2011 Second International Conference on Networking and Computing

Clock Number Reduction Abstraction on CEGAR Loop Approach to Timed Automaton

Kozo OKANO
Graduate School of
Information Science and Technology,
Osaka Univsesity
Suita, Osaka Japan
Email:okano@ist.osaka-u.ac.jp

Behzad BORDBAR
School of Computer Science,
Univsesity of Birmingham
Birmingham
UK
Email:b.bordbar@cs.bham.ac.uk

Takeshi NAGAOKA
Graduate School of
Information Science and Technology
Osaka Univsesity
Suita, Osaka Japan

***Abstract*—This paper presents an adaptation of the CEGAR loop approach based on the reduction of the number of clocks in timed automata. In the presented method, an abstraction of the timed automata in which some of the clocks are removed is used to search for a counter-example for a given temporal logic statement. If the counter-example produced by the abstracted timed automaton is not a counter-example of the original timed automaton, the abstracted model is refined by restoring some of the clocks so that the process can be repeated for the new abstracted model. Reducing the number of the clock may result in a substantial reduction in the amount of the computation required for the model checking as the number states is exponential in the number of clocks.**

I. Introduction

Scalability is seen as one of the main challenges of the Model Checking techniques in timed automata [3]. In the past two decades various techniques for dealing with the so-called State Explosion problems have been proposed. A common approach is to reduce the size of the underlying timed automata, resulting in smaller model. Methods of reducing the size of timed automata include merging a number of locations into a single location [16], eliminating some of the clocks [12] or abstracting predicate to create smaller predicates [8]. This paper aims to introduce a new method based on Counter Example-Guided Abstraction Refinement (CEGAR) [5]. CEGAR has been successfully applied [11], [4] to Markov Decision Process (MDP) [13] and hybrid automata [10], as well as timed automata.

The CEGAR approach, which is sometimes called "CEGAR loop," carried out iteratively starting from an abstracted timed automata model produced from the original timed automata model. Given a CTL statement [6] to be verified, firstly the abstracted model is analyzed for a counter-example. If a counter-example is found for the abstracted model, it is checked against the original model. The process of checking is linear in the size of the counter-example. If the counter-example is also a counter-example of the main model, we have identified a counter-example without directly analyzing the original model which can be a very large. However, if the counter-example of the abstracted model is not a counter-example of the large model, i.e. the counter-example is *spurious*, the abstracted model is refined to produce a new model in which the counter-example of the abstracted model is no longer a counter-example. In this way the spurious counter-example is removed increasing the chance finding a "true" counter-example in the newly abstracted model.

In [12], [8], the authors present a method of applying the CEGAR loop approach aiming at reducing the number of locations when the abstracted timed automata is produced. In this paper, we shall adopt the CEGAR loop approach by focusing on the reduction of the number of clock as opposed to the number of locations. Since the number of the states is exponential in the number of the clocks [3], any reduction in the number of clocks might substantially reduce the computational cost.

The paper is organized as follows. Section 2 presents a set of introductory material about the timed automata, temporal logic, UPPAAL [3] and CEGAR loop method. Section 3 describes the problem addressed in this paper. To illustrate the presented approach, we rely on a running example borrowed from [14]. The example is also used for the evaluation purposes in Section 4. After the example, an outline of the proposed method, which includes details of three algorithms, is explained. Application of the suggested method to the running example is illustrated in Section 4. Section 5 gives the proof of the correctness of our algorithms.

II. Preliminaries

This section presents a short introduction on Automata, UPPAAL as well as a brief review of CEGAR.

A. Timed Automata

Timed automata are extensions of the conventional automata with variable and constraints for expressing real-time dynamics. They are widely used in the modelling and analysis of real-timed systems [1].

Definition 1 (constraints):
1) C represents a finite set of clocks
2) constraints related to time are expressed as inequality of the following form
$E ::= x \sim a \mid x - y \sim b \mid E_1 \wedge E_2$,
where $x, y \in C$, $\sim \in \{\leq, \geq, <, >, =\}$, and $a, b \in \mathbb{Q}_{\geq 0}$,

978-0-7695-4569-1/11 $26.00 © 2011 IEEE
DOI 10.1109/ICNC.2011.42

IEEE computer society

in which $\mathbb{Q}_{\geq 0}$ is the set of all non-negative Rational numbers.

Let $c(C)$ denote a set of constranits over C as described in Definiton 1.

The above time constraints are used to mark edges and nodes of the timed automata for describing the *guards* and *invariants*.

Definition 2 (timed automaton): A timed automaton $\mathscr{A}$ is a 7-tuple $(A, L, l_0, C, T, g, I, r)$, where
A: a finite set of actions;
L: a finite set of locations;
C: a finite set of clocks;
$l_0 \in L$: an initial location; and
$T \subseteq L \times c(C) \times A \times 2^C \times L$ is a set of transitions. The second and fourth items are called a guard and clock resets, respectively.
$I : L \to c(C)$ is a mapping from location to clock constraints, called a location invariant.
A transition $t = (l_1, g, a, r, l_2) \in T$ is denoted by $l_1 \overset{a,g,r}{\to} l_2$. A map $\nu : C \to \mathbb{Q}_{\geq 0}$ is called a clock assignment (or clock valuation). As a result, we have $\nu \in \mathbb{Q}_{\geq 0}^C$. We define $(\nu + d)(x) = \nu(x) + d$ for $d \in \mathbb{Q}_{\geq 0}$. For each reset r, where $r \in 2^C$ we shall write $r(\nu) = \nu[x \mapsto 0], x \in r$.

