Reducing Clocks in Timed Automata while Preserving Bisimulation

Shibasish Guha Chinmay Narayan S. Arun-Kumar

Indian Institute of Technology Delhi
{shibasish, chinmay, sak}@cse.iitd.ac.in

Abstract. Model checking timed automata becomes increasingly complex with the increase in the number of clocks. Hence it is desirable that one constructs an automaton with the minimum number of clocks possible. The problem of checking whether there exists a timed automaton with a smaller number of clocks such that the timed language accepted by the original automaton is preserved is known to be undecidable. In this paper, we give a construction, which for any given timed automaton produces a timed bisimilar automaton with the least number of clocks. Further, we show that such an automaton with the minimum possible number of clocks can be constructed in time that is doubly exponential in the number of clocks of the original automaton.

1 Introduction

Timed automata is a formalism for modelling and analyzing real time systems. The complexity of model checking is dependent on the number of clocks of the timed automaton (TA). Many model checking and reachability problems use a region graph or a zone graph (for the timed automaton) whose sizes are exponential in the number of clocks. Hence it is desirable to construct a timed automaton with the minimum number of clocks that preserves some property of interest. It is known that given a timed automaton, checking whether there exists another timed automaton accepting the same timed language as the original one but with a smaller number of clocks is undecidable. In this paper, we show that checking the existence of a timed automaton with a smaller number of clocks that is timed bisimilar to the original timed automaton is however decidable. Our method is constructive and we provide a 2-EXPTIME algorithm to construct the timed bisimilar automaton with the least possible number of clocks. We also note that if the constructed automaton has a smaller number of clocks, then it implies that there exists an automaton with a smaller number of clocks accepting the same timed language.

Related work: In , an algorithm has been provided to reduce the number of clocks of a given timed automaton and produce a new timed automaton that is timed bisimilar to the original one. The algorithm detects a set of active clocks at every location and partitions these active clocks into classes such that all the clocks belonging to a class in the partition always have the same value. However,
this may not result in the minimum possible number of clocks since the algorithm works on the timed automaton directly rather than on its semantics. Thus if a constraint associated with clock \( x \) implies a constraint associated with clock \( y \), and both of them appear on an edge, then the constraint with clock \( y \) can be eliminated. However, the algorithm of \[6\] does not capture such implication. Also by considering constraints on more than one outgoing edge from a location, e.g. \( l_0 \xrightarrow{a,x \leq 3,0} l_1 \) and \( l_0 \xrightarrow{a,x > 3,0} l_2 \) collectively, we may sometimes eliminate the constraints that may remove some clock. This too has not been accounted for by the algorithm of \[6\].

In \[17\], it has been shown that no algorithm can decide the minimality of the number of clocks while preserving the timed language and for the non-minimal case find a timed language equivalent automaton with fewer clocks. Also for a given timed automaton, the problem of finding whether there exists another TA with fewer clocks accepting the same timed language is undecidable \[9\].

Another result appearing in \[14\] which uses the region-graph construction is the following. A \((C,M)\)-automaton is one with \( C \) clocks and \( M \) is the largest integer appearing in the timed automaton. Given a timed automaton \( A \), a set of clocks \( C \) and an integer \( M \), checking the existence of a \((C,M)\)-automaton that is timed bisimilar to \( A \) is shown to be decidable in \[14\]. The method in \[14\] constructs a logical formula called the characteristic formula and checks whether there exists a \((C,M)\)-automaton that satisfies it. Further, it is shown that a pair of automata satisfying the same characteristic formula are timed bisimilar. Our result and method differ from the above paper in the following three ways: (1) Given a timed automaton \( A \), we construct a timed automaton \( B \) with the least number of clocks such that \( B \) is timed bisimilar to \( A \). (2) We use a zone graph which is usually a much more succinct representation of the state space of the timed automaton. (3) Our method does not involve any intermediate step of creating a logical formula.

The rest of the paper is organized as follows: in Section 2, we describe timed automata and introduce several concepts that will be used in the paper. We also describe the construction of the zone graph used in reducing the number of clocks. In Section 3, we discuss our approach in detail along with a few examples. Section 4 is the conclusion. Due to constraint in space, the proofs have been relegated to the appendix.

2 Timed Automata

Formally, a \textit{timed automaton} (TA) is defined as a tuple \( A = (L, Act, l_0, E, C) \) where \( L \) is a finite set of locations, \( Act \) is a finite set of visible actions, \( l_0 \in L \) is the initial location, \( E \subseteq L \times B(C) \times Act \times 2^L \times L \) is a finite set of \textit{edges} and \( C \) is a finite set of clocks. The set of constraints or guards on the edges \( B(C) \) is given by the grammar \( g ::= x \sim k \mid g \land g \), where \( k \in \mathbb{N} \) and \( x \in C \) and \( \sim \in \{\leq, <, =, >, \geq\} \). Given two locations \( l, l' \), a transition from \( l \) to \( l' \) is of the form \((l, g, a, R, l')\) i.e. a transition from \( l \) to \( l' \) on action \( a \) is possible if the
constraints specified by $g$ are satisfied; $R \subseteq C$ is a set of clocks which are reset to zero during the transition.

The semantics of a timed automaton (TA) is described by a timed labelled transition system (TLTS) \cite{11}. The timed labelled transition system $T(A)$ generated by $A$ is defined as $T(A) = (Q, Lab, Q_0, \{\stackrel{\alpha}{\rightarrow} | \alpha \in Lab\})$, where $Q = \{(l, v) | l \in L, v \in \mathbb{R}_{\geq 0}^{|C|}\}$ is the set of states, each of which is of the form $(l, v)$, where $l$ is a location of the timed automaton and $v$ is a valuation assigned to the clocks of $A$; $Lab = Act \cup \mathbb{R}_{\geq 0}$ is the set of labels. Let $v_0$ denote the valuation such that $v_0(x) = 0$ for all $x \in C$. $Q_0 = (l_0, v_0)$ is the initial state of $T(A)$. A transition may occur in one of the following ways:

(i) Delay transitions : $(l, v) \xrightarrow{d} (l, v + d)$. Here, $d \in \mathbb{R}_{\geq 0}$ and $v + d$ is the valuation in which the value of every clock is incremented by $d$.

(ii) Discrete transitions : $(l, v) \xrightarrow{a} (l', v')$ if for an edge $e = (l, g, a, R, l') \in E$, $v \models g, v' = v_{R-\pi}$, where $v_{R-\pi}$ denotes that every clock in $R$ has been reset to 0, while the remaining clocks are unchanged. From a state $(l, v)$, if $v \models g$, then there exists an $a$-transition to a state $(l', v')$; after this, the clocks in $R$ are reset while those in $C \setminus R$ remain unchanged.

For simplicity, we do not consider annotating locations with clock constraints (known as invariant conditions \cite{13}). Our results extend in a straightforward manner to timed automata with invariant conditions. In Section 3, we provide the modifications to our method for dealing with location invariants. We now define various concepts that will be used in the rest of the paper.

**Definition 1.** Let $A = (L, Act, l_0, E, C)$ be a timed automaton, and $T(A)$ be the TLTS corresponding to $A$.

1. **Timed trace:** A sequence of delays and visible actions $d_1a_1d_2a_2 \ldots d_na_n$ is called a timed trace if there is a sequence of transitions $p_0 \xrightarrow{d_1} p_1 \xrightarrow{a_1} p_1' \xrightarrow{d_2} \ldots \xrightarrow{a_n} p_n \xrightarrow{d_{n+1}} p_{n+1}'$ in $T(A)$, with $p_0$ being the initial state of the timed automaton.

2. **Zone:** A zone $Z$ is a set of valuations $\{v \in \mathbb{R}_{\geq 0}^{|C|} | v \models \beta\}$, where $\beta$ is of the form $\beta := x \sim k \parallel x - y \sim k \parallel \beta \land \beta$, $k$ is an integer, $x, y \in C$ and $\sim \in \{\leq, <, =, >, \geq\}$. $Z^\uparrow$ denotes the future of the zone $Z$. $Z^\uparrow := \{v + d \parallel v \in Z, d \geq 0\}$ is the set of all valuations reachable from $Z$ by time elapse. A zone by definition is a convex set.

3. **Pre-stability:** A zone $Z_1$ of location $l_1$ is pre-stable with respect to another zone $Z_2$ of location $l_2$ if $Z_1 \subseteq \text{preds}(Z_2)$ or $Z_1 \cap \text{preds}(Z_2) = \emptyset$ where $\text{preds}(Z) \overset{def}{=} \{v \in \mathbb{R}_{\geq 0}^{|C|} | \exists v' \in Z \text{ such that } (l, v) \xrightarrow{\alpha} (l', v'), \alpha \in Lab\}$. Here $l = l'$ if $\alpha$ is a delay action.

4. **Canonical decomposition:** Let $g = \bigwedge_{i=1}^{n} \gamma_i \in \mathcal{B}(C)$, where each $\gamma_i$ is an elementary constraint of the form $x_i \sim k_i$, such that $x_i \in C$ and $k_i$ is a non-negative integer. A canonical decomposition of a zone $Z$ with respect to $g$ is obtained by splitting $Z$ into a set of zones $Z_1, \ldots, Z_m$ such that for each $1 \leq j \leq m$, and $1 \leq i \leq n$, either $\forall v \in Z_j, v \models \gamma_i$ or $\forall v \in Z_j, v \not\models \gamma_i$. For example, consider the zone $Z = x \geq 0 \land y \geq 0$ and the guard $x \leq 2 \land y > 1$. 

