Reducing Clocks in Timed Automata while Preserving Bisimulation

Shibashis Guha*, Chinmay Narayan, and S. Arun-Kumar

Indian Institute of Technology Delhi, New Delhi, India
{shibashis,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 [3] 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) [3,2]. 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 [12]. 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 [9], 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*

* The research of Shibashis Guha was supported by Microsoft Corporation and Microsoft Research India under the Microsoft Research India PhD Fellowship Award.

P. Baldan and D. Gorla (Eds.): CONCUR 2014, LNCS 8704, pp. 527–543, 2014.

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 [9] does not capture such implication. Also by considering constraints on more than one outgoing edge from a location, e.g. $l_0 \xrightarrow{a,x\leq 3,\emptyset} l_1$ and $l_0 \xrightarrow{a,x>3,\emptyset} l_2$ collectively, we may sometimes eliminate the constraints that may remove some clock. This too has not been accounted for by the algorithm of [9].

In [19], 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 [12].

Another result appearing in [16] 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 [16]. The method in [16] 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. The problem that we solve in the current paper was in fact left open in [16].

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.

2 Timed Automata

Formally, a *timed automaton* (TA) [3] is defined as a tuple $A = (L, Act, l_0, C, E)$ where L is a finite set of locations, Act is a finite set of visible actions, $l_0 \in L$ is the initial location, C is a finite set of clocks and $E \subseteq L \times \mathcal{B}(C) \times Act \times 2^C \times L$ is a finite set of *edges*. The set of constraints or guards on the edges, denoted $\mathcal{B}(C)$, is given by the grammar $g ::= \ x \bowtie k \mid g \wedge g$, where $k \in \mathbb{N}$ and $x \in C$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. 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) [1]. The timed labelled transition system $T(A)$ generated by A is defined as $T(A) = (Q, Lab, Q_0, \{\xrightarrow{\alpha} \mid \alpha \in Lab\})$, where $Q =$

$\{(l,v) \mid l \in L, v \in {\mathbb{R}_{\geq 0}}^{|C|}\}$ 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 in $|C|$ dimensional real space where each clock is mapped to a unique dimension in this space; $Lab = Act \cup \mathbb{R}_{\geq 0}$ 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 $e = (l,g,a,R,l') \in E$, $v \models g, v' = v_{[R \leftarrow \overline{0}]}$, where $v_{[R \leftarrow \overline{0}]}$ 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 $C \backslash R$ remain unchanged.

For simplicity, we do not consider annotating locations with clock constraints (known as *invariant conditions* [15]). 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 $d_1 a_1 d_2 a_2 \ldots d_n a_n$ is called a timed trace iff there is a sequence of transitions $p_0 \xrightarrow{d_1} p_1 \xrightarrow{a_1} p_1' \xrightarrow{d_2} p_2 \xrightarrow{a_2} p_2' \cdots \xrightarrow{d_n} p_n \xrightarrow{a_n} p'$ in $T(A)$, with p_0 being the initial state of the timed automaton.*
2. ***Zone**: A zone Z is a set of valuations $\{v \in {\mathbb{R}_{\geq 0}}^{|C|} \mid v \models \beta\}$, where β is of the form $\beta ::= \; x \bowtie k \mid x - y \bowtie k \mid \beta \wedge \beta$, k is an integer, $x, y \in C$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. $Z\uparrow$ denotes the future of the zone Z. $Z\uparrow = \{v + d \mid v \in Z, d \geq 0\}$ is the set of all valuations reachable from Z by time elapse. A zone is a convex set of clock valuations.*
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 $Z_1 \subseteq preds(Z_2)$ or $Z_1 \cap preds(Z_2) = \emptyset$ where $preds(Z) \stackrel{def}{=} \{v \in {\mathbb{R}_{\geq 0}}^{|C|} \mid \exists v' \in Z, \exists l, l' \in L, \exists \alpha \in Lab$ such that $(l,v) \xrightarrow{\alpha} (l',v')\}$. Here $l = l'$ if α is a delay action.*
4. ***Canonical decomposition**: Let $g = \bigwedge_{i=1}^{n} \gamma_i \in \mathcal{B}(C)$, where each γ_i is an* elementary constraint *of the form $x_i \bowtie k_i$, 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 $Z = x \geq 0 \wedge y \geq 0$ and the guard $x \leq 2 \wedge y > 1$. Z is split with respect to $x \leq 2$, and then with respect to $y > 1$, hence into four zones : $x \leq 2 \wedge y \leq 1$, $x > 2 \wedge y \leq 1$, $x \leq 2 \wedge y > 1$ and $x > 2 \wedge y > 1$. An elementary constraint $x_i \bowtie k_i$ induces the hyperplane $x_i = k_i$ in a zone graph of the timed automaton.*