Dynamic of a timed automaton can be expressed via a set of states and their evaluations. Changes of one state to a new state can be as a result of firing of an action or elapse of time.

Definition 3 (state of timed automaton): For a given timed automaton $\mathscr{A} = (A, L, l_0, C, T, g, I, r)$, let $S \subseteq L \times V$ be a set of whole states of $\mathscr{A}$. The initial state of $\mathscr{A}$ can be given as $(l_0, 0^C) \in S$, where 0^C stands for 0 valuation for each clock. For a transition $l_1 \overset{a,g,r}{\to} l_2$, the following two transitions are semantically defined. The first one is called an action transition, while the latter one is called a delay transition.

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

Here, $g(\nu)$ and $I(l)(\nu)$ stand for the evaluation of clock constrains g and $I(l)$ under the evaluation of clock value ν.

Semantics of a timed automaton can be interpreted as a Labelled Transition System.

Definition 4 (semantic of a timed automaton): For a timed automaton $\mathscr{A} = (A, L, l_0, C, T, g, I, r)$, an infinite transition system is defined according to the semantics of $\mathscr{A}$, where the model begins with the initial state. By $\mathscr{T}(\mathscr{A}) = (S, s_0, \Rightarrow)$, the semantic model of $\mathscr{A}$ is denoted.

In this paper, a state on a location l means an arbitrary semantic state (l, ν) such that ν satisfies l's invariant.

Definition 5 (run of a timed automaton): For a timed automaton $\mathscr{A}$, a run σ is a finite or infinite sequence of transitions of $\mathscr{T}(\mathscr{A})$.

$\sigma = (l_0, \nu_0) \overset{\alpha_1}{\Rightarrow} (l_1, \nu_1) \overset{\alpha_2}{\Rightarrow} (l_2, \nu_2) \overset{\alpha_3}{\Rightarrow}, \ldots$, where $\alpha \in A \cup \mathbb{Q}_{\geq 0}$

Hereafter, we assume that $\mathscr{A}$ is not Zeno and is not timelock. For further detail about time automata, we refer the reader to [3], [17].

B. Temporal Logic

In timed automata the constraints which are verified are often expressed in temporal logic. Subsets of TCTL [1] is used for expressing the constraints to be verified.

The syntax definition of the logic is defined in Definition 6.

Definition 6: $\phi ::= \exists\Box_{\sim p}\psi \mid \forall\Box_{\sim p}\psi \mid \exists\Diamond_{\sim p}\psi \mid \forall\Diamond_{\sim p}\psi \mid \exists\psi_1\mathcal{U}_{\sim p}\psi_2 \mid \forall\psi_1\mathcal{U}_{\sim p}\psi_2$.

$\psi ::= \neg\psi \mid \psi_1 \vee \psi_2 \mid (l, c)$, where c and p are a clock constraint, and a positive rational number, respectively.

For a given constraint c a region is the set of all states whose clock value satisfies the constraint c. This is formally defined as follows.

Definition 7: (q, c) is a pair of location and a constraint over the clocks, We also use the notation to represent a set of pairs of a location clock regions as follows. (q, c) denotes a set $\{(q, \nu) \mid \forall\nu \models c\}$.

Definition 8 gives the semantics of TCTL.

Definition 8: For a pair of (q, ν) and a TCTL expression:

$(q, \nu), c$ iff $(q, \nu) \models c$, where c is a constraint over C.
$(q, \nu), \neg\psi$ iff $(q, \nu) \not\models \psi$.
$(q, \nu), \psi_1 \vee \psi_2$ iff $(q, \nu) \models \psi_1$ or $(q, \nu) \models \psi_2$ holds.
$(q, \nu), \exists\Box_{\sim p}\psi$ iff there is a run from (q, ν) such that for all states (q', ν') that is t time units from (q, ν) in the run with $t \sim p$, $(q', \nu') \models \psi$.
$\exists\psi_1\mathcal{U}_{\sim p}\psi_2$ iff there is a run from (q, ν) such that

- there is a state $(q', \nu'$ that is t time units from (q, ν) in the run with $t \sim p$ and $(q', \nu') \models \psi_2$; and
- for all states (q'', ν'') before (q', ν') in the run, $(q'', \nu'') \models \psi_1$).

Symbols $\mathtt{tt}$ and $\mathtt{ff}$ stand for true and false, respectively.

For given a timed automata $\mathscr{A}$ and a TCTL expression ϕ, if $(l_0, 0)$, ϕ holds, then we say $\mathscr{A}$ satisfies ϕ, where $(l_0, 0)$ is an initial state of $\mathscr{A}$. Also we denote $\mathscr{A} \models \phi$, if $\mathscr{A}$ satisfies ϕ.

The following expressions are also defined as abbreviations.

$\forall\Box_{\sim p}\psi$ iff $\neg\exists\Diamond_{\sim p}\neg\psi$
$\exists\Diamond_{\sim p}\psi$ iff $\exists\ \mathtt{tt}\ \mathcal{U}_{\sim p}\psi$
$\forall\Diamond_{\sim p}\psi$ iff $\models \neg\exists\Box_{\sim p}\neg\psi$
$\forall\psi_1\mathcal{U}_{\sim p}\psi_2$ iff $\neg\exists\neg\psi_1\mathcal{U}_{\sim p}(\neg\psi_1 \wedge \neg\psi_2) \vee \neg\exists\Box_{\sim p}\neg\psi_2$
$(q, \nu), (l, c)$ iff $(q, \nu) \models c$ and $l = q$, where c is a constraint over C.