Z is split with respect to \( x \leq 2 \), and then with respect to \( y > 1 \), hence into four zones: \( x \leq 2 \land y \leq 1 \), \( x > 2 \land y \leq 1 \), \( x \leq 2 \land y > 1 \) and \( x > 2 \land y > 1 \).

An elementary constraint \( x_i \prec k_i \) induces the hyperplane \( x_i = k_i \) in the zone graph of the timed automaton.

5. Zone graph: Given a timed automaton \( A = (L, l_0, E, I) \), a zone graph \( G_A \) of \( A \) is a transition system \((S, s_0, \text{Lep}, \rightarrow)\), that is a finite representation of \( T(A) \). Here \( \text{Lep} = \text{Act} \cup \{\varepsilon\} \). \( G_A \) consists of nodes and transitions which are the edges of \( G_A \). \( S \subseteq L \times \{Z\} \) is the set of nodes, \( \{Z\} \) being the set of zones of \( A \). The node \( s_0 = (l_0, Z_0) \) is the initial node such that \( v_0 \in Z_0 \).

\( (l_i, Z) \xrightarrow{a} (l_j, Z') \) iff \( l_i \xrightarrow{a \cdot R} l_j \) in \( A \) and \( Z' \subseteq ([Z \cap g]_{R=\bar{R}}) \) obtained after canonical decomposition of \( ([Z \cap g]_{R=\bar{R}}) \). For any \( Z \) and \( Z' \), \( (l_i, Z) \xrightarrow{\varepsilon} (l_i, Z') \) iff there exists a delay \( d \) and a valuation \( v \) such that \( v \in Z \), \( v + d \in Z' \) and \( (l_i, v) \xrightarrow{d} (l, v + d) \) is in \( T(A) \). Here the zone \( Z' \) is called a delay successor of zone \( Z \), while \( Z \) is called the delay predecessor of \( Z' \). We call a zone \( Z \) corresponding to a location to be a base zone if \( Z \) does not have a delay predecessor. The relation \( \varepsilon \) is reflexive and transitive and so is the delay successor relation. A zone \( Z' \) is called the immediate delay successor of a zone \( Z \) iff \( Z' \) is a delay successor of \( Z \) and there does not exist any zone \( Z'' \) such that \( Z'' \) is a delay successor of \( Z \), \( Z' \) is a delay successor of \( Z'' \) and \( Z' \neq Z'' \neq Z \).

6. A hyperplane \( x = k \) is said to bound a zone \( Z \) from above if \( k \) is the smallest integer such that \( \exists \) valuation \( v \in Z \) such that \( v \models x \leq k \) \((v \models x < k)\) but for all positive delays \( d \) so that \( (v + d)(x) > k \) \((v + d)(x) \geq k)\).

A zone, in general, can be bounded above by several hyperplanes.

A hyperplane \( x = k \) is said to bound a zone \( Z \) fully from above if \( k \) is the smallest integer such that \( \forall \) valuations \( v \in Z \) such that \( v \models x \leq k \) \((v \models x < k)\) but for all positive delays \( d \) so that \( (v + d)(x) > k \) \((v + d)(x) \geq k), v + d \notin Z \).

Vis-à-vis, we can also say that a hyperplane \( x = k \) bounds a zone from below if \( k \) is the largest integer and there exists a valuation \( v \in Z \) such that \( v \models x \geq k \) \((v \models x > k)\) but for all positive delays \( d \) so that \( (v - d)(x) < k \) \((v - d)(x) \leq k), v - d \notin Z \). Similarly, we can also define a hyperplane bounding a zone fully from below.

When not specified otherwise, in this paper, a hyperplane bounding a zone implies that it bounds the zone from above. A zone \( Z \) is said to be bounded above if it has an immediate delay successor zone.

We create a zone graph such that for any location \( l \) the zones \( Z \) and \( Z' \) of any two nodes \((l, Z)\) and \((l, Z')\) in the zone graph are disjoint and all zones of the zone graph are pre-stable. This zone graph is constructed in two phases in time exponential in the number of the clocks. The first phase performs a forward analysis of the timed automaton while the second phase ensures pre-stability in the zone graph. The forward analysis may cause a zone graph to become infinite \([10]\). Several kinds of abstractions have been proposed in the literature \([5,10,11]\) to make the zone graph finite. We use location dependent maximal constants abstraction \([10]\) in our construction. In phase 2 of the zone graph creation, the
zones are further split to ensure that the resultant zone graph is pre-stable. The following lemma states an important property of the zone graph which will further be used for clock reduction.

Lemma 1. Pre-stability ensures that there exists a hyperplane \( x = h \), where \( x \) is some clock and \( h \in \mathbb{Z} \) such that a zone is bounded fully from above by the hyperplane \( x = h \).

Some approaches for preserving convexity and implementing pre-stability have been discussed in [18]. As an example, consider the timed automaton in Figure 1. The pre-stable zones of location \( l_1 \) are shown in the right side of the figure. An algorithmic procedure for the construction of the zone graph is given in [12].

A relation \( R \subseteq Q \times Q \) is a timed simulation relation if the following conditions hold for any two timed states \((p, q) \in R\).

\[
\forall a \in \text{Act}, \ p \xrightarrow{a} p' \implies \exists q' : q \xrightarrow{a} q' \text{ and } (p', q') \in R\text{ and}
\]

\[
\forall d \in \mathbb{R}_{\geq 0}, \ p \xrightarrow{d} p' \implies \exists q' : q \xrightarrow{d} q' \text{ and } (p', q') \in R.
\]

A timed bisimulation relation is a symmetric timed simulation. Two timed automata are timed bisimilar if and only if their initial states are timed bisimilar.

### 3 Clock Reduction

Unlike the method described in [6], which works on the syntactic structure of the timed automaton, we use a semantic representation, the zone graph described in Section 2, to capture the behaviour of the timed automaton. This helps us to reduce the number of clocks in a more effective way. For a given TA \( A \), we first describe a sequence of stages to construct a TA \( A_4 \) that is timed bisimilar to \( A \). Later we prove the minimality in terms of the number of clocks for the TA \( A_4 \). The operations involved in our procedure use a difference bound matrix (DBM) [4,7] representation of the zones. A DBM for a set \( C = \{x_1, x_2, \ldots, x_n\} \) of \( n \)
Clocks is an \((n + 1)\) square matrix \(E\) where an extra clock \(x_0\) is introduced such that clock \(x_0\) is always 0. An element \(E_{ij}\) is of the form \((m_{ij}, \prec)\) where \(\prec \in \{<, \leq\}\) such that \(x_i - x_j \prec m_{ij}\). The following are important considerations in reducing the number of clocks:

- There may be some clock constraints on an edge of the timed automaton that do not affect the transition. Such constraints may be removed. (Stage 1)
- At some location, some clocks whose values may be expressed in terms of other clock values, may be removed. (Stage 2)
- Splitting some locations may reduce the number of constraints. (Stage 2)
- Two or more transitions when considered collectively may lead to the removal of some constraints. (Stage 3)
- An efficient way of renaming the clocks across all locations can reduce the total number of clocks further. (Stage 4)

Given a TA, we apply the operations described in the following stages in sequence to obtain the TA \(A_4\).

**Stage 1: Removing unreachable edges and associated constraints:**

This stage involves creating the zone graph of the given timed automaton. The edges and their associated constraints that are never enabled in an actual transition are removed while creating the zone graph. Suppose there is an edge \(l_i \xrightarrow{\text{g}, a, R} l_j\) in \(A\) but in the zone graph, a corresponding transition of the form \((l_i, Z) \xrightarrow{a} (l_j, Z')\) does not exist. This implies that the transition \(l_i \xrightarrow{\text{g}, a, R} l_j\) is never enabled and hence is removed from the timed automaton. Since the edges that do not affect any transition get removed during this stage, we have the following lemma trivially.

**Lemma 2.** The operations in stage 1 produce a timed automaton \(A_1\) that is timed bisimilar to the original TA \(A\).

The time required in this stage is proportional to the size of the zone graph and hence exponential in the number of clocks of the timed automaton.

**Stage 2: Splitting locations and removing constraints not affecting transitions:**

Locations may also require to be split in order to reduce the number of clocks of a timed automaton. Let us consider an example timed automaton of Figure 1 and its zone graph in Figure 2. There are three base zones corresponding to location \(l_1\) in the zone graph, i.e. \(Z_1 = \{0 \leq y - x < 2, x \leq 5\}\), \(Z_2 = \{y - x = 2, x \leq 5\}\) and \(Z_3 = \{2 < y - x \leq 4, y \leq 7\}\). This stage splits \(l_1\) into three locations \(l_{1_1}, l_{1_2}\) and \(l_{1_3}\) (one for each base zones \(Z_1, Z_2\) and \(Z_3\)) as shown in Figure 3(a). While the original automaton, in Figure 1, contains two elementary constraints on the edge between \(l_1\) and \(l_2\), the modified automaton, in Figure 3(a), contains only one of these two elementary constraints on the outgoing edges from each of \(l_{1_1}\), \(l_{1_2}\) and \(l_{1_3}\) to \(l_2\). Subsequent stages modify it further to generate an automaton using a single clock as in Figure 3(b).

Splitting ensures that only those constraints, that are relevant for every valuation in the base zone of a newly created location, appear on the edges originating
from that location. Since clocks can be reused while describing the behaviours from the individual locations created after the split, this may lead to a reduction in the number of clocks.