5. ***Zone graph****: Given a timed automaton $A = (L, Act, l_0, C, E)$, a* zone graph *$\mathcal{G}_A$ of A is a transition system $(S, s_0, Lep, \rightarrow)$, that is a finite representation of $T(A)$. Here $Lep = Act \cup \{\varepsilon\}$. $S \subseteq L \times \mathcal{Z}$ is the set of nodes of $\mathcal{G}_A$, $\mathcal{Z}$ being the set of zones. 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 $l_i \xrightarrow{g,a,R} l_j$ in A and $Z' \subseteq ([Z \cap g]_{R \leftarrow \overline{0}})\uparrow$ obtained after canonical decomposition of $([Z \cap g]_{R \leftarrow \overline{0}})$. 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 $(l_i, v) \xrightarrow{d} (l_i, v + d)$ 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'. The relation ε is reflexive and transitive and so is the* delay successor *relation. We denote the set of delay successor zones of Z with $ds(Z)$. A zone $Z' \neq Z$ is called the* immediate delay successor *of a zone Z iff $Z' \in ds(Z)$ and $\forall Z'' \in ds(Z) : Z'' \neq Z$ and $Z'' \neq Z'$, $Z'' \in ds(Z')$. We call a zone Z corresponding to a location to be a* base zone *if Z does not have a delay predecessor other than itself.*
6. *A hyperplane $x = k$ is said to* bound *a zone Z from above if $\exists v \in Z\, [\, \forall d \in \mathbb{R}_{\geq 0}\, [\, (v + d)(x) \succ k \iff (v + d)(x) \notin Z\,]\,]$, where $\succ \in \{>, \geq\}$. 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 $\forall v \in Z\, [\, \forall d \in \mathbb{R}_{\geq 0}\, [\, (v + d)(x) \succ k \iff (v + d)(x) \notin Z\,]\,]$. Analogously, we can also say that a hyperplane $x = k$ bounds a zone from below if $\exists v \in Z\, [\, \forall d \in \mathbb{R}_{\geq 0}\, [\, (v - d)(x) \prec k \iff (v - d)(x) \notin Z]]$, where $\prec \in \{<, \leq\}$. We can also define a hyperplane bounding a zone fully from below in a similar manner.*
   When not specified otherwise, in this paper, a hyperplane bounding a zone implies that it bounds the zone from above. A zone Z is 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 [4]. Several kinds of abstractions have been proposed in the literature [8,4,5] to make the zone graph finite. We use *location dependent maximal constants* abstraction [4] 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 if the zone Z in any node (l, Z) in the zone graph is bounded above, then it is bounded fully from above by a hyperplane $x = h$, where $x \in C$ and $h \in \mathbb{N}$.*

Some approaches for preserving convexity and implementing pre-stability have been discussed in [20]. 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

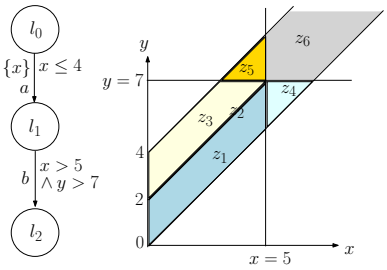

Fig. 1. A timed automaton and the zones for location l_1

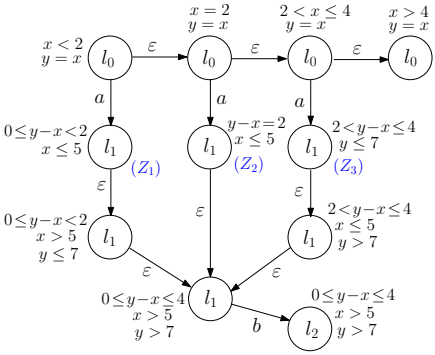

Fig. 2. Zone Graph for the TA in Figure 1

figure. In this paper, from now on, unless stated otherwise, *zone graph* will refer to this form of the pre-stable zone graph that is described above. An algorithmic procedure for the construction of the zone graph is given in [13].

