社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2007-74(2008-3)

反例に基づく抽象化改良ループによる時間オートマトンの抽象化手法

長岡 武志† 岡野 浩三† 楠本 真二†

† 大阪大学大学院情報科学研究科
〒 560–8531 大阪府豊中市待兼山 1–3

あらまし 時間オートマトンを対象としたモデル抽象化手法を著者らは提案している．提案手法では，反例を元に抽象モデルを改良し，適切な抽象モデルを自動的に生成する CEGAR (CounterExample-Guided Abstraction Refinement) ループに基づく．抽象モデルの改良の際，元になる時間オートマトンの変形で行うなどの特徴を持つ．本稿では，提案手法の具体的なアルゴリズムを形式的に記述し，アルゴリズムの正当性の証明を与える．

キーワード モデル検査，時間オートマトン，モデル抽象化，CEGAR

Abstraction of Timed Automata Based on Counterexample-Guided Abstraction Refinement Loop

Takeshi NAGAOKA†, Kozo OKANO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University
Machikane-yama 1–3, Toyonaka City, Osaka, 560–8531 Japan

Abstract We have proposed a method of model abstraction for timed automata. The proposed method is based on CEGAR (CounterExample-Guided Abstraction Refinement) loop which automatically refines an abstract model using counter examples. Our algorithm has some features such as refinements are performed indirectly through transformation of the original timed automaton. This paper gives formal descriptions of the algorithm and the correctness proof of the algorithm.

Key words Model Checking, Timed Automaton, Model Abstraction, CEGAR

1. Introduction

This paper gives correctness proof of our algorithm proposed in [8]. The algorithm is CEGAR [1] based algorithm of abstract model refinement used for model checking on timed automata.

A general CEGAR algorithm 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 counter example (CE) is found, it checks the counter example on the concrete model. If the CE is spurious, it refines the abstract model. The last step is repeated until the valid output is obtained.

In general, most CEGAR based algorithms [1], [2] obtain refined abstract models from the previous abstract models by modifying some transformations. In our algorithm, however, the refined model is obtained indirectly; we transform the original timed automaton preserving the equivalence and from it we generate an abstract model by eliminating clock attributes.

This paper proves that the transformation preserves bi-simulation equivalence and also the refined abstract model is the spurious CE free.

The rest of the paper is organized as follows. In Sec. 2., some definitions are described. Sec. 3. gives our CEGAR algorithm. Sec. 4. proves the correctness of the algorithm. Sec. 5. concludes the paper.

2. Preliminaries

In this Section, we give definitions of a timed automaton, a region automaton which specifies whole states of a timed automaton with finite clock regions, and others.

Let $c(C)$ be a set of whole differential inequalities of 2 clocks over a finite clock set C. A subset of $c(C)$ is called clock constraints.

Definition 2.1 (Timed Automaton). *A timed automaton $\mathscr{A}$ is a 6-tuple (L, l_0, T, I, C, A), where*
C : a finite set of clocks; A : a finite set of actions; L : a finite set of locations; $l_0 \in L$:an initial location; $T \subset L \times A \times 2^{c(C)} \times \mathscr{R} \times L$; where, $2^{c(C)}$ is a set of clock constraints, called guards; $\mathscr{R} = 2^C$

: a set of clocks to reset; and $I \subset (L \to 2^{c(C)})$ *: a mapping from locations to clock constraints, called location invariants.*

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 $\nu : C \to \mathbb{R}_{\geqq 0}$ is called a clock assignment. We can extend the domain of ν into a set of C as follows: $\nu \in \mathbb{R}^C_{\geqq 0}$. We define $(\nu + d)(x) = \nu(x) + d$ for $d \in \mathbb{R}_{\geqq 0}$. $r(\nu) = \nu[x \mapsto 0], x \in r$ is also defined for $r \in 2^C$. By N, a set of whole ν is denoted.

Definition 2.2 (Semantics of Timed Automaton). *For a given timed automaton* $\mathscr{A} = (L, l_0, T, I, C, A)$*, let a set of whole states of* $\mathscr{A}$ *be* $S = L \times N$.
The initial state of $\mathscr{A}$ *shall be given as* $(l_0, 0^C) \in S$.
For a transition $l_1 \xrightarrow{a,g,r} l_2$ $(\in T)$*, the following two transitions are semantically defined. The first one is called an action transition, while the latter one is called a delay transition.*

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

For a given timed automaton $\mathscr{A}$, we can introduce a corresponding clock region $CR(\mathscr{A})$ [4], [5]. In general, a clock region divides a $|C|$-dimensional Euclidean space into finite points, segments, and faces. By $[u]$, an element (a region) in $CR(\mathscr{A})$ is denoted. For $[u] \in CR(\mathscr{A})$, $g([u])$ and $I([u])$ represent that any point in $[u]$ satisfies a guard g and invariant I, respectively. Also by $r([u])$, applying clock resetting r onto $[u]$ is denoted, where $r([u]) = [u][x \mapsto 0]$, and $x \in r$.