We describe a formal procedure for splitting a location into multiple locations in Algorithm 1. We note that a zone can as well be perceived as a set of constraints defining it. Similarly a guard can also be considered in terms of constraints on their incoming edges. If there are constraints defining it. Similarly a guard can also be considered in terms of constraints on their incoming edges. If there are constraints defining it. Similarly a guard can also be considered in terms of constraints on their incoming edges. If there are constraints defining it. Similarly a guard can also be considered in terms of constraints on their incoming edges. If there are constraints defining it. Similarly a guard can also be considered in terms of constraints on their incoming edges. If there are constraints defining it. Similarly a guard can also be considered in terms of constraints on their incoming edges.

For each incoming edge $l_i \xrightarrow{a,g,R} l_j$, there exists a zone $Z_{r_j}$ such that $Z_{r_j}$ has an $a$ transition to $Z_{i_j}$, the $j$th base zone of $l_i$. Line 8 calculates the lower bounding hyperplane of $Z_{i_j}$ by resetting the clocks $R_j$ in the intersection of $Z_{i_j} \cup$ with $\mathbb{R}^{|Cit|}$. In Line 11, $\text{free}(Z'_{i_j}, R_j)$ represents a zone that becomes the same as $Z'_{i_j}$ after resetting the clocks in $R_j$. Further, $g_j$, is calculated as the weakest guard that simultaneously satisfies the constraints $g_r, Z_{r_j}$ and $\text{free}(Z'_{i_j}, R_j)$ and has the same set of clocks as in $g_r$. For our running example if we consider $Z_{i_j} = Z_{i}$, then we have $Z'_{i_j} = Z_{i_j} \cup \mathbb{R}^2_{|a-g_i|} = \{x = 0, y < 2\}, Z_{r_j} = \{x = y, x < 2\}$ and $\text{free}(Z'_{i_j}, \{x\}) = \{x \geq 0, y < 2\}$. We can see that $x < 2$ is the weakest formula such that $x \leq 4 \land x \geq 0 \land y < 2 \land x = y \land x < 2 \Rightarrow x < 2$ holds and hence $g_{r_j} = \{x < 2\}$.

Loop from Line 14 to 28 determines the constraints on the outgoing edges from these new locations. Line 15 checks if the zone $Z_{i_j} \cup$ has any valuation that satisfies the guard $g_i$ on an outgoing edge from location $l_i$. If no satisfying valuation exists then this transition will never be enabled from $l_i$ and hence this edge is not added in $A_2$. Loop from Line 19 to 26 checks if some elementary constraints of the guard are implied by other elementary constraints of the same guard. If it happens then we can remove those elementary constraint from the guard that are implied by the other elementary constraints.

For our running example, the modified automaton of Figure 3(a) does not contain the constraint $x > 5$ on the edge of from $l_1$ to $l_2$ even though it was
Algorithm 1 Algorithm for splitting locations

Input: Timed automaton $A_1$ obtained after stage 1

Output: Modified TA $A_2$ after applying stage 2 splitting procedures

1: $A_2 := A_1$ \hfill $A_1$ is the TA obtained from $A$ after the first two stages
2: for each location $l_i$ in $A_1$ do \hfill $i$ is the index of the location
3: \hspace{1em} Split $l_i$ into $m$ locations $l_{i_1}, \ldots, l_{i_m}$ in $A_2$ \hfill Let $m$ be the number of base zones of $l_i$
4: \hspace{1em} Remove location $l_i$ and all incoming and outgoing edges to and from $l_i$
5: for each $j$ in 1 to $m$ do
6: \hspace{1em} for each incoming edge $l_r \xrightarrow{a,g_{r_j},R_r} l_i$ in $A_1$ do
7: \hspace{2em} Split the constraints on the incoming edges to $l_i$ for the newly created locations
8: \hspace{1em} $Z'_{i_j} := Z_{i_j} \uparrow \cap \mathbb{R}_{\geq 0}^{[C]} \hfill Z_{i_j}$ be the base zone corresponding to $l_{i_j}$
9: \hspace{1em} $Z_{r_j}$ is a zone of location $l_r$ from which there is an $a$ transition to $Z_{i_j}$
10: Let $g_{r_j}$ be the weakest formula such that
11: \hspace{1em} $g_r \land \text{free}(Z'_{i_j}, R_r) \land Z_{r_j} \Rightarrow g_{r_j}$ and $g_{r_j}$ has a subset of the clocks used in $g_r$.
12: \hspace{1em} Create an edge $l_r \xrightarrow{a,g_{r_j},R_r} l_{i_j}$ in $A_2$
13: end for
14: for each outgoing edge $l_i \xrightarrow{a,g_i,R_i} l_r$ in $A_1$ do
15: \hspace{1em} if $Z_{i_j} \uparrow \cap g_i = \emptyset$ then
16: \hspace{2em} Do not create this edge from $l_{i_j}$ to $l_r$ since it is never going to be enabled for any valuation of $Z_{i_j}$;
17: else
18: \hspace{1em} Let $S_r$ be the set of elementary constraints in $g_i$
19: \hspace{1em} loop
20: \hspace{2em} if $\exists s' \in S_r$, s.t. $Z_{i_j} \uparrow \land (\bigwedge_{s \in S_r \setminus \{s'\}} s) \Rightarrow Z_{i_j} \uparrow \land s'$ then
21: \hspace{3em} $S_r = S_r \setminus \{s'\}$
22: else
23: \hspace{3em} Create an edge $l_i \xrightarrow{a,g_{i_j},R_i} l_r$ in $A_2$, where $g_{i_j} = \bigwedge_{s \in S_r}$
24: \hspace{2em} Break;
25: end if
26: end loop
27: end if
28: end for
29: end for
30: end for
present on the edge from \( l_1 \) to \( l_2 \). The reason being that the future of the zone of \( l_1 \) (that is \( 0 \leq y - x < 2 \)) along with the constraint \( y > 7 \) implies \( x > 5 \) hence we do not need to put \( x > 5 \) explicitly on the outgoing edge from \( l_1 \) to \( l_2 \). Such removal of elementary constraints helps future stages to reduce the number of clocks. The maximum number of locations produced in the timed automaton as a result of the split is of the same order as the number of zones in the zone graph. However, we note that the zones of a location in the original TA are distributed across multiple locations as a result of the split. This gives us the following lemma.

**Lemma 3.** The splitting procedure does not increase the underlying state space of the original TA.

Splitting locations and removing constraints as described above do not alter the behaviour of the timed automaton that leads us to the following lemma.

**Lemma 4.** The operations in stage 2 produce a timed automaton \( A_2 \) that is timed bisimilar to the TA \( A_1 \) obtained at the end of stage 1.

**Lemma 5.** The splitting procedure described in this stage does not increase the number of clocks but may cause the timed automaton to have \( |L_{A_2}| \) locations where \( |L_{A_2}| \) is exponential in the number of the clocks of the given timed automaton \( A \). However, there is no increase in the underlying state space of the TA.

The number of locations after the split can be exponential in the number of the clocks. The constraints on the incoming edges of \( l \) are also split appropriately into constraints on the incoming edges of the newly created locations. Hence this stage too runs in time that is exponential in the number of the clocks of timed automaton.

**Stage 3: Removing constraints by considering multiple edges with the same action:** We consider the example in Figure 4 again. Note that the constraints \( x \leq 3 \) and \( x > 3 \) on the edges from \( l_0 \) to \( l_1 \) and from \( l_0 \) to \( l_2 \) respectively could as well be merged together to produce a constraint without any clock.

For every action \( a \) enabled at any location \( l \), this stage checks whether a guard enabling that action at \( l \) can be merged with another guard enabling the same action at that location such that the timed bisimilarity is preserved. The transformation made in this stage has been formally described in Algorithm 2. The input to this algorithm is the TA obtained after stage 2, say \( A_2 \). For each location \( l_i \), the algorithm does the following: for every action \( a \in \text{Act} \), it determines the zones of \( l_i \) from which action \( a \) is enabled. We call this set \( Z_{ia} \). Zone graph construction and splitting of locations in stage 2 ensures that all zones in \( Z_{ia} \) form a linear chain connected by \( \epsilon \) edges (as shown in Figure 5). We use \( < \) to capture this total ordering relation. Let us use ordered indexed variable \( 1, \ldots, m \) to name the zones in this total order, i.e. \( Z_{i_1} < \cdots < Z_{i_k} < Z_{i_{k+1}} \cdots < Z_{i_m} \). Lemma 4 ensures that each \( Z_{i_k}, k \geq 1 \) there exists a hyperplane that bounds the zone fully from above and similarly, for each \( Z_{i_k}, k > 1 \) there exists a
hyperplane that bounds the zone fully from below. For a zone \( Z \), let \( LB(Z) \) and \( UB(Z) \) denote these lower and upper bounding hyperplanes of \( Z \) respectively.