A relation $\mathcal{R} \subseteq Q \times Q$ is a *timed simulation* relation if the following conditions hold for any two timed states $(p, q) \in \mathcal{R}$.
$\forall\ a \in Act,\ p \xrightarrow{a} p' \implies \exists q' : q \xrightarrow{a} q'$ and $(p', q') \in \mathcal{R}$ and
$\forall d \in \mathbb{R}_{\ge 0},\ p \xrightarrow{d} p' \implies \exists q' : q \xrightarrow{d} q'$ and $(p', q') \in \mathcal{R}$.
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. Using product construction on region graphs, timed bisimilarity for timed automata was shown to be decidable in EXPTIME [7].

3 Clock Reduction

Unlike the method described in [9], 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) [6,10] 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 M where an extra variable x_0 is introduced such that the value of x_0 is always 0. An element M_{ij} is of the form $(m_{ij}, \prec)$ where $\prec \in \{<, \le\}$ 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 TA that are never enabled. Such edges and constraints may be removed. (Stage 1)
- Splitting some locations may lead to a reduction in the number of the clocks. (Stage 2)

- At some location, some clocks whose values may be expressed in terms of other clock values, may be removed. (Stage 2)
- Two or more constraints on edges outgoing from a location 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 pre-stable zone graph of the given timed automaton, as described in Section 2. 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 $l_i \xrightarrow{g,a,R} l_j$ 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 $l_i \xrightarrow{g,a,R} l_j$ 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 the example of the timed automaton in 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 of the 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 the clocks can be reused while describing the behaviours from each of 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.

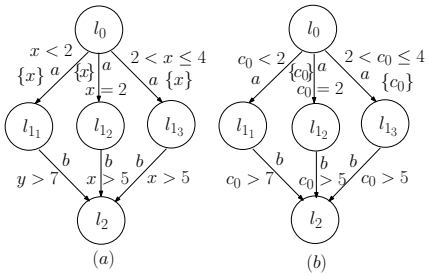

Fig. 3. Splitting locations of the TA in Figure 1

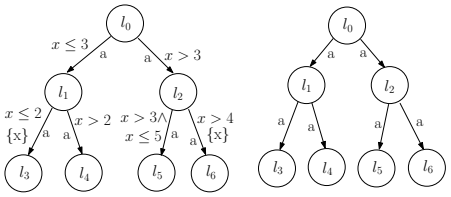

Fig. 4. The two timed automata are timed bisimilar

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: $A_2 := A_1$ ▷ A_1 is the TA obtained from A after the first stage
2: **for each** location l_i in A_1 **do** ▷ i is the index of the location
3: Split l_i into m locations $l_{i_1}, \ldots, l_{i_m}$ in A_2 ▷ Let m be the number of base zones of l_i
4: Remove location l_i and all incoming and outgoing edges to and from l_i from A_2
5: **for each** j in 1 to m **do**
6: **for each** incoming edge $l_r \xrightarrow{a,g_r,R_r} l_i$ in A_1 **do**
7: ▷ Split the constraints on the incoming edges to l_i for the newly created locations
8: $Z'_{i_j} := Z_{i_j} \uparrow \cap \mathbb{R}^{|C|}_{\geq 0\,[R_r \leftarrow \overline{0}]}$ ▷ Let Z_{i_j} be the base zone corresponding to l_{i_j}
9: ▷ Let 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: $g_r \wedge free(Z'_{i_j}, R_r) \wedge Z_{r_j} \Rightarrow g_{r_j}$ and g_{r_j} has a subset of the clocks used in g_r.
12: 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: **if** $Z_{i_j} \uparrow \cap g_i = \emptyset$ **then**
16: Do not create this edge from l_{i_j} to l_r in A_2 since it is never going to be enabled for any valuation of Z_{i_j};
17: **else**
18: Let S_r be the set of elementary constraints in g_i
19: **loop**
20: **if** $\exists s' \in S_r,\ s.t.\ Z_{i_j} \uparrow \wedge (\bigwedge_{s \in S_r \setminus \{s'\}} s) \Rightarrow Z_{i_j} \uparrow \wedge\, s'$ **then**
21: $S_r = S_r \setminus \{s'\}$
22: **else**
23: Create an edge $l_{i_j} \xrightarrow{a,g_{i'},R_i} l_r$ in A_2, where $g_{i'} = \bigwedge_{s \in S_r}$
24: **Break**;
25: **end if**
26: **end loop**
27: **end if**
28: **end for**
29: **end for**
30: **end for**