We focus on expressions of the form $\phi ::= \forall\Box_{\sim p}\psi$ in this paper. Such statements are particularly important as they allow checking if the system is satisfying a property within a period of time. For example we can check if a property holds true within the first k unit of the start of the system. Such expressions can be reduced to reachability problems, which are verifiable via our model checkers. Extension of our method to deal with other types of temporal logic expressions is a topic for future research.

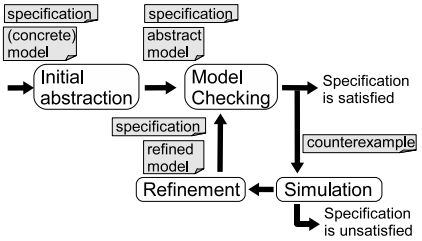


Fig. 1. General CEGAR Algorithm

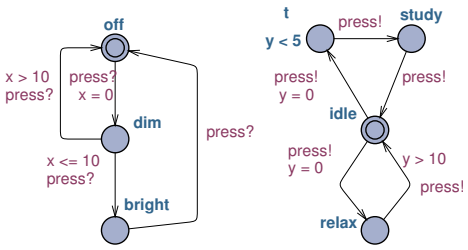


Fig. 2. light-user model

C. UPPAAL

UPPAAL is a famous model checker for extended timed automata by Yi-Wang et al. [3], [2]. It also supports model checking for the conventional timed automata. UPPAAL allows verification of expressions described in an extended version of CTL. In addition, it supports local and global integers and primitive operations on integers, such as addition, subtract and multiplication with constants. Such expressions are also allowed on the guards of transitions. The model of the system can be created from multiple timed automata which are synchronised together via a CCS-like synchronisation mechanisms [15].

D. General CEGAR Algorithm

Model abstraction sometimes over-approximates an original model, which may produce spurious counter-examples which are not actually counter-examples in the original model. Clarke et al. [5] present an algorithm called CEGAR (Counter-Example-Guided Abstraction Refinement) (Fig.1).

In the algorithm, at the first step (called Initial Abstraction), an abstracted model which over-approximates the original model is produced. Next, we perform model checking on the abstracted 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 abstracted 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 counter-example detected is a spurious counter-example or not in the next step (called Simulation). In the Simulation step, if we find the counter-example is valid, we stop the loop as a counter-example for the original model is found. Otherwise, we have to refine the abstracted model to eliminate the spurious counter-example, and repeat these steps until valid counter-example is obtained or model checker outputs the model satisfies the property.

Example 1: Fig. 2 shows a model of light user behaviour in a network of UPPAAL Timed Automata, which uses some extended notions of UPPAAL. The double discs and ordinal discs show the initial locations and normal locations, respectively.

The automaton shows the concurrent behaviour of a mug light and a human. The light has three operation modes, namely *off*, *dim*, and *bright*.

A single click of a switch will change the mode of the light to dim, while if the switch is pressed twice (double click) the mode changes to bright. When the light is either dim or bright, a single click will switch off the light. The difference between the timing of two consecutive clicks is controlled by a clock x and guards on it.

This network of timed automata, however, can be translated into single normal timed automaton.

E. Related Work

Several approaches [9], [8], [16], [7] apply the abstraction mechanism used in the CEGAR loop method to timed automata. The abstraction techniques fall into two main categories: timed abstraction and action abstraction. Paper [16] uses a normal automaton as an abstracted automaton, while this paper uses a timed aumtomaton. Papers [9], [8] present a CEGAR approach by abstracting the variables and clocks. In particular, He et al [9] introduce a compositional framework for dealing with network of timed automata. Their method is based on invariant shift paradigm, which is different from us; we use direct projections. Another point of difference is that our algorithm guides the choice of the next clock to be added. The experimental comparison of our work with [9] is a topic for future research. Our focus on time abstraction steams in high gain of the approach in terms of the complexity. Paper [7] proposes a tool, opaal, which uses CEGAR for a subset of the UPPAAL timed automata language extended with lattice features.

III. Implementing CEGAR loop by reducing clock numbers

The key aim of CEGAR loop approach is to obtain smaller timed automata as an abstraction of the model of the system which can be used for the analysis instead of the large model. The process of abstraction in the original CEGAR approach is focused on the reduction of the number of the locations. In contrast, our approach aims at the reduction of the clock numbers used in the abstraction. We argue that reduction of the clocks can result in smaller abstraction as the number of states in the timed automaton is $O(\mid L \mid \cdot \mid C \mid ! \cdot 2^{\mid C \mid} \cdot \prod_{x \in C}(2c_x + 2))$, where c_x is the maximum constants appeared in clock constraints on clock x used for the timed automaton[2], It can be seen that the number of state is exponential in the number

of clocks, whereas the number of states is polynomial in the number of locations, Form this point of view, reduction of the clocks provides a substantial advantage.

The CEGAR loop approach is iterative. We shall follow an approach similar to the one suggested in [5] and introduce the following four steps. In what follows $\mathscr{A}$ is a timed automaton, ϕ is a TCTL formula. Let assume $\hat{\mathscr{A}}$ is an abstraction of $\mathscr{A}$.

A. Abstraction Methods

The underlying idea of our approach is to eliminate some of the clocks and the constraints that they appear in. Then, during the refinement step, some of the removed clocks are restored.

