Reachability Analysis of Probabilistic Timed Automata Based on an Abstraction Refinement Technique

Takeshi Nagaoka, Akihiko Ito, Toshiaki Tanaka, Kozo Okano, Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University
{t-nagaok,a-ito,tstanaka,okano,kusumoto}@ist.osaka-u.ac.jp

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. 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 our abstraction technique 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. Also, our technique can produce a counter example as a set of paths when a given model does not satisfy a specification. The 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, 3, 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]. In the CEGAR technique, we use a counter example (CE) produced by a model checker as a guide to refine abstracted models. In [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 [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 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.

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 [2, 3, 4] 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. Our approach 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

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. *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 $c(C)$ on a finite set C of clocks is defined as follows: A differential inequality in on C is an element of $c(C)$. 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. *A markov decision process MDP is 3-tuple $(S, s_0, Steps)$, where S is a finite set of states, $s_0 \in S$ is an initial state, and $Steps \subseteq S \times A \times Dist(S)$ is 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.

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.

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 obtain the adversary as well as the probability.

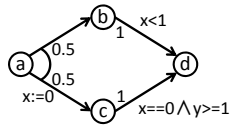

Figure 1: An Example of a PTA

2.4 Timed Automaton

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

DEFINITION 2.5. *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 $(l_0, 0^C) \in S$. For a transition $(l_1, a, g, r, l_2) \in T$, the following two transitions are semantically defined. The former one is called an action transition, while the latter one is called a delay transition.*

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

DEFINITION 2.6. *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. 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. *A probabilistic timed automaton PTA is a 6-tuple $(A, L, l_0, C, I, prov)$, where A is a finite set of actions, L is a finite set of locations, $l_0 \in L$ is an initial location, C is a finite set of clocks, $I \subset (L \to c(C))$ is a location invariant and $prob \subseteq L \times A \times c(C) \times Dist(2^C \times L)$ is a finite set of probabilistic transitions, 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 1 shows an example of a PTA. In the figure, from the location a, the control 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.

DEFINITION 2.8. *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, $S \subseteq L \times \mathbb{R}^C$ is a set of states, $s_0 = (l_0, 0^C)$ is an initial state, and $TSteps \subseteq S \times A \cup \mathbb{R}_{\geq 0} \times Dist(S)$ is composed of action transitions and delay transitions, where*

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* $\mu(l, \nu + d) = 1$.

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

DEFINITION 2.9. *A path* ω *with length of* n *on a timed probabilistic system* $TPS_{PTA} = (S, s_0, TSteps, L')$ *is denoted as follows.*

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

, where $(l_i, \nu_i) \in S$ *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. *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.*

$isCompatible(\Omega) =$

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

In Lemma2.1, 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.

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.2).

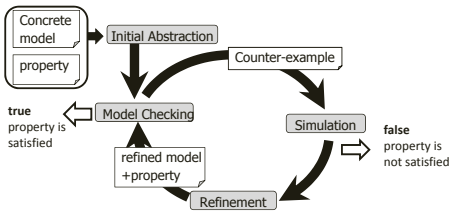

Figure 2: A General CEGAR Technique

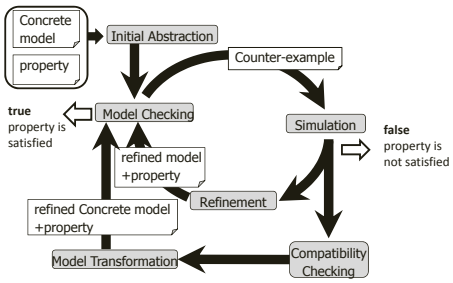

Figure 3: Our CEGAR Technique for Reachability Analysis of a Probabilistic Timed Automaton

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 [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

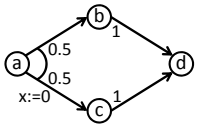

Figure 4: An Initial Abstract Model

in [6]. 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 3 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.

3.1 Initial Abstraction

The initial abstraction removes all the clock attributes from a given probabilistic timed automaton as well as the technique in The generated abstract model over-approximates the original probabilistic timed automaton.