Note that a zone can be considered to be a set of constraints defining it. Similarly a guard can also be considered in terms of the valuations satisfying it. Input of this algorithm, A_1 is the TA obtained after stage 1. If there are m base zones in $\mathcal{G}_{A_1}$ corresponding to a location l_i in A_1, then Line 3 and Line 4 split l_i into m locations $l_{i_1}, \cdots l_{i_m}$ in the new automaton, say A_2. For each of these newly created locations, Line 6 to Line 11 determine the constraints on their incoming edges.

For each incoming edge $l_r \xrightarrow{a,g_r,R_r} l_i$, 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_r in the intersection of $Z_{i_j} \uparrow$ with $\mathbb{R}^{|C|}_{\geq 0}$. In Line 11, $free(Z'_{i_j}, R_r)$ represents a zone that becomes the same as Z'_{i_j} after resetting the clocks in R_r. Further, g_{r_j} is calculated as the weakest guard that simultaneously satisfies the constraints g_r, Z_{r_j} and $free(Z'_{i_j}, R_r)$ and has the same set of clocks as in g_r. For our running example, if we consider $Z_{i_j} = Z_1$ then we have $Z'_{i_j} = Z_{i_j} \uparrow \cap\, \mathbb{R}^2_{[x \leftarrow 0]} = \{x = 0, y < 2\}$, $Z_{r_j} = \{x = y, x < 2\}$ and $free(Z'_{i_j}, \{x\}) = \{x \geq 0, y < 2\}$. We can see that $x < 2$ is the weakest formula such that $x \leq 4 \wedge x \geq 0 \wedge y < 2 \wedge x = y \wedge x < 2 \Rightarrow x < 2$ holds and hence $g_{r_j} = \{x < 2\}$.

Loop from Line 14 to Line 28 determines the constraints on the outgoing edges from these new locations. Line 15 checks if the zone $Z_{i_j} \uparrow$ 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_j} and hence this edge is not added in A_2. Loop from Line 19 to Line 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 constraints 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 from l_{l_1} to l_2 even though it was present on the edge from l_1 to l_2. The reason being that the future of the zone of l_{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_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 bounded by the number of zones in the zone graph. This is exponential in the number of clocks of the original TA A. However, we note that the base zones of a location l in the original TA are distributed across multiple locations as a result of the split of l and no new valuations are created. This gives us the following lemma.

Lemma 3. *The splitting procedure described in this stage does not increase the number of clocks in A_2, but the number of locations in A_2 may become exponential in the number of clocks of the given TA A. However, there is no addition of new valuations to the underlying state space of the original TA A and corresponding to every state (l, v) of a location l in the original TA A, exactly one state (l_i, v) is created in the modified TA A_2, where l_i is one of the newly created locations as a result of splitting l.*

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.*

The number of locations after the split can become 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 the timed automaton.

Stage 3: Removing constraints by considering multiple edges with the same action: We consider the example in Figure 4. 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 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 $a \in Act$, it determines the zones of l_i from which action a is enabled. We call this set $\mathcal{Z}_{ia}$. Zone graph construction and splitting of locations in stage 2 ensures that all zones in $\mathcal{Z}_{ia}$ form a linear chain connected by ε edges as shown in Figure 5. We use $\lessdot$ 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. $Z_{i_1} \lessdot \cdots \lessdot Z_{i_k} \lessdot Z_{i_{k+1}} \cdots \lessdot Z_{i_m}$. Lemma 1 ensures that for each Z_{i_k}, $k \geq 1$, that is bounded above, 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. Further, $UB(Z)$ is ∞ if Z is not bounded from above.