Definition 2.3 (Region Automaton). *A region automaton* $\mathscr{A}_r = (L_r, l_{r\,0}, T_r, A)$ *of a given timed automaton* $\mathscr{A} = (L, l_0, T, I, C, A)$ *is defined as follows.*
$L_r \subset L \times CR(\mathscr{A})$, $l_{r\,0} = (l_0, [0^C])$*, where* $[0^C]$ *satisfies* $I(l_0)$*,* $T_r \subset L_r \times A \times L_r$*,* T_r *consists of*

- $(l, [u]) \overset{a}{\Rightarrow} (l', [v])$ *if* $(l, u) \overset{d}{\Rightarrow} (l, u')$ *for* $d \in \mathbb{R}_{\geqq 0} \wedge (l, u') \overset{a}{\Rightarrow} (l', v)$ *for* $a \in A$.

There is a bi-simulation equivalence between a timed automaton $\mathscr{A}$ and its region automaton $\mathscr{A}_r$ [3].

In [6], [7], a data structure DBM (Difference Bound Matrix) is introduced to represent a convex space in $|C|$ -dimensional Euclidean space, where C is a set of clock variables. It is also represented as a set of some elements in the clock region $CR(\mathscr{A})$. A state set of states of a region automaton $\mathscr{A}_r = (L_r, l_{r\,0}, T_r, A)$, can be represented in $(l, D) = \{(l, [u]) \mid [u] \in D\}$ using the corresponding DBM D. Paper [6] gives operation functions on DBM, such as *up*, *and* and other functions, which represent elapsing time, intersection of time spaces and so on, respectively. There is a minimum set of differential inequalities which can represents DBM D [6]. Such a set is denoted by $c(D)$. $c(D)$ can be obtained by reduction operations on DBM. A set of every region which satisfies an invariant $I(l)$ of location l is denoted by (l, D_{Inv}).

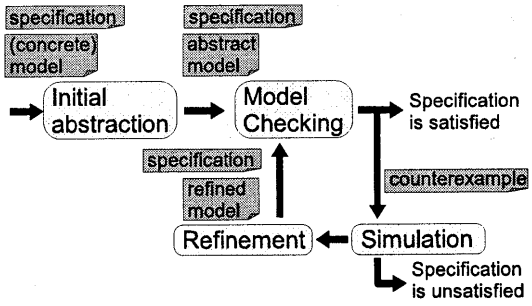

Figure 1 General CEGAR Algorithm

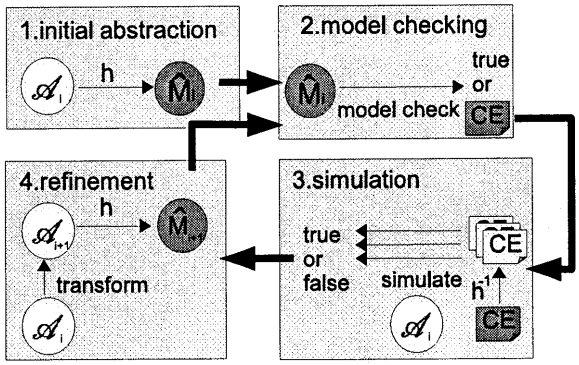

Figure 2 Our Proposed Algorithm

3. Algorithm

3.1 General CEGAR Algorithm

Model abstraction sometimes over-approximates an original model, which causes spurious counter examples which are not actually counter examples in the original model. Paper [1] gives an algorithm called CEGAR (Counterexample-Guided Abstraction Refinement). In the algorithm, at the first step (called Initial Abstraction), it approximates the original model, and the next step, if a spurious counter example is found in the abstract model, it refines the abstract model as it does not admit the spurious counter example. The next step is repeated until valid output is obtained. Figure 1 shows the general flow of the CEGAR algorithm.

3.2 Our Proposed Algorithm

Our proposed algorithm generates an abstract model $\hat{M}$ from a given timed automaton $\mathscr{A}$ by applying an abstraction function h, and performs model checking on $\hat{M}$. If a counter example $\hat{T}$ (represented as a path on the abstract model) is found while model checking, it concretizes $\hat{T}$ by applying inverse function h^{-1}.The concretized one is a set of paths. We denote it by T (which is a set of paths on $\mathscr{A}$). At Simulation Step, it checks whether each path in T is feasible on $\mathscr{A}$ or not. If every path in T is infeasible, the next step shall refine the model so that the counter example $\hat{T}$ becomes infeasible. Our algorithm does not directly refine $\hat{M}$ but it refines $\mathscr{A}$ and then obtains a new abstract mode by applying h to the refined timed automaton. Figure 2 shows flow of our CEGAR algorithm.