DEFINITION 3.1. *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* $\hat{S} = L$*,* $\hat{s}_0 = l_0$*, and* $\hat{Steps} = \{\ (s, a, p)\ |\ (s, a, g, p) \in prob\ \}$

Figure 4 shows an initial abstract model for the PTA shown in Fig.1 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 [9] 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. *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,*

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

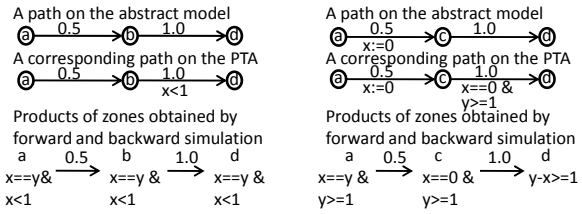

Figure 5: Products of zones obtained by Forward and Backward Simulation

, *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.4, Value Iteration outputs 1.0 as the probability that it reaches to the state d from the state a. On the other hand, k-shortest paths search ($k \geq 2$) detects two paths $\hat{\omega}^\alpha = a \xrightarrow{\tau, 0.5, \{\}} b \xrightarrow{\tau, 1.0, \{\}} d$ and $\hat{\omega}^\beta = a \xrightarrow{\tau, 0.5, \{x:=0\}} c \xrightarrow{\tau, 1.0, \{\}} d$, 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 [6] If there is at least one path which is infeasible on the original PTA, we proceed to the abstraction refinement step.

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 [6]. Since the algorithm of [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 2.1. Paper[10] shows the algorithm of compatibility checking in detail. In the algorithm, we check the compatibility of such products of zones at every location of the paths.

Figure 5 shows the products of zones obtained by both forward and backward simulation for two paths $\hat{\omega}^\alpha$ and $\hat{\omega}^\beta$ in Sec.3.2. For the path $\hat{\omega}^\alpha$, the product zone at a is given as $D^{\hat{\omega}^\alpha}_{c,0} = (x == y \wedge x <$

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

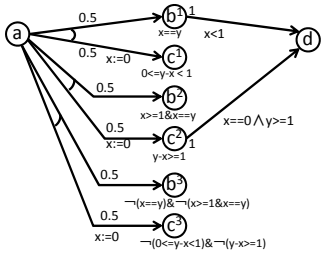

Figure 6: A Transformed PTA

1), which means a zone which is reachable from the initial state and also can move to d. 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. Detailed Algorithms to transform the model are given in [10].

Figure 6 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.6, Value Iteration on its abstract model outputs 0.5 as the maximum probability.

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 Goal of the Experiments

The goal of this experiment is to check that our approach, which also searches a CE, is performed within realistic time. 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], which is considered as a basic approach, 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[11] 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 paths search algorithm be 5000. For evaluation of existing approach, we used the probabilistic model checker PRISM[12].

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

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. The main contribution of this work is generation of a CE. Also, the experimental results show the efficiency of our technique compared to one of existing approaches.

Future work includes completion of implementation. General DBM does not support not operator[13]; 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.

6. 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*, 205(7):1027–1077, 7 2007.

[3] M. Kwiatkowska, G. Norman, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Int. Journal on Formal Methods in System Design*, 29(1):33–78, 7 2006.

[4] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In *Proc. of the 7th Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *LNCS*, pages 212–227, 9 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*, 50(5):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*, E93-D(5):994–1005, 5 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] H. Aljazzar and S. Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Transactions on Software Engineering*, 36(1):37–60, 1 2010.

[10] A. Ito, T. Nagaoka, K. Okano, and S. Kusumoto. Reachability analysis for probabilistic timed system based on timed abstraction refinement technique (in japanese). In *IEICE Technical Report*, volume 109, pages 85–90, 3 2010.

[11] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the ieee1394 firewire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 4 2003.

[12] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker:. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of the 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920, pages 441–444, 2006.

[13] A. David, J. Hakansson, K G. Larsen, and P. pettersson. Model checking timed automata with priorities using dbm subtraction. In *Proc. of the 4th Int. Conf. on Formal Modelling and Analysis of Timed Systems*, pages 128–142, 2006.