Let $\Gamma_{(l_i,a)} = \{g \mid l_i \xrightarrow{a,g,R} l' \in E_{A_2}\}$ 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;

- $\mathrm{Strt}(g)_{(l_i,a)} = Z \in \mathcal{Z}_{ia}$ is the zone in $\mathcal{Z}_{ia}$ which is bounded from below by the same constraints as the lower bound of the constraints in g.
- $\mathrm{End}(g)_{(l_i,a)} = Z \in \mathcal{Z}_{ia}$ is the zone in $\mathcal{Z}_{ia}$ which is bounded from above by the same constraints as the upper bound of the constraints in g.
- $\mathrm{Ran}(g)_{(l_i,a)} = \{Z \in \mathcal{Z}_{ia} \mid \mathrm{Strt}(g)_{(l_i,a)} \lessdot Z \wedge Z \lessdot \mathrm{End}(g)_{(l_i,a)}\} \cup \{\mathrm{Strt}(g)_{(l_i,a)}\} \cup \{\mathrm{End}(g)_{(l_i,a)}\}$ is the set of zones ordered by $\lessdot$ relation in between $\mathrm{Strt}(g)_{(l_i,a)}$ and $\mathrm{End}(g)_{(l_i,a)}$.

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 the upper bounds of g respectively. If g does not have any constraint defining the upper bound then $C_2 = \infty$. We define a total order $\lll$ on $\Gamma_{(i,a)}$ such that for any $g, g' \in \Gamma_{(i,a)}$,

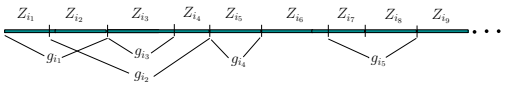

Fig. 5. Merging constraints in stage 3

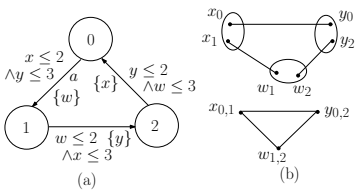

Fig. 6. Colouring clock graph

$g \lll g'$ iff $\exists Z \in \text{Ran}(g)_{(l_i,a)}$ such that $Z \lessdot Z'$ for all $Z' \in \text{Ran}(g')_{(l_i,a)}$. Similar to the zones let us use ordered indexed variable $g_{i_1}, \ldots, g_{i_p}$ to denote $g_{i_1} \lll \cdots g_{i_k} \lll g_{i_{k+1}} \cdots \lll g_{i_p}$. One such total order on guards is shown in Figure 5. 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 *next* initialized to 2. In every iteration of this loop the invariant $g_{curr} \lll g_{i_{next}}$ 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_{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 A' is set as the current automaton which is A_3 (Line 13 and Line 29) and *next* is incremented to process the next guard. Otherwise the guard g_{curr} is set to $g_{i_{next}}$ 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 $\text{Strt}(g_{curr})_{(l_i,a)}$ and the upper bound of $\text{End}(g_{i_{next}})_{(l_i,a)}$. For case (iii), there are three possibilities of combining guards, mentioned in Line 20, Line 22 and Line 24. The first possibility is the same as in case (ii). The second and the third possibilities are replacing the upper bound of g_{curr} with the lower bound of $\text{Strt}(g_{i_{next}})_{(l_i,a)}$ and the lower bound of $g_{i_{next}}$ with the upper bound of $\text{End}(g_{curr})_{(l_i,a)}$ 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 [21,13]. This is why we create the pre-stable zone graph as described in Section 2 as it enables one to directly check timed bisimilarity on this zone graph [13].