The proposed algorithm checks a property AG $\bigvee_{e \in E} \neg e$, where E ($\subset L$) of a timed automaton $\mathscr{A}$ is a set of error locations of the target system. The property means there is no path to locations in E from the initial state. Please note that any counter example of such a property can be represented in a finite length of sequence without loops. Therefore, hereafter, we assume that counter examples are finite sequences without loops.

3.2.1 Abstract Model

Definition 3.1 defines the abstraction function h on L_r of a region automaton $\mathscr{A}_r$.

Definition 3.1 (Abstraction Function h). *For a region automaton $\mathscr{A}_r = (L_r, l_{r\,0}, T_r, A)$ of a given timed automaton $\mathscr{A}$, an abstraction function $h : L_r \to \hat{S}$ is defined as follows:*

- $\forall l_{r\,i}, l_{r\,j} \in L_r.\ h(l_{r\,i}) = h(l_{r\,j}) \iff Loc(l_{r\,i}) = Loc(l_{r\,j})$,

where $Loc : L_r \to L$ is a function which retrieves a location attribute from a state of $\mathscr{A}_r$. The inverse function $h^{-1} : \hat{S} \to 2^{L_r}$ of h is also defined as in a usual manner.

The abstraction function h defined in Definition 3.1 maps any state of L_r which belongs to the same location into the same abstract state. Otherwise they are mapped into the different states. This means that there is a one-to-one correspondence between the location set of $\mathscr{A}$ and the abstract state set $\hat{S}$. Therefore, the abstraction function h can be extended its domain as in Definition 3.2.

Definition 3.2 (Extension of Abstraction Function h). *Abstraction function $h : L \to \hat{S}$ of a timed automaton $\mathscr{A} = (L, l_0, T, I, C, A)$ is defined as follows:*

- $\forall l_i, l_j \in L.\ h(l_i) = h(l_j) \iff l_i = l_j$.

Similarly, the inverse function $h^{-1} : \hat{S} \to L$ of h is also defined.

Definition 3.3 gives an abstract model $\hat{M}$ of a given timed automaton $\mathscr{A}$ using the abstraction function h defined in Definition 3.2.

Definition 3.3 (Abstract Model). *An abstract model $\hat{M} = (\hat{S}, \hat{s}_0, \hat{\rightarrow})$ of a given timed automaton $\mathscr{A} = (L, l_0, T, I, C, A)$ using the abstraction function h defined in Definition 3.2 is defined as follows:*

- $\hat{S} = \{h(l) \mid l \in L)\}$,
- $\hat{s_0} = h(l_0)$,
- $\hat{\rightarrow} = \{(\hat{l_1}, a, \hat{l_2}) \mid (l_1, a, g, r, l_2) \in T\}$.

Definition 3.4 (Counter Example). *A counter example on $\hat{M}$ is a sequence of states of $\hat{S}$. A counter example $\hat{T}$ of length n is represented in $\hat{T} = \langle \hat{s}_0, \cdots, \hat{s}_n \rangle$.*

A set T of a run sequences on $\mathscr{A}$ obtained by concertizing a counter example $\hat{T} = \langle \hat{s}_0, \cdots, \hat{s}_n \rangle$, is defined as follows:

$$T = \{(l_0 \xrightarrow{a_1, g_1, r_1} l_1 \xrightarrow{a_2, g_2, r_2} \cdots \xrightarrow{a_n, g_n, r_n} l_n) \mid (\, l_i = h^{-1}(s_i) \text{ for } 0 \leqq i \leqq n\,) \wedge (\, (l_{i-1}, a_i, g_i, r_i, l_i) \in T \text{ for } 1 \leqq i \leqq n\,)\}.$$

Abstraction

Inputs $\mathscr{A}, h$

```
{h = abstraction function}
Ŝ := ∅,  ⇢ := ∅ {M̂ = (Ŝ, ŝ0, ⇢)}
foreach l ∈ L do
  Ŝ := Ŝ ∪ {h(l)}
end for
ŝ0 := h(l0)
foreach (l1, a, g, r, l2) ∈ T do
  →:=→ ∪{(h(l1), h(l2))}
end for
return M̂
```

Figure 3 Abstraction

Simulation

Inputs $\mathscr{A}, (l_0 \xrightarrow{a_1, g_1, r_1} l_1 \xrightarrow{a_2, g_2, r_2} \cdots \xrightarrow{a_n, g_n, r_n} l_n (l_n = e))$

```
R0 := (l0, D0) {D0 = {0^C}}
D := up(D0) {Any elapsing time}
D := and(D, I(l0)) {Add Invariant of l0}
for i := 1 to n do
  Ri :=Reach(A, Ri−1, (li−1, ai, gi, ri, li))
  if Ri = ∅ then
    return false
  end if
end for
return true
```

Figure 4 Simulation

Reach

Inputs $\mathscr{A}, R = (l, D), (l_1, a, g, r, l_2)$

```
D := and(D, g) {add guards of transitions}
D := reset(D, r) {reset the clocks}
D := and(D, I(l2)) {add Invariant of l2}
D := up(D) {Any elapsing time}
D := and(D, I(l2)) {add Invariant of l2}
return (l2, D)
```

Figure 5 Reach

3.2.2 Initial Abstraction

Initial Abstraction generates an abstract model $\hat{M}$ from a timed automaton $\mathscr{A} = (L, l_0, T, I, C, A)$ using the abstraction function h. Figure 3 shows the algorithm of Initial Abstraction.