Let \( \Gamma_{(i,a)} = \{ g \mid l_i \xrightarrow{a,g,R} l' \in E_{A_2} \} \) be the set of guards on the outgoing edges from \( l_i \) in \( A_2 \) which are labelled with \( a \). For any \( g \in \Gamma_{(i,a)} \) let us define the following:

\begin{itemize}
  \item \( Strt(g)_{(i,a)} = Z \in \mathcal{Z}_{ia} \) is the zone in \( \mathcal{Z}_{ia} \) which is bounded from below by the same constraints as the lower bound of constraints in \( g \).
  \item \( End(g)_{(i,a)} = Z \in \mathcal{Z}_{ia} \) is the zone in \( \mathcal{Z}_{ia} \) which is bounded from above by the same constraints as the upper bound of constraints in \( g \).
  \item \( Ran(g)_{(i,a)} = \{ Z \in \mathcal{Z}_{ia} \mid \text{Strt}(g)_{(i,a)} \leq Z \land Z \leq \text{End}(g)_{(i,a)} \} \cup \{ \text{Strt}(g)_{(i,a)} \} \cup \{ \text{End}(g)_{(i,a)} \} \) is the set of zones ordered by \( \prec \) relation in between \( \text{Strt}(g)_{(i,a)} \) and \( \text{End}(g)_{(i,a)} \).
\end{itemize}

In Algorithm 2 we use a rather informal notation \( g := C1, C2 \) to denote that \( C1 \) and \( C2 \) are the constraints defining the lower and upper bound of \( g \). We define a total order \( \ll \) on \( \Gamma_{(i,a)} \) such that for any \( g, g' \in \Gamma_{(i,a)} \), \( g \ll g' \) iff \( \exists Z \in \text{Ran}(g)_{(i,a)} \) such that \( Z \ll Z' \) for all \( Z' \in \text{Ran}(g')_{(i,a)} \). Similar to the zones let us use ordered indexed variable \( g_{i_1}, \ldots, g_{i_p} \) to denote \( g_{i_1} \ll \cdots g_{i_k} \ll g_{i_{k+1}} \cdots g_{i_p} \). One such total order on guards is shown in Figure 3. The loop from Line 5 to Line 38 in Algorithm 2 traverses the elements of \( \Gamma_{(i,a)} \) in this total order with the help of a variable \( \text{next} \) initialized to 2. In every iteration of this loop the invariant \( g_{\text{curr}} \ll g_{\text{next}} \) holds. Three possibilities exist based on whether the set union of zones corresponding to these guards is (i) not convex (ii) convex but non-overlapping, or (iii) convex as well as overlapping.

If the union is non-convex then both \( g_{\text{curr}} \) and index are changed in Line 7 to pick the next ordered pair in this order. For cases (ii) and (iii) new guards are created by merging corresponding zones as long as the modified automaton preserves timed bisimilarity. If timed bisimilarity is preserved then the modified automaton is set as the current automaton (Line 13 and Line 28) and next is incremented to process the next guard. Otherwise the guard \( g_{\text{curr}} \) is set to \( g_{\text{next}} \) and next is incremented by 1 (Line 15 and Line 36). Therefore the only difference in these two cases is in creating the new guard.

For case (ii), convex but non-overlapping zones, a new guard is created from the lower bound of \( \text{Strt}(g_{\text{curr}})_{(i,a)} \) and the upper bound of \( \text{End}(g_{\text{next}})_{(i,a)} \). For case (iii), there are three possibilities of combining guards, mentioned in Line 20, Line 22 and Line 24. First possibility is the same as in case (ii). Second and third possibilities are replacing the upper bound of \( g_{\text{curr}} \) with the lower bound of \( \text{Strt}(g_{\text{next}})_{(i,a)} \) and the lower bound of \( g_{\text{next}} \) with the upper bound of \( \text{End}(g_{\text{curr}})_{(i,a)} \) respectively.

A zone graph captures the behaviour of the timed automaton and hence timed bisimilarity between two TAs can be checked using their zone graphs \([19][12]\). This is the reason for creating the pre-stable zone graph as described in Section 2 as it enables one to directly check timed bisimilarity on this zone graph \([12]\).
Algorithm 2 Algorithm for stage 3

Input: Timed automaton $A_2$ obtained after stage 2
Output: Modified TA $A_3$ after applying stage 3 procedures

1: $A_3 := A_2$ \Comment{$A_2$ is the $TA$ obtained from $A$ after the first two stages}
2: for each location $l_i$ in $A_2$ do
3:   for each $a \in \text{sort}(l_i)$ do \Comment{$i$ is the index of the location, the set of locations does not change in this stage}
4:     $g_{curr} := g_{i_1}$, $next := 2$
5:     while $next < |\Gamma(l_i)| - 1$ do
6:       if $\text{Ran}(g_{curr})(l_i,a) \cup \text{Ran}(g_{i_{next}})(l_i,a)$ is not convex then
7:         $g_{curr} := g_{i_{next}}$, $next := next + 1$
8:       else if $\text{Ran}(g_{i_{next}})(l_i,a) \cap \text{Ran}(g_{curr})(l_i,a) = \emptyset$ then \Comment{non-overlapping but contiguous}
9:         $g_{curr} := [LB(\text{Strt}(g_{curr})(l_i,a)), UB(\text{End}(g_{i_{next}})(l_i,a))]$
10:        $g_{i_{next}} := g_{curr}$
11:        Let $A'$ be the $TA$ obtained by replacing all occurrences of $g_{curr}$ and $g_{i_{next}}$ with $g'_{curr}$ and $g'_{i_{next}}$ respectively in $A_3$
12:        if $A'$ is timed bisimilar to $A_3$ then
13:           $A_3 := A'$, $g_{curr} := g'_{i_{next}}$, $next := next + 1$
14:        else
15:           $g_{curr} := g_{i_{next}}$, $next := next + 1$
16:        end if
17:     else
18:        \Comment{There are three ways to combine $g_{curr}$ and $g_{i_{next}}$, and resultant new $guard$ should be checked for timed bisimilarity in the following order}
19:        (i). $g'_{curr} := [LB(\text{Strt}(g_{curr})(l_i,a)), UB(\text{End}(g_{i_{next}})(l_i,a))]$
20:        (ii). $g'_{curr} := [LB(\text{Strt}(g_{i_{next}})(l_i,a)), LB(\text{Strt}(g_{curr})(l_i,a))]$
21:        (iii). $g'_{curr} := [LB(\text{Strt}(g_{i_{next}})(l_i,a)), UB(\text{End}(g_{curr})(l_i,a))$, $UB(\text{End}(g_{i_{next}})(l_i,a))]$
22:        (iv). $g'_{curr} := [UB(\text{End}(g_{curr})(l_i,a)), UB(\text{End}(g_{i_{next}})(l_i,a))]$
23:     end if
24: while $1 \leq i \leq 3$ do \Comment{Corresponding to the three cases above}
25:     Let $A'$ be the TA obtained by replacing all occurrences of $g_{curr}$ and $g_{i_{next}}$ with $i^{th}$ $g'_{curr}$ and $g'_{i_{next}}$ respectively in $A_3$
26:     if $A'$ is timed bisimilar to $A_3$ then
27:        $A_3 := A'$, $g_{curr} := g'_{i_{next}}$, $next := next + 1$
28:     else
29:        Break
30:     end if
31:     if $i = 4$ then \Comment{Bisimilarity could not be preserved in any of these three cases}
32:        $g_{curr} := g_{i_{next}}$, $next := next + 1$
33:     end if
34: end while
35: end for
36: end for
Lemma 6. The operations in stage 3 produce a timed automaton $A_3$ that is timed bisimilar to the TA $A_2$ obtained at the end of stage 2.

As mentioned above, in this stage, while merging the constraints, timed bisimilarity is checked and the number of bisimulation checks is bounded by the number of zones in the TA obtained after stage 2. Checking timed bisimilarity is done in EXPTIME [15]. The zone graph is constructed prior to every bisimulation check and the construction is done in EXPTIME. Hence this entire stage runs in EXPTIME.

Stage 4: Active clocks, clock replacement and renaming: Given a location $l$, an iterative method for finding the set of active clocks at $l$, denoted $\text{act}(l)$, is given in [6]. The method has been modified and stated below for the situation where clock assignments (of the form $x := y$, where $x,y \in C$) are disallowed.

Determining active clocks: For a location $l$, let $\text{clk}(l)$ be the set of clocks that appear on the constraints in the outgoing edges of $l$ after the original timed automaton is modified through the previous three stages. Let $\rho : (2^C \times E) \to 2^C$ be a partial function such that for an edge $e = l \xrightarrow{g,a,R} l'$, $\rho(\text{act}(l'), e)$ gives the set of active clocks of $l'$ that are not reset along $e$. For all $l \in L$, $\text{act}(l)$ is the limit of the convergent sequence $\text{act}_0(l) \subseteq \text{act}_1(l) \ldots$ such that $\text{act}_0(l) := \text{clk}(l)$ and $\text{act}_{i+1}(l) := \text{act}_i(l) \cup \bigcup_{e = (l,g,a,R,l') \in E} \rho(\text{act}_i(l'), e)$.

Removing redundant resets: Once we find the active clocks of a location $l$, we remove all resets of clock $x$ on the incoming edges of $l$ if $x \notin \text{act}(l)$.

Partitioning active clocks: Using the DBM representation of the zones, one can determine from the set of active clocks in every location whether some of the clocks in the timed automaton can be expressed in terms of other clocks and thus be removed. Here we investigate the presence of such clocks. Any $x, y \in \text{act}(l)$ belong to an equivalence class iff the same relation (of the form $x - y = k$, for some integer $k$) is maintained between these clocks across all zones of $l$. This is checked using the DBM of the zones of $l$. In this case either $x$ can be replaced by $y + k$ or $y$ can be replaced by $x - k$. Let $\pi_l$ be the partition induced by this equivalence relation.

We note that the size of the largest partition does not give the minimum possible number of clocks required to represent a TA while preserving timed bisimulation. An example is shown in Figure 6(a). Though the automaton in the
figure has two active clocks in every location, a timed bisimilar TA cannot be constructed with only two clocks. Assigning the minimum number of clocks to represent the timed automaton so that timed bisimilarity is preserved can be reduced to the problem of finding the chromatic number of a graph as described below.