Step1 Initial Abstraction: All clocks and their constraints (guards, invariants) and resets are removed.

Step2 Model Checking: Given any abstracted timed automaton, the statement ϕ is analysed. There are two outcomes to the model checking: existence of the counter-example or no counter-example. For the later case, we have no counter-example for property ϕ on the over-approximated model. This means that there are no counter-examples for the original model too. As a result there is nothing more to do and the CEGAR loop can stop. Otherwise, we move to step 3.

Step3 Simulation: The created counter-example for the abstracted model must be checked against the automaton $\mathscr{A}$. This is carried out by an attempt to simulate the counter-example on $\mathscr{A}$. If the run cannot be successfully executed on $\mathscr{A}$, a counter-example for the validation of ϕ on $\mathscr{A}$ is a spurious counter-example, thus, the abstracted model must be refined. However, if the counter-example can be simulated on $\mathscr{A}$ we have obtained a counter-example by analysing a smaller model.

Step4 Refinement: In this step one of the clocks is restored, i.e. the guards and invariants related are included in the abstracted automaton to create a new abstracted automaton.

In the rest of this section, the above four steps are illustrated in details.

B. Abstraction Function

The abstraction function in the original CEGAR [5] algorithm modifies the structure of the involving timed automata by removing locations and diverting transitions from one location to another. This results in a modification of the underlying graphical representation of the automata. In contrast, the proposed method here makes no alternations to the location or transitions except eliminating some of the constraints related to the guards, invariants and reset clocks.

To be precise, suppose that $\mathscr{A}$ is a timed automaton and AC is a subset of clocks of $\mathscr{A}$ which will appear in $\hat{\mathscr{A}}$, i.e. all other clocks will be eliminated. Definition 9 precisely define $\hat{\mathscr{A}}$, an abstraction of $\mathscr{A}$ that included the clocks of AC.

Definition 9: [Abstraction] Given a timed automaton $\mathscr{A} = (A, L, l_0, C, T, g, I, r)$ and a set $AC \subseteq C$ of the clocks, $\hat{A} = A, \hat{L} = L, \hat{l_0} = l_0$ and $\hat{T} = T$ for a timed automaton $\hat{\mathscr{A}} = (\hat{A}, \hat{L}, \hat{l_0}, \hat{C}, \hat{T}, \hat{g}, \hat{I}, \hat{r})$ which is an abstraction of $\mathscr{A}$.

In addition,

- $\hat{C} = AC$, i.e. clocks in the abstraction are only the clocks in $\mathscr{A}$,
- $\hat{g} = \pi_{AC}(g)$ for each guard g,
- $\hat{I} = \pi_{AC}(I)$, and
- $\hat{r} = r \cap AC$ for each reset set r,

where π_{AC} is defined later.

To explain informally the project function π_{AC} removes all constraints related to variables which are not in AC, i.e. the clocks that should not be included. Next we shall give the definition of π_{AC}.

Definition 10: [projection]

1) $\pi_{AC}(\{x_i \sim \alpha\}) = \emptyset$, if $x_i \notin AC$
2) $\pi_{AC}(\{x_i - x_j \sim \alpha\}) = \emptyset$, if $x_i, x_j \notin AC$
3) $\pi_{AC}(\{x_i \sim \alpha\}) = \{x_i \sim \alpha\}$, if $x_i, x_j \in AC$
4) $\pi_{AC}(\{x_i - x_j < \alpha\}) = \mathtt{tt}$, if $x_i \in AC$ and $x_j \notin AC$
5) $\pi_{AC}(\{x_i - x_j < \alpha\}) = \{x_j \geq 0\}$, if $x_i \notin AC$ and $x_j \in AC$
6) $\pi_{AC}(E_1 \cup E_2) = \pi_{AC}(E_1) \cup \pi_{AC}(E_2)$
7) $\pi_{AC}(\emptyset) = \emptyset$
8) $\pi_{\emptyset}(g) = (\mathbb{Q}_{\geq 0})^n$
9) $\pi_{\emptyset}(I) = (\mathbb{Q}_{\geq 0})^n$
10) $\pi_{\emptyset}(r) = \emptyset$

To explain Definition 10, all constraints for clocks not in AC are removed (see *1*). In *3* and *4*, a constant of the form $x_i - x_j < \alpha$ where only one of the clocks in $\mathscr{A}$ is replaced with a larger region involving the clock which is in $\mathscr{A}$, because the regions are getting larger, none of the trajectories are eliminated. This means that in *4* the constraint is replaced by a trivial constraint of the form $-x_j < \alpha$ which is equivalent with $x_j > -\alpha$ replaced by $x_j > 0$. As clocks have a positive value. In *3* and *4* π_{AC} for $\geq, >, \leq$ can be defined similarly. In 7-10, we deal with the case that AC is an empty set. In this case, the projections result in no restrictions on the firing of the transitions.

A crucial property of an abstraction is compatibility.

Definition 11 (compatibility): For a given TCTL property ϕ and a timed automaton $\mathscr{A}$, we shall refer $\hat{}$ as the Concrete Abstraction Map. If the following holds, we say that the abstraction $\hat{}$ is compatible to ϕ.

$\hat{\mathscr{A}} \models \phi$ implies $\mathscr{A} \models \phi$.

Theorem 1 (compatibility): For a given ϕ and a timed automaton $\mathscr{A}$, the concrete abstraction mapping $\hat{}$ which will be given below is compatible to ϕ.