3.2.3 Simulation

For a set T of concretized counter example sequences obtained from $\hat{T}$ on $\hat{M}$, Simulation performs the algorithm in Fig. 4 on each sequence $t \in T$. Reachability from the first location of t to the last location of t is checked in Simulation using a procedure Reach in Fig. 5. Reach uses some operation functions of DBM. When the algorithm in Fig. 4 returns false, the counter example $\hat{T}$ is judged as a spurious counter example.

3.2.4 Refinement of Abstract Model

In this step, we have to generate a refined abstract model which does not admit the spurious counter example (we call it the spurious CE free model for a given CE). When a counter example is judged as a spurious counter example, there is a Bad State $\hat{l}_b$ which has

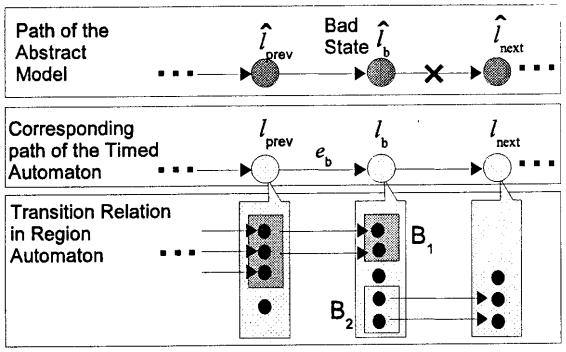

Figure 6 Counter Example

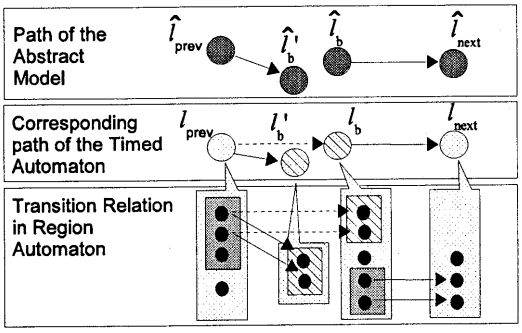

Figure 7 Refined Model

Refinement

Inputs $\mathscr{A}_i, h, B_1 = (l_b, D_1), e_b = (l_{prev}, a, g, r, l_b)$

$\{e_b$ = a transition to $l_b\}$

$\mathscr{A}_{i+1} := \mathscr{A}_i$

$\mathscr{A}_{i+1} :=$ DuplicateState$(\mathscr{A}_{i+1}, B)$ {Duplication of States}

$\mathscr{A}_{i+1} :=$ DuplicateTransition$(\mathscr{A}_{i+1}, B, e_b)$

{Duplication of Transitions}

$\mathscr{A}_{i+1} :=$ RemoveTransition$(\mathscr{A}_{i+1}, B)$ {Removal of Transitions}

$\hat{M}_{i+1} :=$Abstraction$(\mathscr{A}_{i+1}, h)$

return $\hat{M}_{i+1}$

Figure 8 Refinement

a corresponding state set $B_1 = (l_b, D_1)$ reachable from the initial state but unreachable to l_{next}, and another state set $B_2 = (l_b, D_2)$ unreachable from the initial state but reachable to l_{next}, are merged (mapped into the same state) as in Fig. 6.

In general, refinement algorithm should divide state $\hat{l}_b$ into more than two states as state B_1 and state B_2 are mapped into differential states. Dividing of a state space of a timed automaton usually needs Subtraction operation of DBM. However, DBM is not closed under Subtract operation [7], so applying such an approach is difficult.

We proposes another approach, in which it duplicates state B_1 in the concrete model and also performs other transformation on the concrete model. Applying the abstraction function to the transformed concrete model produces a new refinement abstract model where a state mapped from B_2 is unreachable (refer in Fig. 7).

The algorithm of Refinement in Fig. 8 consists of three sub algorithms, called duplication of states, duplication of transitions, and removal of transitions, shown in Fig.9, 10, and 11, respectively.