Clock graph colouring and clock renaming: A clock graph, $G_D$, for a timed automaton $D$ is constructed in the following way. This graph contains a vertex for each class in the partition $\pi_l$, for every location $l$. Let $V_l$ be the set of vertices corresponding to the classes of $\pi_l$. For each pair of distinct vertices $r_1, r_2 \in V_l$, an edge between $r_1$ and $r_2$ exists denoting that $r_1$ and $r_2$ cannot be assigned the same colour. This is because two classes in the partition $\pi_l$ cannot be represented using the same clock.

Moreover, if at least one clock, say $c$, is common in two classes corresponding to two different locations without any intervening reset of $c$ then only one vertex represents these two classes. For example, in Figure 6, clock $x$ is active in both locations 0 and 1. $\{x\}$ forms a class in the partition of the active clocks for each of locations 0 and 1. Thus we create vertices $x_0$ and $x_1$ corresponding to these two classes. However, since there is no intervening reset of clock $x$ between locations 0 and 1, the vertices $x_0$ and $x_1$ are merged together in the clock graph. Thus after merging some classes into one class, the resultant class can have active clocks corresponding to multiple locations. For a class $T$, let $\text{loc}(T)$ represent the set of locations whose active clocks are members of $T$.

Finding the minimum number of clocks to represent the TA $D$ is thus equivalent to colouring this graph with the minimum number of colours so that no two adjacent vertices have the same colour. The number of colours gives the minimum number of clocks required to represent the TA. If a colour $c$ is assigned to a vertex $r$, then all the clocks in the class corresponding to $r$, say $T$, are renamed $c$. The value of $c$ can be chosen to be equal to some clock in $T$ that is considered to be the representative clock for that class. The constraints involving the rest of these clocks in $T$ are adjusted appropriately and any resets of the clocks, different from the representative clock, present on the incoming edges to $l$ such that $l \in \text{loc}(T)$ are also removed.

For example, suppose vertex $r$ corresponds to a class $T$ having clocks $x$, $y$ and $z$ such that the valuations of the clocks are related as: $x - y = k_1$ and $y - z = k_2$. If colour $c$ is assigned to vertex $r$, then the clocks $x$, $y$ and $z$ in class $T$ are replaced with $c$. If the value of clock $c$ is chosen to be the same as clock $y$, then every occurrence of $x$ in $T$ is replaced with $y + k_1$, while every occurrence of $z$ in $T$ is replaced with $y - k_2$ in the constraints involving $x$ and $z$. The corresponding resets of clocks $x$ and $z$ are also removed.

In Figure 6(a), a TA with three locations is shown. In locations 0, 1 and 2, the sets of active clocks are $\{x, y\}$, $\{w, x\}$ and $\{w, y\}$ respectively. At every location, in this example, each of the active clocks itself makes a class of the partition. Since there are six classes in total, we draw initially six vertices. As mentioned earlier, the vertices $x_0$ and $x_1$ are merged into a single vertex. Similarly $w_1$, $w_2$ and $y_0$, $y_2$ are also merged. We call the resultant vertices $x_{0,1}$, $w_{1,2}$ and $y_{0,2}$. 
Adding the edges as described previously, we get the clock graph which is a
triangle as shown in Figure 6(b). Thus the chromatic number of this graph is
3 which translates to the number of clocks obtained through the operations in
stage 4. Algorithm 3 describes briefly the operations of stage 4. Since this stage

\begin{algorithm}
\begin{algorithmic}[1]
\caption{Algorithm for stage 4}
\label{alg:stage4}
\begin{algorithmic}[1]
\State \textbf{Input:} Timed automaton $A_3$ obtained after stage 3
\State \textbf{Output:} Modified TA $A_4$ after applying stage 4 procedures
\EndState
\For{each location $l$ in $A_3$}
\State Determine $act(l)$
\State Remove resets of clocks on the incoming edges of $l$ that are not active at $l$
\State Determine partition $\pi(l)$ of $act(l)$
\EndFor
\State Construct clock graph and colour it with the minimum possible number of colours.
\State In each class of the partition, replace all clocks in the class with a represent-
\State ative clock of the class and modify the constraints involving each clock that is not
\State a representative clock appropriately. Remove resets of those clocks that are not
\State representative clocks of a class from the incoming edges of the locations the class
\State corresponds to, to obtain the final TA $A_4$.
\end{algorithmic}
\end{algorithm}

\end{algorithm}

consists of finding the active clocks and renaming them, we have the following
lemma.

\textbf{Lemma 7.} The operations in stage 4 produce a timed automaton $A_4$ that is
timed bisimilar to the TA $A_3$ obtained after stage 3.

Further, we can reduce from determining the chromatic number of a graph to
the problem of clock renaming which gives us the following lemma given that
the reduction also holds in the other direction as described above.

\textbf{Lemma 8.} The problem of clock renaming operation on the clock graph as de-
scribed above is NP-complete in the size of the input.

We look at the complexity of the operations in this stage. The sequence of
computation of active clocks converges within $n$ iterations and every iteration
runs in time $O(|E|)$, where there are $n$ locations and $|E|$ edges respectively in
the timed automaton after the first three stages. This is due to the fact that
in iteration $i$, for some location $l$, its active clocks are updated so as to include
the active clocks of the locations $l'$ that are not reset between $l$ and $l'$ and there
exists a path of length at most $i$ between $l$ and $l'$. In each iteration, each edge
is traversed once for updating the set of active clocks of the locations. Thus the
complexity of finding active clocks is $O(n \times |E|)$. Partitioning the active clocks
each of the locations too requires traversing the zone graph and checking the
clock relations from the DBM of the zones. This can be done in time equal to the
order of the size of the zone graph times the size of DBM which is in EXPTIME.

Finally, determining the chromatic number of a graph is possible in time
exponential in the number of the vertices of the graph [108]. Since the number
of locations after the splitting operation in stage 2 is exponential in the number of clocks in $A$, renamining the clocks using the clock graph runs in time doubly exponential in the number of the clocks of the original timed automaton $A$. Thus we have the following theorem.

**Theorem 1.** The stages mentioned above run in 2-EXPTIME.

In the presence of an invariant condition, considering an edge $l \xrightarrow{g,a,R} l'$, a zone $Z'$ of $l'$ is initially created from a zone $Z$ of $l$ such that $Z' = (Z \cap g)_{|R\rightarrow \bar{0}} \cap I(l')$, if $(Z \cap g)_{|R\rightarrow \bar{0}} \cap I(l') \neq \emptyset$, where $I(l')$ is the invariant on location $l'$. The edge can be removed from the timed automaton if for all zones $Z$ of $l$, $(Z \cap g)_{|R\rightarrow \bar{0}} \cap I(l') = \emptyset$. The invariant condition on location $l'$ can be entirely removed if for every incoming edge $l \xrightarrow{g,a,R} l'$ and for each zone $Z$ of $l$, $(Z \cap g)_{|R\rightarrow \bar{0}} \cap I(l') = (Z \cap g)_{|R\rightarrow \bar{0}} \uparrow$ holds. An elementary constraint in the invariant too is removed if it is implied by the rest of the elementary constraints in the invariant. In stage 4, $clk(l)$ becomes the union of the set of clocks appearing in the constraints on the outgoing edges from $l$ and the set of clocks appearing in $I(l)$.

**Proof of Minimality of clocks:** Given a TA $A$, let $A_4$ be the TA obtained from $A$ through the four stages described earlier. We can show that for each location $l$ in $A_4$, for every clock $x \in act(l)$, there exists at least one constraint involving clock $x$ which is indispensable for any TA that preserves timed bisimilarity.

**Lemma 9.** In the TA $A_4$, for each location $l$ and clock $x \in act(l)$, there exists at least one constraint involving $x$ on some outgoing edge of $l$ or on the outgoing edge of another location $l'$ reachable from $l$ such that there is at least one path from $l$ to $l'$ without any intervening reset of $x$. Moreover the TA obtained by removing the constraint is not timed bisimilar to the given timed automaton $A$.

**Proof sketch:** In stage 1, the construction of the zone graph removes the edges and the constraints from the given TA $A$, that never enable a transition, to produce the TA $A_1$. In stage 2, a location $l$ in $A_1$ is split corresponding to every base zone of $l$. This leads to some guards $g$ on the outgoing edges of location $l$ in $A_1$ to be modified while creating $A_2$. In $A_2$, the locations created by splitting location $l$ retain only those elementary constraints of $g$ that affect the behaviour of all the valuations in the base zone of the newly created locations. Thus further splitting of the locations do not eliminate any of the elementary constraints on the guards.

In stage 3, some of the consecutive zones connected by $\varepsilon$ edges, such that from each of them an action $a$ is enabled, are merged. This results into some of the guards being combined as specified in Lines 9 and 10 in Algorithm (iii) or some of the elementary constraints being removed as done in case (ii) and (iii) (Lines 22 to 25), *wherever* such transformations involving merging of guards or removal of elementary constraints preserves timed bisimilarity. Thus after stage 3, none of the elementary constraints appearing in the guards on the edges can further
be removed. Removing any of them changes the behaviour and the resultant TA does not remain timed bisimilar any more.

In stage 4, clocks are renamed and thus the Lemma holds for the TA $A_4$. A more rigorous proof that uses an induction on the structure of the TA $A_4$ is given in the appendix.

**Definition 2. Minimal bisimilar TA**: For a given timed automaton $D$, a minimal bisimilar TA is one that is timed bisimilar to $D$ and has the minimum number of clocks possible.