Proof: see Section 5.

C. Step1: Initial Abstraction

To start the CEGAR loop we shall use an Initial abstraction with an empty clock set, i.e. $AC = \emptyset$. This means the projection function π_{AC} will remove all the involved guards and reset sets. This results in an abstracted automaton with no clock related restrictions. In other words for all guards g and invariant I, $\hat{g} = \hat{I} = (\mathbb{Q}_{\geq 0})^n$. In addition for each reset set r, $\hat{r} = \emptyset$.

D. Step2: Model Checking

The model checking is carried out via UPPAAL model checker. If no counter-example when checking ϕ on $\hat{\mathscr{A}}$ is found, by Theorem 1, we have known that ϕ would be valid for $\mathscr{A}$. However, in case of finding a counter-example, the counter-example must be checked against $\mathscr{A}$.

E. Step3: Simulation

In Step3 a tool from UPPAAL, called Tracer can produce a counter-example for an abstracted automaton $\mathscr{A}$. Such a counter-example is in the form $\sigma = (l_0, c\hat{r}_0) \overset{a_1}{\Rightarrow} (l_1, c\hat{r}_1) \overset{a_2}{\Rightarrow} (l_2, c\hat{r}_2)\cdots$, where each $a_i = (t, a)$ when t is the amount of time spent in the location l_{i-1} and $a \in A$ is an action causing a change of location from l_{i-1} to l_i. Each $c\hat{r}_i$ is a set of constraints representing the set of possible clock values when the location is l_i. The time elapsing is not shown in the counter-example produced by the tool. More precisely, form of sequence $(\vec{l_0}, c\hat{r}_0) \overset{a_1}{\Rightarrow} (\vec{l_1}, c\hat{r}_1) \overset{a_2}{\Rightarrow} (\vec{l_2}, c\hat{r}_2)\cdots$, where a_i is just the information of an edge, and the time elapsing is not shown [1]. The sequence σ is an execution sequence of $\hat{\mathscr{A}}$. The aim of this step is to check if σ is also an execution sequence of $\mathscr{A}$. We make use of forwards simulation for checking it. Forwards simulation is the same as the simulation of the original CEGAR [5]. Step-by-step, starting at the state $(l_0, \vec{0})$ of $\mathscr{A}$ we check if actions $a_1, a_2, \ldots$ can execute. If all actions can execute, then we have found a counter-example for checking ϕ on $\mathscr{A}$. Otherwise the counter-example is spurious.

Definition 12: [a bad location] Suppose that $\sigma = (l_0, c\hat{r}_0) \overset{a_1}{\Rightarrow} (l_1, c\hat{r}_1) \overset{a_2}{\Rightarrow} (l_2, c\hat{r}_2) \overset{a_3}{\Rightarrow} \cdots \overset{a_m}{\Rightarrow} (l_m, c\hat{r}_m)$ is a spurious counter-example. Suppose that $l_k (1 \leq k \leq m)$ is the first location in $\mathscr{A}$ such that $a_1, a_2, \ldots, a_{k-1}$ cannot execute. We refer l_k as a bad location.

Fig. 3 shows an example of Simulation and counter-example. In the example, location $(d,$ Norm$)$ is the bad location.

F. Step4: Refinement

Let us denote by $\mathscr{A}_i$ a timed automaton produced in the i-th iteration of CEGAR loop. Suppose that $\mathscr{A}_i$ has produced a spurious counter-example. The aim of Step4 is to use the counter-example and produce a new timed automaton $\mathscr{A}_{i+1}$ which increases the number of the clocks in $\mathscr{A}_i$ so that the spurious counter-example is removed. In other words to create $\mathscr{A}_{i+1}$, the set of clocks used in $\mathscr{A}_i$ are increased and the abstraction process in Definition 9 is applied to $\mathscr{A}_{i+1}$.

Suppose that $\sigma = (l_0, c\hat{r}_0) \overset{a_1}{\Rightarrow} \cdots \overset{a_m}{\Rightarrow} (l_m, c\hat{r}_m)$ is a spurious counter-example. This means an attempt to run the sequence of execution has resulted in identifying a bad location $l_k (0 \leq k \leq m)$ as defined in Definition 12. Our policy is choosing the clock that has made the counter-example spurious. In other words, assume that C_i and C are the set of clocks in $\mathscr{A}_i$ and $\mathscr{A}$, respectively. We will identify a clock in $C \setminus C_i$ that has made l_k a bad location. As l_k is a bad location none of the transitions from l_k to l_{k+1} can fire. This means that none of the guards of the transitions from l_k to l_{k+1} can fire. This could be because of a variable from $C \setminus C_i$ that the guards of such transitions are invalid. In such a case we choose one of the variables from the invariants of l_k, guards or set of resets of transitions from l_k to l_{k+1}. However, there is a chance that all variables used in the transition from l_k to l_{k+1} and the invariants of l_k are already in C_i. In that case we look for the variables of transitions from l_{k-1} to l_k, the invariant of l_{k-1} or the set of resets of transitions from l_{k-1} to l_k. This process of searching can be repeated until a variable from C/C_i of transitions from l_i to $l_{i+1} (0 \leq i \leq k)$ is found. Lemma 1 ensures that such a variable exists.