Lemma 5. *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 zone graph of the TA obtained after stage 2. From Lemma 3, in stage 2, no new valuations are added to the underlying state space of the original TA A, and corresponding to every valuation (l, v), exactly one valuation (l_i, v) is created, where l and l_i are as described in Lemma 3. Thus the number of zones

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

$A_3 := A_2$ ▷ A_2 is the TA obtained from A after the first two stages
for each location l_i in A_2 **do** ▷ i is the index of the location, the set of locations do not change in this stage
  for each $a \in sort(l_i)$ **do** ▷ $sort(l_i)$ is the set of actions in l_i that can be performed from l_i
    $g_{curr} := g_{i_1}$, $next := 2$
    while $next < |\Gamma_{(i,a)}| - 1$ **do**
      if $\mathrm{Ran}(g_{curr})_{(l_i,a)} \cup \mathrm{Ran}(g_{i_{next}})_{(l_i,a)}$ is not convex **then**
        $g_{curr} := g_{i_{next}}$, $next := next + 1$
      else if $\mathrm{Ran}(g_{i_{next}})_{(l_i,a)} \cap \mathrm{Ran}(g_{curr})_{(l_i,a)} = \emptyset$ **then** ▷ non-overlapping but contiguous
        $g'_{curr} := [LB(\mathrm{Strt}(g_{curr})_{(l_i,a)}), UB(\mathrm{End}(g_{i_{next}})_{(l_i,a)})]$
        $g'_{i_{next}} := g'_{curr}$
        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
        if A' is timed bisimilar to A_3 **then**
          $A_3 := A'$, $g_{curr} := g'_{i_{next}}$, $next := next + 1$
        else
          $g_{curr} := g_{i_{next}}$, $next := next + 1$
        end if
      else ▷ $\mathrm{Ran}(g_{curr})_{(l_i,a)}$ and $\mathrm{Ran}(g_{i_{next}})_{(l_i,a)}$ have overlapping zones
        ▷ There are three ways to combine g_{curr} and $g_{i_{next}}$, and
        ▷ Resultant new guards should be checked for timed bisimilarity in the following order
        (i). $g'_{curr} := [LB(\mathrm{Strt}(g_{curr})_{(l_i,a)}), UB(\mathrm{End}(g_{i_{next}})_{(l_i,a)})]$,
          $g'_{i_{next}} := g'_{curr}$
        (ii). $g'_{curr} := [LB(\mathrm{Strt}(g_{curr})_{(l_i,a)}), LB(\mathrm{Strt}(g_{i_{next}})_{(l_i,a)})]$,
          $g'_{i_{next}} := [LB(\mathrm{Strt}(g_{i_{next}})_{(l_i,a)}), UB(\mathrm{End}(g_{i_{next}})_{(l_i,a)})]$
        (iii). $g'_{curr} := [LB(\mathrm{Strt}(g_{curr})_{(l_i,a)}), UB(\mathrm{End}(g_{curr})_{(l_i,a)})]$,
          $g'_{i_{next}} := [UB(\mathrm{End}(g_{curr})_{(l_i,a)}), UB(\mathrm{End}(g_{i_{next}})_{(l_i,a)})]$
        while $1 \leq i \leq 3$ **do** ▷ Corresponding to the three cases above
          Let A' be the TA obtained by replacing all occurrences of g_{curr} and $g_{i_{next}}$ with the i^{th} g'_{curr} and $g'_{i_{next}}$ respectively in A_3
          if A' is timed bisimilar to A_3 **then**
            $A_3 := A'$, $g_{curr} := g'_{i_{next}}$, $next := next + 1$
            Break
          else
            $i := i + 1$
          end if
        end while
        if $i = 4$ **then** ▷ Bisimilarity could not be preserved in any of these three cases
          $g_{curr} := g_{i_{next}}$, $next := next + 1$
        end if
      end if
    end while
  end for
end for

in the zone graph of A_2 is still exponential in the number of cocks of the original TA A. Checking timed bisimilarity is done in EXPTIME [7,17]. A 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: Finding Active clocks, clock replacement and renaming: Given a location l, an iterative method for finding the set of active clocks at l,denoted $act(l)$, is given in [9]. The method has been modified and stated below for the case where clock assignments of the form $x := y, x, y \in C$ are disallowed.