DuplicateState

Input $\mathscr{A}, B_1 = (l_b, D_1)$

$l'_b := newLoc()$ {Generate a new location l'_b}

$L := L \cup \{l'_b\}$

$I(l'_b) := c(D_b)$ {A set of inequalities representing D_b}

Figure 9 Duplication of States

DuplicateTransition

Inputs $\mathscr{A}, B_1 = (l_b, D_1), e_b = (l_{prev}, a, g, r, l_b)$

$\{e_b$ = a transition to $l_b\}$

$T := T \cup \{(l_{prev}, a, g, r, l'_b)\}$

{Duplicate a transition e_b to a $BadState$}

foreach $(l_1, a', g', r', l_2) \in T$ such that $l_1 = l_b$ **do**

if Reach$(\mathscr{A}, (l_b, D_b), (l_1, a', g', r', l_2)) \neq \emptyset$ **then**

$T := T \cup \{(l'_b, a', g', r', l_2)\}$

{duplicate transitions from l_b only enable from $((l'_b, D_b).)$}

end if

end for

Figure 10 Duplication of Transitions

RemoveTransition

Inputs $\mathscr{A}, B_1 = (l_b, D_1), e_b = (l_{prev}, a, g, r, l_b)$

$\{e_b$ = a transition to $l_b\}$

$Prev := (l_{prev}, D_{Inv})$

{a set of every region satisfying an invariant of l_{prev}}

$R :=$Reach$(\mathscr{A}, Prev, e_b)$ {obtain regions of l_b reachable from $Prev$}

if $relation(R, B_1) =$ ⟨**true**, **true**⟩ **then**

{when $R = B$, $relation(R, B_1)$ returns ⟨ **true**, **true**⟩.}

$T := T \setminus \{(l, a, g, r, l_b)\}$

end if

Figure 11 Removal of Transitions

Here, we gives definitions of states to duplicate, transitions to duplicate, and transitions to remove.

Definition 3.5 (States to Duplicate). *Let $B_1 = (l_b, D_1)$ and duplication of a location l_b be l'_b. A set of states to duplicate, of a region automaton is defined as (l'_b, D_1).*

Duplication of transition duplicates the following kinds of transitions: "transitions from l_{prev} to l_b," and " transitions not only from l_b but also enable from (l_b, D_1)."

Definition 3.6 (Transitions to Duplicate). *For a region automaton $\mathscr{A}_r = (L_r, l_{r\,0}, T_r, A)$, $B_1 = (l_b, D_1)$, states to duplicate (l'_b, D_1), and a previous location l_{prev} of a location l_b in a counter example, transitions to duplicate of a region automaton is defined as follows:*

$T_{r\,d} = \{((l_{prev}, [v]) \stackrel{a}{\Rightarrow} (l'_b, [v']) \mid \forall (l_{prev}, [v]) \in (l_{prev}, D_{Inv}).$
$\forall (l_b, [v']) \in (l_b, D_1).(l_{prev}, [v]) \stackrel{a}{\Rightarrow} (l_b, [v']) \in T_r\} \cup \{(l'_b, [v]) \stackrel{a}{\Rightarrow}$
$(l, [v']) \mid \forall (l_b, [v]) \in (l_b, D_1).\ \forall (l, [v']) \in L_r.\ (l_b, [v]) \stackrel{a}{\Rightarrow}$
$(l, [v']) \in T_r\}$.

Definition 3.7 (Transitions to Remove). *For a region automaton $\mathscr{A}_r = (L_r, l_{r\,0}, T_r, A)$, $B_1 = (l_b, D_d)$, states to duplicate (l'_b, D_1), and a previous location l_{prev} of a location in a counter example, transitions to remove of a region automaton is defined as follows:*

$T_{r\,r} = \{(l_{prev}, [v]) \stackrel{a}{\Rightarrow} (l_b, [v']) \mid \forall (l_{prev}, [v]) \in (l_{prev}, D_{Inv}).$

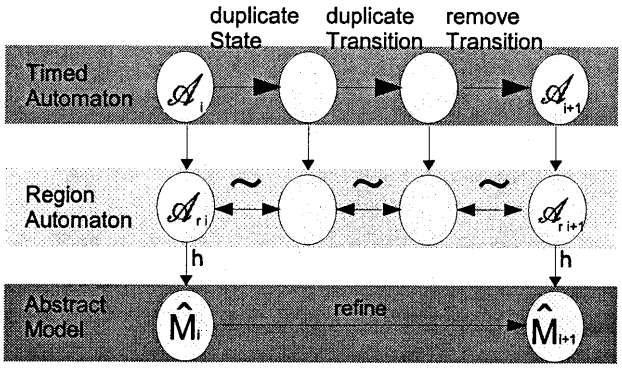

Figure 12 Relations among models

$(l_{prev}, [v]) \stackrel{a}{\Rightarrow} (l_b, [v']) \in T_r\}$.

The algorithm of Removal of Transitions removes transitions only when a set of states reachable from l_{prev} is the same as a set (l_b, D_1) of Bad States. Therefore, for every $(l_{prev}, [v]) \stackrel{a}{\Rightarrow} (l_b, [v']) \in T_{r\,r}$, $(l_b, [v']) \in (l_b, D_1)$ holds. It means that every transition in $T_{r\,r}$ has its duplication in $T_{r\,d}$.

4. Correctness Proof

As mentioned in Section3., the proposed algorithm checks a property $\mathtt{AG} \bigvee_{e \in E} \neg e$, where E ($\subset L$) of a timed automaton $\mathscr{A}$ is a set of error locations of the target system.