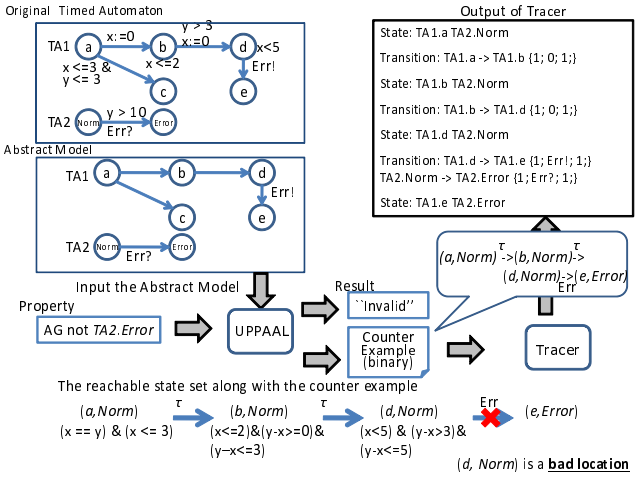


Fig. 3. Example: Simulation

Lemma 1: Suppose that $\mathscr{A}_i$ is an abstraction of the automaton $\mathscr{A}$. Suppose that C_i is the set of clocks in $\mathscr{A}_i$ Let us assume that a sequence $\sigma = (\hat{l_0}, c\hat{r}_0) \overset{a_1}{\Rightarrow} \cdots \overset{a_m}{\Rightarrow} (\hat{l_m}, c\hat{r}_m)$ is an execution trace of $\mathscr{A}_i$. Assume that all guards and reset of edges from l_j to l_{j+1} are using only the clocks from C_i. Then σ is not spurious.

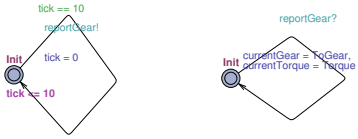
Proof: The proof is trivial as the part of the underlying timed automata $\mathscr{A}_i$ which has generated the counter-example is identical with the corresponding part of the automata. ■

IV. Experimental Results

In this section we shall describe some experimental results to evaluate the approach. To do so, we shall make use of a modified version of the widely used example Gearbox Controller example [14] by adding an engine monitor module, which observes the torques and states of the gears.

We evaluate memory consumption and execution time for the both of model checking with abstraction and no abstraction in order to evaluate effectiveness of our proposed method. We used verifyTA tool of UPPAAL. VerifyTA is the stand-

[1]In this section, we shall assume that the (Network of) timed automata consists of a single automaton as opposed to multiple automata which are synchronised via half actions, see [2]. This assumption is to simplify our notation. However, this assumption will not affect the validity of our argument. For the general case, we need to only replace the location with a vector of locations of the Automata involved in the Network of Timed Automata.



View Display

Fig. 4. Modules to Modify Gear Controller

TABLE I
RESULT OF 1,000 UNITS OF TIME

query	type	time(sec)	memory(KB)	loops	restored clocks
0	no abst	0.8	41000	-	-
0	abst	1.54	48868	3	2
1	abst	1.53	48868	3	2
2	abst	1.45	47824	3	2
3	abst	1.47	47824	3	2
4	abst	1.49	47172	5	4
5	abst	1.06	49376	1	0

alone verifier of UPPAAL [2]. The original Gearbox Controller consits of six timed automata. We add modules View and Display. View periodically watches the status of the engine and sends to Display. Display shows the current torques and state of gears when it receives the signal. We also add a variable which represents the states of gears (Fig. 4).

In the experiments, we obtain the results for the case of 1,000 units of time and 10,000 unit of time as the periodical parameter for the sending signals from View to Display, respectively.

We choose six properties which are able to apply our method from the properties in [14]. The results for the case of 1,000 units of time are shown in Table I.

In Table I., the rows with the second entry "no abst" show the result obtained from conventional model checking, i.e. without using our method, and the rows with the entry "abst" are the result of the application of our approach. Column query, time, memory and loop show the query (property) we used, the execution time, the maximum of memory consumption and the number of iteration of CEGAR loops, respectively. Column *restored clock* shows the number of restored clocks while the CEGAR loop. Though, time and memory consumption are larger than the original, not all of the clocks are restored, i,e, at least two clocks are yet abstracted. The result shows that the approach is useful for abstraction of more complicated examples.

We have performed the same experiment for the case of 10,000 units of times. It can be seen that using our method show considerable reduction in the time required (Table II). This shows a major advantage of our method. However, in terms of the memory consumption, we observe only a slight reduction in the memory consumption. We think, when using 1000 units of time, the range of clock variable ticks are larger, as a result, the system consumes more memory. However, further experiments must be carried out to establish this point.

TABLE II
RESULTS OF 10,000 UNITS OF TIME

query	type	time(sec)	memory(KB)	loops	restored clocks
0	no abst	4.2	52716	-	-
0	abst	1.57	48868	3	2
1	abst	1.92	48868	3	2
2	abst	1.45	47692	3	2
3	abst	1.46	47824	3	2
4	abst	1.49	42172	5	4
5	abst	1.06	49508	1	0

V. PROOF OF THEOREM 1

The aim of this section is to present the proof of Theorem 1. Suppose that $\mathscr{A}$ is a timed automaton with the abstraction $\hat{\mathscr{A}}$ as obtained via Definition 9. Consider temporal logic constraints of the form $\psi = \neg\psi \mid \psi \wedge \psi \mid (q, c)$, where c is conjunction of the clock constraints of the form $x \sim p$ or $x - y \sim p$ which x and y are clocks and c is a non-negative rational number. By (q, c), in which q is a location of the timed automaton, we mean that in the location q the clocks satisfy the constraint c. To prove Theorem 1, i.e. the compatibility of abstraction, we must show that for each ψ, $\hat{\mathscr{A}} \models \forall\Box_{\sim p}\psi$ implies $\mathscr{A} \models \forall\Box_{\sim p}\psi$. To do we shall start with a lemma.