Determining active clocks : For a location l, let $clk(l)$ be the set of clocks that appear on the constraints in the outgoing edges of l. Let $\rho : (2^C \times E) \rightarrow 2^C$ be a partial function such that for an edge $e = l \xrightarrow{g,a,R} l'$, $\rho(act(l'), e)$ gives the set of active clocks of l' that are not reset along e. For all $l \in L$, $act(l)$ is the limit of the convergent sequence $act_0(l) \subseteq act_1(l) \ldots$ such that $act_0(l) := clk(l)$ and $act_{i+1}(l) := act_i(l) \cup \bigcup_{e=(l,g,a,R,l') \in E} \rho(act_i(l'), e)$.

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 $x \notin act(l)$.

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. Any $x, y \in act(l)$ belong to an equivalence class iff the same relation of the form $x - y = k$, for some fixed 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 π_l be the partition induced by this equivalence relation.

We note that the size of the largest partition does not give the minimum 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 partitioned into two different classes 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_{A_3}, for the timed automaton A_3 is constructed in the following way. This graph contains a vertex for each class in the partition π_l, for every location l. Let V_l be the set of vertices corresponding to the classes of π_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 π_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 and the resultant graph is termed 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 $\mathcal{T}$, let $loc(\mathcal{T})$ represent the set of locations whose active clocks are members of $\mathcal{T}$.

Finding the minimum number of clocks to represent the TA A_3 is thus equivalent to colouring its clock 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 $\mathcal{T}$, are renamed c. The value of c can be chosen to be equal to some clock in $\mathcal{T}$ that is considered to be the *representative clock* for that class. The constraints involving the rest of these clocks in $\mathcal{T}$ are adjusted appropriately and any resets of the clocks, different from the representative clock, present on the incoming edges to l such that $l \in loc(\mathcal{T})$ are also removed.

For example, suppose vertex r corresponds to a class $\mathcal{T}$ 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 $\mathcal{T}$ 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 $\mathcal{T}$ is replaced with $y + k_1$, while every occurrence of z in $\mathcal{T}$ 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. Since this stage consists of finding the active clocks and renaming them, we have the following lemma.

Lemma 6. *The operations in stage 4 produce a timed automaton A_4 that is timed bisimilar to the TA A_3 obtained after stage 3.*

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 those active clocks of the locations l' such that there exists a path of length at most i between l and l' and these clocks are not reset along this path. 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 of 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 [18,11]. Since the number of locations after the splitting operation in stage 2 is exponential in the number of clocks in A, renaming 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 such that $Z' = (Z \cap g)_{[R \leftarrow \overline{0}]}\uparrow \cap I(l')$, if $(Z \cap g)_{[R \leftarrow \overline{0}]}\uparrow \cap I(l') \neq \emptyset$, 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, $(Z \cap g)_{[R \leftarrow \overline{0}]} \uparrow \cap I(l') = \emptyset$. 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, $(Z \cap g)_{[R \leftarrow \overline{0}]} \uparrow \cap I(l') = (Z \cap g)_{[R \leftarrow \overline{0}]} \uparrow$ holds. Considering the clock relations in the zones of a location, 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: Let A_4 be the TA obtained from a TA 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 7. *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 : We prove this lemma by induction on the structure of the timed automaton A_4. 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 $act(l) = clk(l) \cup \left(\bigcup_{i=1,\, e=(l,g,a,R_i,l_i) \in E}^{m} (act(l_i) \setminus R_i) \right)$.

Consider a clock $c_0 \in act(l)$. If $c_0 \in act(l_i)$, for some $i \in \{1, \ldots, m\}$, and c_0 is not reset on the edge between l and l_i, then from the induction hypothesis, the lemma holds for l trivially. Otherwise, if for each $i \in \{1, \ldots, m\}$, such that $c_0 \in act(l_i)$ implies c_0 is reset on the edge from l to l_i, we can show that there exists a constraint on an outgoing edge from l involving c_0 that cannot be removed while preserving timed bisimilarity. □

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 TA that is minimal bisimilar to A_4 can replace the clocks of A_4 which gives us the following lemma.