**Fact 1** For every TA $D$, there exists a minimal bisimilar TA.

One can show that the clocks of a minimal bisimilar TA can replace the clocks of $A_4$ which gives us the following lemma.

**Lemma 10.** The timed automaton $A_4$ has the same number of clocks as a minimal bisimilar TA for $A$.

**Proof sketch**: As stated in the proof of Lemma 9, all the elementary constraints in each of the guards are necessary. In stage 4, further, clock renaming is done in a way so that the constraints can be specified using minimum number of clocks, i.e. no other renaming can lead to a smaller number of clocks than used in $A_4$.

Since all the elementary constraints are necessary to preserve timed bisimilarity, any TA timed bisimilar to $A$ will have constraints that affect the behaviour of the timed automaton in the same way as the constraints of $A_4$. Formally we can show that the clocks of a minimal bisimilar TA can replace the clocks of $A_4$. Replacing the clocks can be considered to be a renaming operation and since after stage 4, any renaming cannot reduce the number of clocks further in $A_4$, the number of clocks in $A_4$ is equal to the number of clocks in the minimal bisimilar TA.

**Corollary 1.** The timed automaton $A_4$ obtained by applying the four stages described above has the minimum possible number of clocks and it is timed bisimilar to the original automaton $A$.

**Theorem 2.** There exists an algorithm to construct a TA $A_4$ that is timed bisimilar to a given TA $A$ such that among all the timed automata that are timed bisimilar to $A$, $A_4$ has the minimum number of clocks. Further the algorithm has a time complexity which is doubly exponential in the number of clocks of $A$.

4 Conclusion

Since model checking of timed automata uses region graph or zone graph whose size increases exponentially with the number of clocks, it is desirable to consider the problem of finding an equivalent automaton with smaller number of clocks. While the problem of checking whether a timed automaton accepting the same timed language but with a smaller number of clocks exists is known to
be undecidable [9], in this paper, we have described an algorithm, which given a timed automaton \( A \), produces another timed automaton \( A_4 \) with the smallest number of clocks that is timed bisimilar to \( A \). It also follows trivially that \( A_4 \) accepts the same timed language as \( A \). If we find such a \( A_4 \) with fewer clocks than \( A \), this also implies the existence of a timed automaton with fewer clocks accepting the same timed language.

For reducing the number of clocks of the timed automaton, we rely on a semantic representation of the timed automaton rather than its syntactic form as in [6]. This helps us to reason about the behaviour of the timed automaton more effectively. Besides, the zone graph we use in our approach is usually much smaller in size than the region graph and its size is independent of the constants used in the timed automaton. Also our method excludes the intermediate step of construction of a characteristic formula as done in [14]. There is an exponential increase in the number of locations while producing the TA with the minimal number of clocks. However, this does not increase the underlying state space of the timed automaton since the splitting of a location \( l \) involves distributing the zones of \( l \) across the locations \( l \) is split into.

References

1. L. Aceto, A. Ingólfsdóttir, K.G. Larsen, and J. Srba. Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, 2007.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. Inf. Comput., 104:2–34, May 1993.
3. R. Alur and D.L. Dill. A theory of timed automata. Theoretical Computer Science, 126:183–235, 1994.
4. B. Berthomieu and M. Menasche. An enumerative approach for analyzing time petri nets. In IFIP Congress, pages 41–46, 1983.
5. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In Proceedings of TACAS, pages 313–329. Springer-Verlag, 1998.
6. C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In Proc. RTSS’96, 73-81, IEEE, pages 73–81. IEEE Computer Society Press, 1996.
7. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Automatic Verification Methods for Finite State Systems, pages 197–212, 1989.
8. D. Eppstein. Small maximal independent sets and faster exact graph coloring. J. Graph Algorithms Appl., 7:131–140, 2003.
9. O. Finkel. Undecidable problems about timed automata. In Proceedings of FORMATS, pages 187–199. Springer-Verlag, 2006.
10. E. Fleury G. Behrmann, P. Bouyer and K. G. Larsen. Static guard analysis in timed automata verification. In Proceedings of TACAS, pages 254–270. Springer-Verlag, 2003.
11. K. G. Larsen G. Behrmann, P. Bouyer and R. Pelanek. Lower and upper bounds in zone-based abstractions of timed automata. International Journal on Software Tools for Technology Transfer, 8:204–215, 2006.
12. S. Guha, S. N. Krishna, C. Narayan, and S. Arun-Kumar. A unifying approach to decide relations for timed automata and their game characterization. In Proceedings of Express/SOS, EPTCS, volume 120, pages 47–62, 2013.
13. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:394–406, 1992.

14. F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic – and back. In *Proceedings of MFCS*, pages 529–539, 1995.

15. F. Laroussinie and Ph. Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In *Proceedings of FoSSaCS*, pages 192–207. Springer-Verlag, 2000.

16. E. L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5:66–67, 1976.

17. S. Tripakis. Folk theorems on the determinization and minimization of timed automata. *Inf. Process. Lett.*, 99:222–226, September 2006.

18. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18:25–68, 2001.

19. C. Weise and D. Lenzkes. Efficient scaling-invariant checking of timed bisimulation. In *Proceedings of STACS*, volume 1200, pages 177–188. Springer, Berlin, 1997.
Appendix

A Proofs of Lemmas

A.1 Proof of Lemma 1

Proof. We show the proof for two clocks. The same argument holds for arbitrary number of clocks. Consider a $|C|$ dimensional zone $Z$. Since $|C| = 2$, a zone $Z$ that is bounded above is of the form $k_{xy_1} < x - y < k_{xy_2}$, $k_{x_1} < x < k_{x_2}$ and $k_{y_1} < y < k_{y_2}$, where $x$ and $y$ are the two clocks, $\prec \epsilon \{<,\leq\}$. Let $Z$ be bounded above by two hyperplanes $x = k_{x_2}$ and $y = k_{y_2}$. As can be seen from Figure 7 there are two zones $Z_1$ and $Z_2$ such that $Z_1$ is defined by the inequations $k_{xy_1} < x - y < k_{xy_2}$, $k_{x_1} < x < k_{x_2}$ and $k_{y_1} < y$. Making $Z$ pre-stable will divide it into two parts:

- $Z'$ defined by the inequations $k_{xy_1} < x - y < k_{xy_2}$, $k_{x_1} < x < k_{x_2}$ and $k_{y_1} < y$
- $\tilde{Z}$ define by the inequations $k_{x_2} - k_{y_2} < x - y < k_{xy_2}$, $k_{x_1} < x$ and $y < k_{y_2}$

$Z'$ is bounded fully from above by the hyperplane $k_{x_2}$ while $\tilde{Z}$ is bounded fully from above by the hyperplane $k_{y_2}$. Note that the inequations defining the zones $Z'$ and $\tilde{Z}$ may vary depending on the relation among the various constants $k_{xy_1}, k_{xy_2}, k_{x_1}, k_{x_2}, k_{y_1}$ and $k_{y_2}$. In all cases, pre-stability will ensure that there exists a single hyperplane that fully bounds a zone from above.

\[ \begin{align*}
&x - y = k_{xy_2} \\
&x - y = k_{xy_1} \\
&k_{y_2} \\
&k_{y_1} \\
&k_{x_2} \\
&k_{x_1}
\end{align*} \]

Fig. 7. For every zone, there exists a hyperplane that bounds it fully from above in a pre-stable zone graph

A.2 Proof of Lemma 2

Proof. During the construction of the zone graph, an edge is entirely removed if the corresponding transition is never enabled. Removing those edges corresponding to which no transition takes place does not affect the behaviour of the TA. Hence after stage 1, the resultant TA remains timed bisimilar to the original TA.
A.3 Proof of Lemma 4

Proof. In the second stage, a location \( l \) is split such that corresponding to every base zone of \( l \), new locations \( l_1, \ldots, l_n \) are created. Consider valuations \( v_1, \ldots, v_n \) in \( l_1, \ldots, l_n \) respectively. Since the zones of \( l \) are distributed over locations \( l_1, \ldots, l_n \), before the split, all the valuations \( v_1, \ldots, v_n \) are reachable in \( l \). For all \( v_i, 1 \leq i \leq n \), \((l, v_i)\) (before \( l \) is split) is timed bisimilar to \((l_i, v_i)\) after the split. Hence the timed automaton after stage 2 remains timed bisimilar to the original TA \( A \).

A.4 Proof of Lemma 6

Proof. In the third stage, a zone of some location \( l \) may be merged with another zone if their union is convex and this process may be repeated a finite number of times. This operation is done if the resultant zone graph is still timed bisimilar to the original zone graph, i.e. the initial states of the two zone graphs are timed bisimilar. The changes in the zone graph are reflected in the timed automaton in the following way. If from a location \( l_i \) there are edges of the form \( l_i \xrightarrow{a,g_1,R_1} l_{i1}, \ l_i \xrightarrow{a,g_2,R_2} l_{i2}, \ldots, l_i \xrightarrow{a,g_n,R_n} l_{in} \) in the timed automaton \( A \) such that they are labelled with the same action \( a \), then some of the constraints may be replaced with a constraint equivalent to \( g_1 \cup g_2 \cup \ldots \) if the resultant timed automaton is timed bisimilar to the original timed automaton. Hence the operations in this stage create a timed bisimilar TA.

A.5 Proof of Lemma 7

Proof. In stage 4, clocks are renamed. Also in every zone in each location, the active clocks are identified and partitioned such that all clocks belonging to a class in the partition are represented with a single clock, say \( c \). If there is a clock in the partition such that \( x \sim c = m \), where \( m \) is an integer, then the constraint of the form \( x \sim k \) is replaced with \( c \sim k - m \). The clock renaming and replacing the integer constants in the way mentioned above do not change any bisimulation property of a timed state. Hence the operations in stage 4 preserve the timed bisimulation property of the original TA \( A \).