Lemma 2: An expression $\neg(q, c)$ equals to $(q, \neg c)\vee(\neg q, \mathtt{tt})$, where $(\neg q, \mathtt{tt}) = \bigvee_{q' \in L,\ s.t.,\ q' \neq q}(q', \mathtt{tt})$.

Proof: By Definition7, it is trivial. ■

This can be interpreted as follow. $\neg(q, c)$ means negation of being at location q while c holds. Such negation can be seen as either we are not in q, no matter what condition $(\neg q, \mathtt{tt})$ or we are in q but c is not valid i.e. $(q, \neg c)$.

Lemma 3: Each ϕ of the form of $\forall\Box_{\sim p}\psi$ can be written as $\forall\Box_{\sim p}\bigwedge_i\bigvee_j\bigvee_k\neg(q_{ij}, c_{ijk})$, where c_{ijk} is $x \sim c$ or $x - y \sim c$.

Proof: The proof is by considering all possible cases for $\psi = \neg\psi \mid \psi \wedge \psi \mid (q, c)$.

case 1 $(\psi = (q, c))$

$(q, c) = \neg((q, \neg c) \vee (\neg q, \mathtt{tt}))$ by Lemma 2
$= \neg(q, \neg c) \wedge \neg(\neg q, \mathtt{tt})$
$= \neg(q, \neg c) \wedge \neg\bigvee_{q' \in L,\ s.t.,\ q' \neq q}(q', \mathtt{tt})$
$= \neg(q, \neg c) \wedge \bigwedge_{q' \in L,\ s.t.,\ q' \neq q}\neg(q', \mathtt{tt})$

case 2 $(\psi = \psi_1 \wedge \psi_2)$

We let ψ_1 and ψ_2 be $\bigwedge_i\bigvee_i\bigvee_k\neg(q_{ij}, c_{ijk})$ and $\bigwedge_h\bigvee_g\bigvee_f\neg(q_{hg}, c_{hgf})$, respectively.
$\psi_1\wedge\psi_2 = (\bigwedge_i\bigvee_j\bigvee_k\neg(q_{ij}, c_{ijk}))\wedge(\bigwedge_h\bigvee_g\bigvee_f\neg(q_{hg}, c_{hgf}))$.

case 3 $(\neg\psi)$

We let ψ be $\bigwedge_i\bigvee_j\bigvee_k\neg(q_{ij}, c_{ijk})$.
$\neg\psi = \neg\bigwedge_i\bigvee_j\bigvee_k\neg(q_{ij}, c_{ijk})$
$= \bigvee_i\bigwedge_j\bigwedge_k\neg\neg(q_{ij}, c_{ijk})$
$= \bigvee_i\bigwedge_j\bigwedge_k(q_{ij}, c_{ijk})$.

Any logical expression of the disjunctive normal form can be translated into an equivalent conjunctive normal form. The translated form might include a sub-formula (q, c) which is not the negation form. Such a sub-formula, however can be translated into the form of conjunction of negations according to the proof of the case 1. This completes the proof. ■

Definition 13 (Reachability Problem): Suppose that $\mathscr{A}$ is a timed automaton. We say $(q, c)$is a reachable state if there is a

run of $\mathscr{A}$, see Definition 5, starting from the initial state and terminating in a state satisfying (q, c).

Theorem 2: The problem on Theorem 1 can be reduced into a Reachability Problem. In other words, $\hat{\mathscr{A}} \models \forall\Box_{\sim p} \bigwedge_i \bigvee_j \bigvee_k \neg(q_{ij}, c_{ij})$ iff $\bigwedge_j \bigvee_k (q_{ij}, c_{ijk})$ is not reachable for each i within the period specified by $\sim p$.

Proof: $\hat{\mathscr{A}} \models \forall\Box_{\sim p} \bigwedge_i \bigvee_j \neg(q_i, c_{ij})$ iff
$\hat{\mathscr{A}} \models \bigwedge_i \forall\Box_{\sim p} \bigvee_j \neg(q_i, c_{ij})$ iff
$\hat{\mathscr{A}} \models \bigwedge_i \forall\Box_{\sim p} \neg \bigwedge_j (q_i, c_{ij})$ iff
$\bigwedge_j (q_i, c_{ij})$ is not reachable for every i within the period specified by $\sim p$. ■

Definition 14: Suppose that y is a clock variable. Let $C\backslash y$ denote removing of all constraints involving y from C.

For example in Definition 14, if C equals to $\{x < 2, x - y \geq 5\}$, then $C\backslash y$ will be $\{x < 2\}$.

Lemma 4: For any valuation if C is valid then $C\backslash y$ is also valid.

