we perform model checking several times shown in the column of $Loop$.

On the other hand, in the case when $deadline = 20\mu s$ and $p = 1.85 \times 10^{-4}$, the execution time increases drastically. We think that as shown in Table 2 we have to search 2534 paths and this causes the increase of execution time especially for k-shortest paths search. A more detailed analysis shows that the execution time for k-shortest paths search accounts for 1123 seconds of total execution time of 1186 seconds. Also, the results shows that the JVM needs 64MB as its heap size in this case. This is because compatibility checking for 2534 of paths needs a large amount of the memory. From the results, we have to resolve a problem of the scalability when the number of candidate paths for a CE becomes large.

5 CONCLUSION

This paper proposed the abstraction refinement technique for a probabilistic timed automaton by extending the existing abstraction refinement technique for a timed automaton.

Future work includes completion of implementation. General DBM does not support not operator[15]; so we have to investigate efficient algorithms for the not operator.

ACKNOWLEDGMENTS

This work is being conducted as a part of Stage Project, the Development of Next Generation IT Infrastructure, supported by Ministry of Education, Culture, Sports, Science and Technology, as well as Grant-in-Aid for Scientific Research C(21500036), as well as grant from The Telecommunications Advancement Foundation.

REFERENCES

[1] E. M. Clarke, O. Grumberg and D. A. Peled, editors, "Model Checking," MIT Press (1999).

[2] M. Kwiatkowska, G. Norman, J. Sproston and F. Wang, "Symbolic model checking for probabilistic timed automata," Information and Computation, Vol. 205, No. 7, pp. 1027–1077 (2007).

[3] M. Kwiatkowska, G. Norman and J. Sproston, "Performance Analysis of Probabilistic Timed Automata Using Digital Clocks," Formal Methods in System Design, Vol. 29, No. 1, pp. 33–78 (2006).

[4] M. Kwiatkowska, G. Norman and D. Parker, "Stochastic Games for Verification of Probabilistic Timed Automata," Proc. of the 7th Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS'09), Vol. 5813 of *LNCS*, pp. 212–227 (2009).

[5] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and V. Helmut, "Counterexample-guided Abstraction Refinement

for Symbolic Model Checking," Journal of the ACM, Vol. 50, No. 5, pp. 752–794 (2003).
[6] T. Nagaoka, K. Okano and S. Kusumoto, "An Abstraction Refinement Technique for Timed Automata Based on Counterexample-Guided Abstraction Refinement Loop," IEICE Transactions on Information and Systems, Vol. E93-D, No. 5, pp. 994–1005 (2010).
[7] C. Derman, editor, "Finite-State Markovian Decision Processes," New York: Academic Press (1970).
[8] D. P. Bertsekas, "Dynamic Programming and Optimal Control," Athena Scientific (1995).
[9] E. Clarke, E. Emerson and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logics," ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, pp. 244–263 (1986).
[10] A. Pnueli, "The temporal logic of programs," Proc. of the 18th Int. Symp. on Foundation of Computer Science (FOCS), pp. 46–57 (1977).
[11] H. Aljazzar and S. Leue, "Directed Explicit State-Space Search in the Generation of Counterexamples for Stochastic Model Checking," IEEE Transactions on Software Engineering, Vol. 36, No. 1, pp. 37–60 (2010).
[12] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms and Tools," Lecture Notes on Concurrency and Petri Nets, Vol. 3098, pp. 87–124 (2004).
[13] M. Kwiatkowska, G. Norman and J. Sproston, "Probabilistic Model Checking of Deadline Properties in the IEEE1394 Firewire Root Contention Protocol," Formal Aspects of Computing, Vol. 14, No. 3, pp. 295–318 (2003).
[14] A. Hinton, M. Kwiatkowska, G. Norman and D. Parker, "PRISM: A Tool for Automatic Verification of Probabilistic Systems," Proc. of the 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06), Vol. 3920 of *LNCS*, pp. 441–444 (2006).
[15] A. David, J. Hakansson, K G. Larsen and P. pettersson, "Model Checking Timed Automata with Priorities using DBM Subtraction," Proc. of the 4th Int. Conf. on Formal Modelling and Analysis of Timed Systems, pp. 128–142 (2006).
[16] A. Morimoto, R. Komagata and S. Yamane, "Probabilistic Timed CEGAR," Technical report of IEICE CST 109 (73), pp. 25–30 (2009).

(Received September 2, 2010)
(Revised April 24, 2011)

Takeshi Nagaoka received the MI, and Ph.D degrees in Computer Science from Osaka University in 2007, and 2011, respectively. His research interests include abstraction techniques in model checking, especially a timed automaton and a probabilistic timed automaton.

Akihiko Ito received the BE and MI degrees in Computers Sciences from Hiroshima University and Osaka University in 2008 and 2010, respectively. His research interests include model checking, especially timed automaton and probabilistic timed automaton.

Toshiaki Tanaka received the BE and MI degrees in Computer and Systems Engineering from Kobe University, and Osaak University in 2009, and 2011, rewspectively. His research interests include parallelization of model checking, especially a timed automaton.

Kozo Okano received the BE, ME, and Ph.D degrees in Information and Computer Sciences from Osaka University, in 1990, 1992, and 1995, respectively. Since 2002 he has been an associate professor in the Graduate School of Information Science and Technology, Osaka University. In 2002, he was a visiting researcher of the Department of Computer Science, University of Kent at Canterbury. In 2003, he was a visiting lecturer at the School of Computer Science, University of Birmingham. His current research interests include formal methods for software and information system design. He is a member of IEEE, IEICE of Japan and IPS of Japan.

Shinji Kusumoto received the BE, ME, and DE degrees in information and computer sciences from Osaka University in 1988, 1990, and 1993, respectively. He is currently a professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include software metrics and software quality assurance technique. He is a member of the IEEE, the IEEE Computer Society, IPSJ, IEICE, and JFPUG.