A.6 Proof of Lemma 8

Proof. Consider any arbitrary graph \( G \). We show that there exists a TA whose clock graph is the same as \( G \). Thus coloring the vertices of \( G \) is reduced to assigning minimum number of clocks to the clock graph.

The reduction is the following: Let the vertices of \( G \) be \( x_1, x_2, \ldots, x_n \). Corresponding to every edge \( e \) between \( x_i \) and \( x_j \), create a location \( l \) of the timed automaton whose active clocks are \( x_i \) and \( x_j \). For the reduction to be correct, we also need to show that \( x_i \) cannot be replaced with \( x_j \) in location \( l \). Let the locations of the timed automaton be numbered \( l_1, l_2, \ldots, l_m \) where \( m \) is the number of edges in \( G \).
Now note that every location \( l_1, \ldots, l_m \) has exactly two active clocks. We create three additional locations \( l_0, l_{m+1} \) and \( l_{m+2} \) and draw edges from \( l_0 \) to \( l_1 \), \( l_m \) to \( l_{m+1} \) and from \( l_{m+1} \) to \( l_{m+2} \). If the active clocks of \( l_1 \) are \( x_i \) and \( x_j \), then we draw an edge from \( l_0 \) to \( l_1 \) with the constraint \( x_i \leq 1 \) and reset \( x_j \). The other edges in the timed automaton are drawn in the following manner.

An edge is drawn between two locations \( l_u \) and \( l_v \) if \( u < v \) and \( \text{act}(l_u) \cap \text{act}(l_v) \neq \emptyset \). Note that there can be at most one active clock common between two distinct locations \( l_u \) and \( l_v \). Also note that since we add edges from \( l_u \) to \( l_v \) such that \( u < v \leq m \), the timed automaton thus produced does not have any cycle.

Now we describe the constraints and the resets on the edges of the timed automaton. Consider two locations \( l_u \) and \( l_v \) connected by an edge from \( l_u \) to \( l_v \) and let the active clocks of \( l_u \) be \( x_i \) and \( x_j \) while the active clocks of \( l_v \) be \( x_j \) and \( x_i \). On this edge we add a constraint \( x_i \leq k_i \), where \( k_i \in \mathbb{N} \) and reset \( x_i \). The value of \( k_i \) is chosen in the following way. Suppose clock \( x_i \) was reset on an incoming edge of \( l_u \) such that there is a path from \( l_w \) to \( l_u \) without any reset of \( x_i \). We define the weight of clock \( x_i \) on the path from \( l_w \) to \( l_u \), denoted \( wt(x_i)_{w,u} \), to be the sum of the integers used in the constraints on the path from \( l_w \) to \( l_u \). Now we assign to \( k_i \) the value \( 1 + \max(wt(x_i)_{w,u}) \), i.e., one added to the maximum weight of clock \( x_i \) at location \( l_u \) computed over all incoming paths. This value of \( k_i \) ensures that the locations \( l_1, \ldots, l_m \) are not further split.

If the active clocks at \( l_m \) be \( x_r \) and \( x_s \), then we add an edge from \( x_m \) to \( x_{m+1} \) with the constraint \( x_r \leq k_r \) wlog and an edge from \( x_{m+1} \) to \( x_{m+2} \) with the constraint \( x_s \leq k_s \). \( k_r \) and \( k_s \) are calculated in the same way as \( k_i \) as described above.

Clearly, the timed automaton thus constructed has a clock graph that is the same as \( G \).

![Fig. 8.](image)

**Fig. 8.** (a) a graph \( G \) (b) TA with clock graph same as \( G \)

Figure [8](a) shows a graph \( G \) and [8]b) shows the TA constructed following the procedure described above. Note that there are 6 edges in \( G \) corresponding to which there are 6 locations in the TA which are \( l_1 \) to \( l_6 \). There are three additional locations \( l_0, l_7 \) and \( l_8 \). Corresponding to every location in \( l_1 \) to \( l_6 \), its active clocks are written in blue inside parentheses. For example, the active
A.7 Proof of Lemma 9

Proof. We prove this lemma by induction on the structure of the timed automaton. We prove that the statement of the lemma holds for a location \( l \) if it holds for all the locations reachable from \( l \) through a single transition. The proof proceeds by considering the exhaustive set of cases shown in Figure 9. In the rest of the proof, wlog, we consider the label of an edge as the action \( a \) whenever not specified otherwise.

Let us suppose that we have edges in the timed automaton \( A_4 \) from location \( l \) to locations \( l_1, \ldots, l_m \). By induction hypothesis, the lemma holds for \( l_1, \ldots, l_m \).

Note that act\( (l) = clk(l) \cup \bigcup_{i=1}^m (act(l_i) \setminus R_i) \). Consider a clock \( c_0 \in act(l) \). Also note that a clock \( c_0 \) will be replaced by some clock in \( \bigcup_{i=1}^m act(l_i) \) if \( c_0 \notin \bigcup_{i=1}^m act(l_i) \) to ensure minimality while coloring the clock graph. We show that if \( c_0 \in act(l) \), then there is a constraint involving \( c_0 \) that cannot be removed while preserving timed bisimilarity.

If there exists some location \( l_i, 1 \leq i \leq m \), such that on the edge from \( l \) to \( l_i \), \( c_0 \) has not been reset, then by IH, \( c_0 \in act(l_i) \) and thus the lemma holds at \( l_i \) too. Consider two edges \( l \xrightarrow{g_i,a,R_i} l_1 \) and \( l \xrightarrow{g_j,a,R_j} l_j \) such that with reference to Algorithm 2, \( g_{\text{curr}} \) is same as \( g_i \) and \( g_{\text{next}} \) is same as \( g_j \) while \( g'_{\text{curr}} \) is the same as \( g \) appearing below. Let the hyperplane bounding End\( (g_i)(l,a) \) be \( c_0 = k \) that is induced by the elementary constraint \( c_0 \sim k \) of UB\( (\text{End}(g_i)(l,a)) \). If Ran\( (g_i)(l,a) \cup \text{Ran}(g_j)(l,a) \) is not convex, then the constraints \( g_i \) and \( g_j \) cannot be merged so that the resulting TA remains timed bisimilar.

For the following cases, we consider Ran\( (g_i)(l,a) \cup \text{Ran}(g_j)(l,a) \) to be convex but Ran\( (g_i)(l,a) \) and Ran\( (g_j)(l,a) \) do not overlap. We show that if \( g_i \) and \( g_j \) could not be merged to produce a constraint \( g \) as shown in Line 9 of Algorithm 2, then there exists a clock constraint involving \( c_0 \) that cannot be removed while preserving timed bisimilarity. The case where Ran\( (g_i)(l,a) \) and Ran\( (g_j)(l,a) \) overlap can also be reasoned about similarly.

Suppose \( c_0 \) is reset on all of the edges from \( l \) to \( l_i \) where it appears as part of some constraint, \( 1 \leq i \leq m \). Now since the resets of \( c_0 \) exist on the edges from \( l \) to \( l_i \) and from \( l \) to \( l_j \), \( c_0 \in act(l_i) \) and \( c_0 \in act(l_j) \). Otherwise, the resets on the edges would have been removed in stage 4. Since \( c_0 \) has been reset on edges to both \( l_i \) and \( l_j \), the clock constraints involving \( c_0 \) from \( l_i \) and from \( l_j \) have to be the same for replacing \( g_i \) and \( g_j \) with \( g \). Since it was not replaced in the TA \( A_4 \), it implies that the constraint \( c_0 \sim k \) cannot be removed while preserving timed bisimilarity.
The following cases include those where \( c_0 \) is reset on all of the edges from \( l \) to \( l_i \), \( 1 \leq i \leq m \) and the constraints in the paths from \( l_j \) and \( l_j \neq l_j \) involving \( c_0 \) are the same for that part of the path where there is no intervening reset of \( c_0 \).

Now wlog we consider a clock \( c_1 \in \text{act}(l_j) \) and \( c_1 \) different from \( c_0 \) such that \( c_1 \) has not been reset on the edge from \( l \) to \( l_j \). This implies that \( c_1 \in \text{act}(l) \) as well. If \( c_1 \notin \text{act}(l_i) \) and if \( g_i \) is replaced with \( g_i \), then the behaviour of the states in zone \( Z_{l_i[R_i \leftarrow 0]} \) of \( l_i \) will depend on an additional constraint involving clock \( c_1 \) which was not the case in the original automaton. Thus \( g_i \) and \( g_j \) cannot be replaced with \( g \) which implies that the constraint \( c_0 \sim k \) cannot be removed while preserving timed bisimilarity.