Proof: Let ν and c are an evaluation and a condition in C. From the assumption $\nu \models c$, if c involves a variable y then it is not in $C\backslash y$, otherwise $\nu \models c$. Thus, Lemma 4 holds. ■

Lemma 5: For a given set AC of clocks and any expression exp used in a guard or an invariant, $\pi_{AC}(\{exp\}) \subseteq \{exp\}$

Proof: Definition of π_{AC}, see Definition 10, consists of ten parts. For each parts it can be seen that π_{AC} maps its region to one of its subsets. ■

Lemma 6: Consider an action transition a of the form $(l, \nu) \stackrel{a}{\Rightarrow} (l', \nu')$ or a delay transition d of the form $(l, \nu) \stackrel{d}{\Rightarrow} (l, \nu')$ in the timed automata $\mathscr{A}$. If the state (l, ν) is a reachable state in the abstracted timed automata $\hat{\mathscr{A}}$, then the transitions $(l, \nu) \stackrel{a}{\Rightarrow} (l', \nu')$ and $(l, \nu) \stackrel{d}{\Rightarrow} (l, \nu')$ can also occur in $\hat{\mathscr{A}}$.

Proof: First, we consider an action transition. Let assume that $(l, \nu) \stackrel{a}{\Rightarrow} (l', \nu')$ of $\mathscr{A}$ holds. Then there exists a transition $l \stackrel{a,g,r}{\rightarrow} l'$ in $\mathscr{A}$. Also $g(\nu)$ and $I(l')(r(\nu))$ hold. Let assume the corresponding transition in $\hat{\mathscr{A}}$ be $l \stackrel{a,\hat{g},\hat{r}}{\rightarrow} l'$. By Lemma 4 and 5, $\hat{g}(\nu)$ and $\hat{I}(l')(\hat{r}(\nu))$ also hold.

For a delay transition, a similar proof can be given. Let assume that $(l, \nu) \stackrel{d}{\Rightarrow} (l, \nu')$ of $\mathscr{A}$ holds. Then there exists a location l in $\mathscr{A}$ and $\forall d' \leq d\ I(l)(\nu + d')$ holds. The corresponding location in $\hat{\mathscr{A}}$ is also l. By Lemma 4 and 5, $\forall d' \leq d\ \hat{I}(l)(\nu + d')$ also holds. ■

Now, we can see the proof of Theorem 1. The expression $\bigwedge_j (q_i, c_{ij})$ is not reachable for any i within any period specified by $\sim p$ in $\hat{\mathscr{A}}$ if it is not reachable in $\mathscr{A}$, by Lemma 6. By Theorem 2 with this fact, Theorem 1 is proved.

VI. Conclusion

This paper provided a new CEGAR loop method for a timed automaton to address the state explosion problem. The key idea is to use timed automata with smaller number of clocks to reduce the complexity of the analysis. Drawing on the conventional CEGAR approach, in our method some of the clocks and their constraints are removed to produce an abstraction of the model. If the abstracted model can result in producing a counter-example, which is also a counter example of the original timed automaton, then we have obtained a counter-example for a larger model by analysing a smaller model. Otherwise, the model is refined. At the refinement step, some of the removed clocks are restored. The restored clocks are determined on the basis of valuation errors. The paper also give a proof for the presented result. Experimental results show the proposed approach can be very effective.

Acknowledgment

We thank Prof. Teruo Higashino and Prof. Shinji Kusumoto for financial assistance. This research is partially supported by the research grant of Telecommunications Advancement Foundation and Grant-in-Aid for Scientific Research (C)(21500036).

References

[1] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In *Proc. of the 5th Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE, 1990.

[2] G. Behrmann, A. David, and K G. Larsen. A tutorial on uppaal. In *Proc. of the 4th Int. School on Formal Methods for the Design of Computer, Communication, and Software Systems*, volume 3185, pages 200–236, 2004.

[3] J. Bengtsson and W .Yi. Timed automata: Semantics, algorithms and tools. In *Lecture Notes on Concurrency and Petri Nets*, volume 3098, pages 87–124, 2004.

[4] E M. Clarke, A. Fehnker, Z. Han, J Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. Journal of Foundations of Computer Science*, 14(4):583–604, 2003.

[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] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[7] A.E. Dalsgaard, R.R. Hansen, K.Y. Joergensen, K.G. Larsen, M.Chr. Olesen, P. Olsen, and J. Srba. opaal: A lattice model checker. In *Proceedings of the 3rd NASA Formal Methods Symposium (NFM'11)*, volume 6617 of *LNCS*, pages 487–493. Springer-Verlag, 2011.

[8] H. Dierks, S. Kupferschmid, and K G. Larsen. Automatic abstraction refinement for timed automata. In *Proc. of the 5th Int. Conf. on Formal Modelling and Analysis of Timed Systems*, volume 4763, pages 114–129, 2007.

[9] F. He, H. Zhu, W. N. N. Hung, X. Song, and M. Gu. Compositional abstraction refinement for timed systems. In *Proc. 2010 Fourth International Symposium on Theoretical Aspects of Software Engineering*, pages 168–176, 2010.

[10] T.A. Henzinger. The theory of hybrid automata. In *Symp. on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.

[11] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic cegar. In *Computer Aided Verification, Lecture Notes in Computer Science*, volume 5123, pages 162–175. Springer, 2008.

[12] S. Kemper and A. Platzer. Sat-based abstraction refinement for real-time systems. In *Proc. of the Third Int. Workshop on Formal Aspects of Component Software*, volume 182, pages 107–122, 2006.

[13] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for markov decision processes. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE CS Press, 2006.

[14] M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear controller: An industrial case study using uppaal. In *Lecture Notes in Computer Science*, volume 1384, pages 289–297, 1998.

[15] R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989.

[16] 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, 2010.

[17] F. Wang, K. Schmidt, G. D. Huang, F. Yu, and B. Y. Wang. Formal verification of timed systems: A survey and perspective. In *Proc. of the IEEE*, volume 92(8), pages 1283–1307, 2004.