Paper [2] gives a theorem on a conservative class of abstractions which attempts to preserve semantics of automata against state reductions under the condition that it checks only a property $\mathtt{AG}\ p$ for a proposition p.

From the theorem, we can derive the following theorem.

Theorem 4.1. *For a timed automaton $\mathscr{A}$ and a set E of error locations. Let the abstract model and a set of error states of the abstract model be $\hat{M}$ and $\hat{E} = \{h(e) \mid e \in E\}$, respectively. The following statement always holds.*

$$\hat{M} \models \mathtt{AG} \bigvee_{\hat{e} \in \hat{E}} \neg \hat{e} \Rightarrow \mathscr{A} \models \mathtt{AG} \bigvee_{e \in E} \neg e \qquad (1)$$

Proof. Let a concrete model and its abstract model abstracted by h be M and $\hat{M}$, respectively. For a proposition p, if an abstraction function h satisfies the following for every $s \in S$:

$$h(s) \models p \Rightarrow s \models p \qquad (2)$$

then $\hat{M} \models \mathtt{AG}\ p \Rightarrow M \models \mathtt{AG}\ p$ holds from Theorem 1 in Paper [2].

Here we assume that $p = \bigvee_{\hat{e} \in \hat{E}} \neg \hat{e}$ for $\hat{M}$, and $p = \bigvee_{e \in E} \neg e$ for $\mathscr{A}$. In addition, an abstraction function defined in Definition 3.2 maps each location in $\mathscr{A}$ to a state $\hat{M}$ and the mapping is one-to-one mapping. Thus, $h(l) = \hat{e} \iff l = e$ holds. As a result, the abstraction function h satisfies the statement 2; Theorem 4.1 is proved. □

Lemma 4.1 (Bi-simulation equivalence among timed automata). *Let denote by $\mathscr{A}_i$ and $\mathscr{A}_{i+1}$ a timed automaton before applying i+1-th application of Refinement and one after applying $i+1$-th application of Refinement, respectively. $\mathscr{A}_i$ is bi-simulation equivalent to $\mathscr{A}_{i+1}$.*

Proof. Let denote by $\mathscr{A}_{r\,i}$ and $\mathscr{A}_{r\,i+1}$ their region automaton for $\mathscr{A}_i$ and $\mathscr{A}_{i+1}$, respectively. In a similar way, $\mathscr{A}_i^1$, $\mathscr{A}_{r\,i}^1$, $\mathscr{A}_i^2$, $\mathscr{A}_{r\,i}^2$, $\mathscr{A}_i^3 (= \mathscr{A}_{i+1})$, $\mathscr{A}_{r\,i}^3 (= \mathscr{A}_{r\,i+1})$ are defined, where the superfix means a sub algorithm of the Refinement. Therefore the superfixes 1, 2, and 3 stand for after applying Duplication of States, Duplication of Transitions, and Removal of Transition, respectively.

We will prove that $\mathscr{A}_i$ is bi-simulation equivalent to $\mathscr{A}_{i+1}$ by proving bi-simulation equivalence over the corresponding region automata. For l_b, let l_b' be a duplicated state. For a set D_1 of regions which associates to a location to duplicate, a set of states in $\mathscr{A}_r$ will be (l_b, D_1), and (l_b', D_1). Let $T_{r\,d}$ and $T_{r\,r}$ be a set of transitions be added in $\mathscr{A}_r$ and that to be removed in $\mathscr{A}_r$, respectively.

i) $\mathscr{A}_{r\,i}$ and $\mathscr{A}_{r\,i}^1$

Let's consider $\mathscr{A}_{r\,i} = (L_{r\,i}, l_{r\,i\,0}, T_{r\,i}, A_i)$ and $\mathscr{A}_{r\,i}^1 = (L_{r\,i}^1, l_{r\,i\,0}^1, T_{r\,i}^1, A_i^1)$. From the assumption, $l_{r\,i\,0} = l_{r\,i\,0}^1$, $T_{r\,i} = T_{r\,i}^1$, $A_i = A_i^1$ and $L_{r\,i}^1 = L_{r\,i} \cup (l_b', D_1)$ hold.

The initial state $l_{r\,i\,0} = l_{r\,i\,0}^1$, and $T_{r\,i} = T_{r\,i}^1$. So, there is no transition to the duplicated state set (l_b', D_1) in $\mathscr{A}_{r\,i}^1$. Thus, there is bi-simulation equivalence between $\mathscr{A}_{r\,i}$ and $\mathscr{A}_{r\,i}^1$.

ii) $\mathscr{A}_{r\,i}^1$ and $\mathscr{A}_{r\,i}^2$

For $\mathscr{A}_{r\,i}^2 = (L_{r\,i}^2, l_{r\,i\,0}^2, T_{r\,i}^2, A_i^2)$, obviously $L_{r\,i}^2 = L_{r\,i}^1$ and $l_{r\,i\,0}^2 = l_{r\,i\,0}^1$, $A_i^2 = A_i^1$ hold. $T_{r\,i}^2 = T_{r\,i}^1 \cup T_{r\,d}$ also holds.