Lemma 8. *The timed automaton A_4 has the same number of clocks as a minimal bisimilar TA for A.*

Proof sketch : We consider a minimal bisimilar TA D_1 for A and apply the operations in the four stages on it to produce a TA D. Since D_1 is already minimal, D has the same number of clocks as D_1. The transformations ensure that in the resultant zone graph of D, for each bounded zone in every location, there exists a hyperplane that fully bounds it. Similarly in the zone graph of A_4 too, each bounded zone of every location is fully bounded by a hyperplane.

We use the zone graphs of A_4 and D for mapping the clocks of A_4 to the clocks of D. From Lemma 7, 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 $x \bowtie k$ bounding a zone that cannot be removed if timed bisimulation has to be preserved. Since A_4 and D are timed bisimilar, there is a corresponding hyperplane, say $y = k'$, induced by a constraint $y \bowtie k'$ which too cannot be removed from the zone graph of D while preserving timed bisimulation. Clock x in A_4 can thus be renamed y. The clock renaming in stage 4 ensures that, any further renaming cannot reduce the number of clocks in A_4. Hence if the clocks of A_4 can be replaced with the clocks of D, we have $|C_{A_4}| \leq |C_D|$, thus giving $|C_{A_4}| = |C_D|$ since D is a minimal bisimilar TA for 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 runs in time that is doubly exponential in the number of clocks of A.*

4 Conclusion

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. The problem solved in this paper was left open in [16].

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 [9]. 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. There is an exponential increase in the number

of locations while producing the TA with the minimal number of clocks. However, there is no addition of new valuations to the underlying state space of the timed automaton since the splitting of a location l, described in stage 2, involves distributing the zones of l across the locations l is split into.

References

1. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press (2007)
2. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Information and Computation 104, 2–34 (1993)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
4. Behrmann, G., Bouyer, P., Fleury, E., Larsen, K.G.: Static guard analysis in timed automata verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 254–270. Springer, Heidelberg (2003)
5. Behrmann, G., Bouyer, P., Larsen, K.G., Pelanek, R.: Lower and upper bounds in zone-based abstractions of timed automata. International Journal on Software Tools for Technology Transfer 8, 204–215 (2006)
6. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time petri nets. In: IFIP Congress, pp. 41–46 (1983)
7. Čerāns, K.: Decidability of bisimulation equivalences for parallel timer processes. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 302–315. Springer, Heidelberg (1993)
8. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 313–329. Springer, Heidelberg (1998)
9. Daws, C., Yovine, S.: Reducing the number of clock variables of timed automata. In: IEEE Proc. RTSS 1996, pp. 73–81. IEEE Computer Society Press (1996)
10. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
11. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. Journal of Graph Algorithms and Applications 7, 131–140 (2003)
12. Finkel, O.: Undecidable problems about timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 187–199. Springer, Heidelberg (2006)
13. Guha, S., Krishna, S.N., Narayan, C., Arun-Kumar, S.: A unifying approach to decide relations for timed automata and their game characterization. In: Express/SOS, pp. 47–62 (2013)
14. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic Model Checking for Real-time Systems. In: Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS 1992), Santa Cruz, California, USA, June 22-25, pp. 394–406. IEEE Computer Society (1992)
15. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. In: The Proceedings of Logic in Computer Science, LICS (1992)
16. Laroussinie, F., Larsen, K.G., Weise, C.: From timed automata to logic – and back. In: Wiedermann, J., Hájek, P. (eds.) MFCS 1995. LNCS, vol. 969, pp. 529–539. Springer, Heidelberg (1995)

17. Laroussinie, F., Schnoebelen, P.: The state explosion problem from trace to bisimulation equivalence. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 192–207. Springer, Heidelberg (2000)
18. Lawler, E.L.: A note on the complexity of the chromatic number problem. Information Processing Letters 5, 66–67 (1976)
19. Tripakis, S.: Folk theorems on the determinization and minimization of timed automata. Information Processing Letters 99, 222–226 (2006)
20. Tripakis, S., Yovine, S.: Analysis of timed systems using time-abstracting bisimulations. Formal Methods in System Design 18, 25–68 (2001)
21. Weise, C., Lenzkes, D.: Efficient scaling-invariant checking of timed bisimulation. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 177–188. Springer, Heidelberg (1997)