Now if \( c_1 \in \text{act}(l_i) \) and \( c_1 \in \text{act}(l_j) \), then the upper bound of the constraint involving clock \( c_1 \) on both the edges from \( l_i \) to \( l_j \) have to be the same for replacing \( g_i \) and \( g_j \) with \( g \). The upper bound is \( k \), if the constraint is of the form \( c_1 < k \) or \( c_1 \leq k, k \in \mathbb{N} \). Otherwise, the states in the zone \( Z_{l_i[R_i \leftarrow 0]} \) will behave differently if \( g_i \) is replaced with \( g \) in which case the constraint \( c_0 \sim k \) cannot be removed while preserving timed bisimilarity.
Now consider the case where in the constraints involving $c_1$ on the edges reachable from $l_i$ and $l_j$ without any intervening reset of $c_1$, the upper bounds of $c_1$ are the same but the lower bounds of $c_1$ are not the same. Now let the lower bound of $c_1$ on the constraint on the edge reachable from $l_j$ be $b_{low}$, i.e. the constraint is of the form $c_1 > b_{low}$ or $c_1 \geq b_{low}$, $b_{low} \in \mathbb{N}$. We note that in location $l_j$, if every reachable state $(l_j, v'_j)$ be such that $v'_j(c_1) > b_{low}$, then the constraint is eliminated in stage 2 since all reachable valuations in $l_j$ satisfy the constraint implicitly. If there exists a state $(l_j, v_j)$ that is reachable from some state $(l, v)$ such that $b_{low} > v_j(c_1)$, then we argue that $g_i$ and $g_j$ cannot be merged. Consider a state $(l_i, \hat{v}_i)$ reachable from $(l, \hat{v})$ such that $v_i(c) < b_{low}$. Such a state $(l_i, \hat{v}_i)$ exists since zone $Z_j$ of $l$ is an immediate delay successor of zone $Z_i$. We have $\hat{v}_i = \hat{v}_{|R_i \leftarrow \overline{\bar{F}}_i|}$. If $g_i$ and $g_j$ are merged, then both $g_i$ and $g_j$ are replaced with $g$ and we have a state $(l_j, \hat{v}_j)$ reachable from $(l, \hat{v})$ such that $\hat{v}_j = \hat{v}_{|R_j \leftarrow \overline{\bar{F}}_j|}$. Thus with the merging of $g_i$ and $g_j$, we have a transition $(l, \hat{v}) \xrightarrow{a} (l_j, \hat{v}_j)$ such that the behaviour of $(l_j, \hat{v}_j)$ is affected by a constraint of the form $c_1 \sim b_{low}$ that was not the case in the original timed automaton. Since the merging of $g_i$ and $g_j$ does not preserve timed bisimulation, we cannot merge them. This further implies that the constraint $c_0 \sim k$ cannot be removed while preserving timed bisimilarity.

Now consider the case where the lower bounds of $c_1$ are not the same and the lower bound on the constraint on the edge reachable from $l_i$ be greater than $v(c_1)$, where $v$ is some valuation in the zone $Z_{l_j}[l_j, v] \rightarrow 0$ of location $l_j$. Arguing analogously to the previous case, we can show that $g_i$ and $g_j$ cannot be merged, i.e. $g_i$ and $g_j$ cannot be replaced with $g$ while preserving timed bisimilarity. This further implies that the constraint $c_0 \sim k$ cannot be removed while preserving timed bisimilarity.

Otherwise, if the lower bounds on the constraints involving $c_1$ on the edges reachable from both $l_i$ and $l_j$ are not the same and the lower bounds on these constraints involving $c_1$ on the edges reachable from both $l_i$ and $l_j$ are less than or equal to $v(c_1)$ for every valuation $v$ reachable in $l_j$, then the pre-stabilization operation replaces the constraints from $l$ to $l_i$ and from $l$ to $l_j$ involving $c_0$ with constraints involving $c_1$ and other active clocks in $l$ as shown in Figure 10. In the figure, $x$ represents $c_0$ while $y$ represents $c_1$. Thus this particular case cannot occur in the automaton $A_4$. Also $l_i$ and $l_j$ as described in the previous cases can be considered to be $l_2$ and $l_3$ in the TA on the left. Note that the value of clock $y$ in every state reachable from $l_3$ is greater than 3 while the lower bounds on the constraints on $y$ reachable from $l_2$ and $l_3$ are 3 and 0 respectively.

Hence the lemma holds for location $l$. $\square$

Towards defining the clock mapping from $D$ to $A_4$, we first define a corner point trace (cp-trace) in the zone graph.

**Definition 3.** A corner point of a zone in the zone graph is a state where corresponding to each clock $x$, the hyperplanes of the form $x = c$ that define the zone intersect each other. Considering $M$ to be the largest constant appearing in the constraints or the location invariants of the timed automaton, each coordinate of the corner point is of the form $n, n + \delta$ or $n - \delta$ where $n \in \{0, 1, \ldots, M\}$.
and δ is any infinitesimally small value. In a zone \( Z \), there can be two kinds of corner points, an entry corner point and an exit corner point. For an entry corner point, there is a unique exit corner point, that can be reached from the entry corner point by performing a delay \( d \) such that any delay more than \( d \) will lead to a state that is in the immediate delay successor zone of \( Z \).

For a zone, it is possible to have a pair of such entry and exit corner points that are not distinct, for example, in Figure 1, in zone \( Z_4 \), the corner point \((x = 5 + \delta, y = 7)\) is both an entry as well as an exit corner point, while the corner point \((x = 5 + \delta, y = 5 + \delta)\) is another entry corner point for the same zone \( Z_4 \) and the corresponding exit corner point is \((x = 7, y = 7)\). If the entry and the exit corner points are distinct, then a non-zero or a non-infinitesimally small delay can be performed from the entry corner point to reach the corresponding exit corner point whereas an infinitesimally small delay from the exit corner point causes it to evolve into an entry corner point of the immediate delay successor zone. A zone that is not bounded above does not have any exit corner point.

For an entry corner point, the corresponding exit corner point is the next delay corner point whereas for an exit corner point, the entry corner point in the immediate delay successor zone is the next delay corner point.

**Definition 4.** A corner point trace is a timed trace ending at a corner point through delays and visible actions such that each delay takes the trace from one corner point to the next delay corner point.

A.8 Proof of Lemma 10

Proof. We prove the lemma by showing that the clocks of \( A_4 \) can be replaced with the clocks of a minimal TA that uses say \(|C_D|\) clocks. Let \( A_4 \) has \(|C_{A_4}|\)
clocks. We know that if $D$ is a minimal TA, we have $|C_D| \leq |C_{A_4}|$. Since any other renaming cannot reduce the number of clocks in $A_4$ and if the clocks of $A_4$ can be replaced with the clocks of $D$, we have $|C_{A_4}| \leq |C_D|$, thus giving $|C_{A_4}| = |C_D|$.

We consider a minimal TA $D_1$ timed bisimilar to the given TA $A$ and apply the operations in the four stages on it to produce a TA $D$. The application of these stages on the minimal TA does not add to the number of clocks and since $D_1$ is already minimal, $D$ has the same number of clocks as $D_1$. Due to the transformation from $D_1$ to $D$, it is ensured that in $D$’s zone graph, for each zone in every location, there exists a hyperplane that fully bounds it. Similarly in the zone graph of $A_4$ too, for each zone of every location, there exists a hyperplane that fully bounds it.

We use the zone graphs of $A_4$ and $D$ for mapping the clocks of $A_4$ to the clocks of $D$. From Lemma 9 for a location $l_{A_4}$ of $A_4$, corresponding to every clock $x \in act(l_{A_4})$, there is a hyperplane of the form $x = k$ corresponding to a constraint $x \sim k$ which cannot be removed so that the modified TA remains timed bisimilar. Since $A_4$ and $D$ are timed bisimilar, there is a corresponding hyperplane, say $y = k'$, induced by a constraint $y \sim k'$ which too cannot be removed from $D$ while preserving timed bisimulation.

The cp-trace we consider below are the ones that start from the initial state of the zone graph and the delays in the trace are such that it moves from an entry corner point to the exit corner point of the same zone and from an exit corner point to an entry corner point of the immediate delay successor zone. We consider multiple finite cp-traces in the zone graph such that all the corner points in the zone graph are traversed by some cp-trace at least once.

Now we consider a part of a cp-trace in the zone graph of $A_4$ from the initial state to an exit corner point $(l_{A_4}, v)$ of zone $Z_{A_4}$. Suppose the hyperplane $x = k$ bounds the zone $Z_{A_4}$ and the constraint $x \sim k$ inducing the hyperplane be such that it cannot be removed while preserving timed bisimulation. Since $A_4$ and $D$ are timed bisimilar, there exists a state $(l_D, v')$ in the zone graph of $D$ that is timed bisimilar to $(l_{A_4}, v)$ and $(l_D, v')$ is an exit corner point of some zone $Z_D$. Continuing this way, we consider all the different exit corner points in $Z_{A_4}$ and the corresponding bisimilar corner points in $Z_D$ to find out the clock involved in the hyperplane bounding the zone $Z_{A_4}$, that is $x$ and the clock in the corresponding hyperplane bounding $Z_D$ that is $y$ and thus we can replace clock $x$ in $A_4$ with clock $y$ of $D$. This is repeated until all the occurrences of all the clocks in $A_4$ are replaced with clocks in $D$.

As mentioned above, here $x \in act(l_{A_4})$ and $y \in act(l_D)$. Since an active clock of a location cannot be replaced with another active clock of the same location, every active clock in every location in $A_4$, can be mapped uniquely to a clock in $D$ which can replace the clock in $A_4$. Continuing this way, the clocks associated to the constraints in all the edges in $A_4$ are replaced.

Now since the clocks of $A_4$ have been replaced with the clocks of $D$, we have $|C_{A_4}| \leq |C_D|$. On the other hand, since $D$ is a minimal bisimilar TA, we have $|C_D| \leq |C_{A_4}|$. Thus we have $|C_{A_4}| = |C_D|$ and hence $A_4$ and $D$ have the same
number of clocks. Stated otherwise, $A_4$ has the least number of clocks among the TAs that are timed bisimilar to the original timed automaton $A$. □