We show that for every $[v] \in D_1$, a state $(l_b, [v])$ and a state $(l_b', [v])$ have a bi-simulation equivalence relation. When there exists a transition $(l_b, [v]) \stackrel{a}{\Rightarrow} (l, [v'])$, as defined in definition 3.6, the corresponding transition $(l_{b'}, [v]) \Rightarrow (l, [v'])$ is generated. Also, when there exists a transition $(l_b', [v]) \stackrel{a}{\Rightarrow} (l, [v'])$, there must be an original transition $(l_b, [v]) \stackrel{a}{\Rightarrow} (l, [v'])$. Thus, we proved the first goal.

Thus, the concrete bi-simulation equivalence relation $\sim$ between $l_{r\,i}^1 \in L_{r\,i}^1$ and $l_{r\,i}^2 \in L_{r\,i}^2$ is defined as follows:

$$l_{r\,i}^1 \sim l_{r\,i}^2 \iff l_{r\,i}^1 = l_{r\,i}^2 \ \mathtt{or}\ l_{r\,i}^2 \ \mathtt{is\ duplication\ of}\ l_{r\,i}^1 \qquad (3)$$

For the initial states, $l_{r\,i\,0}^1 \sim l_{r\,i\,0}^2$ holds. A transition set $T_{r\,i}^1$ satisfies $T_{r\,i}^1 \subset T_{r\,i}^2$. For each transition in $T_{r\,i}^1$, thus, there is a corresponding transition in $T_{r\,i}^2$. Suppose that $l_{r\,i}^1 \sim l_{r\,i}^2$ and $l_{r\,i}^1 \stackrel{a}{\Rightarrow} l_{r\,i}^{1\prime}$. Then there exists a transition $l_{r\,i}^2 \stackrel{a}{\Rightarrow} l_{r\,i}^{2\prime}$ and $l_{r\,i}^{1\prime} \sim l_{r\,i}^{2\prime}$. Let consider converse. For each transition in $T_{r\,i}^2$, there is the corresponding transition in $T_{r\,i}^1$. Please note that for a transition in $T_{r\,d}$, there exists the original transition. Suppose that $l_{r\,i}^1 \sim l_{r\,i}^2$ and $l_{r\,i}^2 \stackrel{a}{\Rightarrow} l^{2\prime}$. Then there exists a transition $l_{r\,i}^1 \stackrel{a}{\Rightarrow} l_{r\,i}^{1\prime}$ and $l_{r\,i}^{1\prime} \sim l_{r\,i}^{2\prime}$.

Therefore, $\mathscr{A}_{r\,i}^1$ and $\mathscr{A}_{r\,i}^2$ are bi-simulation equivalent.

iii) $\mathscr{A}_{r\,i}^2$ and $\mathscr{A}_{r\,i}^3$

Let's consider $\mathscr{A}^3_{r\,i} = (L^3_{r\,i}, l^3_{r\,i\,0}, T^3_{r\,i}, A^3_i)$. Obviously $L^3_{r\,i} = L^2_{r\,i}$, $l^3_{r\,i\,0} = l^2_{r\,i\,0}$ and $A^3_i = A^2_i$ hold. $T^3_{r\,i} = T^1_{r\,i} \setminus T_{r\,r}$ also holds.

The case when the algorithm in Fig. 11 does not perform any removal of transitions is trivial. $\mathscr{A}^2_{r\,i}$ is equivalent to $\mathscr{A}^3_{r\,i}$, thus also holds the relation $\sim$.

Otherwise, in other words, in the case of removal of a transition, from Definition 3.7, each element in $T_{r\,r}$ has its duplication in $T_{r\,d}$. Thus, even if the transition is removed, $\sim$ is also preserved between $(l_{prev}, [v]) \in (l_{prev}, D_{Inv})$ of $\mathscr{A}^2_{r\,i}$ and $(l_{prev}, [v]) \in (l_{prev}, D_{Inv})$ of $\mathscr{A}^3_{r\,i}$. Thus each state of $L^2_{r\,i}$ and that of $L^3_{r\,i}$ satisfy the relation defined in (3). In a similar way of case ii), $\mathscr{A}^2_{r\,i}$ and $\mathscr{A}^3_{r\,i}$ are bi-simulation equivalent.

From the facts i), ii) and iii), we can conclude that $\mathscr{A}_{r\,i}$ and $\mathscr{A}_{r\,i}$ are bi-simulation equivalent. □

Lemma 4.2. *At most n times repetition of Refinement yields the spurious CE free model, where n is the length of the spurious counter example.*

Proof. Let $\mathscr{A}$, $\mathscr{A}_r$ and $\hat{M}$ be a timed automaton, its region automaton and its abstract model, respectively. For a counter example $\hat{T} = \langle \hat{s}_0, \hat{s}_1, \cdots, \hat{s}_n \rangle$, where $\hat{s}_n$ is an abstract state obtained by reducing the error location, let consider one of the corresponding sequences $t = (l_0 \xrightarrow{a_1, g_1, r_1} l_1 \xrightarrow{a_2, g_2, r_2} \cdots \xrightarrow{a_n, g_n, r_n} l_n)$ to $\hat{T}$ on $\mathscr{A}$, where l_n is error location. Let R_i be a set of reachable i-th states along with the sequence t, and UR_i be that of unreachable $(= (l_i, D_{Inv}) \setminus R_i)$.

We prove that "for sub-sequence starting from l_0 to $l_k (1 \leqq k \leqq n)$ of t, by applying at most k times repetition of Refinement yields that it is reachable to an abstract state corresponding to R_k but unreachable to an abstract state corresponding to UR_k." (*)

Let duplicated location from R_i be l'_i. Let the abstract state of l'_i be $\hat{s}'_i (= h(l'_i))$.

i) $k = 1$

$R_0 = (l_0, D_{Inv})$ holds. A set of reachable states from (l_0, D_{Inv}) through a transition $(l_0, a_1, g_1, r_1, l_1)$ is in fact R_1 from the definition of R_i. Therefore, Refinement duplicates R_1, which is a location l'_1 and Refinement also removes a transition from l_0 to l_1. In the obtained abstract model, it is reachable to only $\hat{s}'_1$ corresponding to R_1, and it is unreachable to a state $h(l_1)$ corresponding to UR_1.

ii) $k \geqq 2$

As inductive assumption, we assume that at most $k-1$ times repetition of Refinement yields that it is reachable to an abstract state corresponding to R_{k-1} but unreachable to an abstract state corresponding to UR_{k-1}.

Let $R'_k (\supseteq R_k)$ be a set of reachable states from (l_{k-1}, D_{Inv}). If $R_k = R'_k$, then in a similar way as $k = 1$, applying one more Refinement leads to the goal.

Let consider when $R_k \subset R'_k$ holds. A transition from l_{k-1} to l_k cannot be removed because UR_k is reachable from (l_{k-1}, D_{Inv}). In such a case, from the inductive assumption, we can obtain the refined abstract model, in which an abstract state corresponding to R_{k-1} is reachable but UR_{k-1} is not. Let l'_{k-1} and l'_k be duplicated locations of R_{k-1} in $k-1$-th time-Refinement and R_k in k-th time-Refinement, respectively. Adding transition from l'_{k-1} to l'_k improves the model so that it is reachable to only a state corresponding to R_k.

From (i) and (ii), statement (*) is proved.

If the counter example is spurious, it is unreachable from R_{n-1} to error state (l_n, D_{Inv}) in M. Similarly, in $\hat{M}$, it is unreachable from $\hat{s}'_{n-1}$ to $\hat{s}_n$. Thus the lemma is proved. □

Theorem 4.2 (Correctness). *If a counter example is spurious, at most n times repetition of Refinement in Fig. 8 yields a spurious CE free model.*

Proof. From Lemma 4.1, Refinement preserves bi-simulation equivalence. From Lemma 4.2, at most n times repetition of Refinement yields a refined spurious CE free model. □

5. Conclusion

This paper gives a formal description and correctness proof of our proposed CEGAR algorithm in [8].

The future work will be extension of our algorithm to handle integer variables used in UPPAAL timed automata.

Acknowledgment

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.

References

[1] E M. Clarke, O. Grumberg, S. Jha, Y. Lu, and V. Helmut: "Counterexample-guided Abstraction Refinement," In Proc. of the 12th Int. Conf. on Computer Aided Verification, vol.1855, pp.154-169, July, 2000.

[2] E M. Clarke, A, Gupta, J. Kukula, and O. Strichman: "SAT based Abstraction-Refinement using ILP and Machine Learning Techniques," In Proc. of the 14th Int. Conf. on Computer Aided Verification, vol.2404, pp.695-709, July, 2002.

[3] E M. Clarke, O. Grumberg, D A. Peled: "Model Checking," MIT Press, 2000.

[4] R. Alur: "Techniques for Automatic Verification of Real-Time Systems," PhD thesis, Stanford University, 1991.

[5] 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, pp.414-425, IEEE Computer Society Press, 1990.

[6] J. Bengtsson, and W .Yi: "Timed Automata: Semantics, Algorithms and Tools," In Lectures on Concurrency and Petri Nets, vol.3098, pp.87-124, 2004.

[7] 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, pp.128-142, 2002

[8] T. Nagaoka, K. Okano, and S. Kusumoto: "Abstraction of Extended Timed Automata for UPPAAL Based on Counterexample-Guided Abstraction Refinement Loop (in Japanese)," IEICE Technical Report, Vol.107, No.176, pp.77-82, 2007.