On Decidability of Prebisimulation for Timed Automata

Shibashis Guha, Chinmay Narayan, S. Arun-Kumar

Department of Computer Science and Engineering,
Indian Institute of Technology,
Delhi.
{shibashis,chinmay,sak}@cse.iitd.ac.in

Abstract. In this paper, we propose an *at least as fast as* relation between two timed automata states and investigate its decidability. The proposed relation is a prebisimulation and we show that given two processes with rational clock valuations it is decidable whether such a prebisimulation relation exists between them. Though bisimulation relations have been widely studied with respect to timed systems and timed automata, prebisimulations in timed systems form a much lesser studied area and according to our knowledge, this is the first of the kind where we study the decidability of a timed prebisimulation. This prebisimulation has been termed *timed performance prebisimulation* since it compares the efficiency of two states in terms of their performances in performing actions. $s \precsim t$ if s and t are time abstracted bisimilar and every possible delay by s and its successors is no more than the delays performed by t and its successors where the delays are real numbers. The prebisimilarity defined here falls in between timed and time abstracted bisimilarity.

Key words: Timed automata, timed bisimulation, time abstracted bisimulation, prebisimulation, timed transition system

1 Introduction

Bisimulation [17] is an important relation to establish the equivalence between two reactive systems. The concept has been extended to timed systems as well. The common form of bisimulations used to compare two timed systems are *timed* and *time abstracted* bisimulations. While timed bisimulation is too strong a relation where time delays have to match exactly, time abstracted bisimulation does not compare exact timing requirements. Both timed and time-abstracted bisimulations have been proved to be decidable for timed automata [6][16][21].

Timing requirements, in real time systems in particular, increasingly affect design decisions and implementation procedure. We propose a prebisimulation relation between two timed automata states which establishes an *at least as fast as* relation between them. Using action hiding or action abstraction, the proposed bisimulation relation will be useful in comparing functionally equivalent

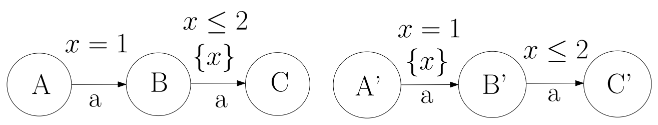


Fig. 1. Example: A preorder relation

systems in terms of their relative performance. We call this *timed performance prebisimulation* as it distinguishes two states based on their performance for doing each action they are capable of doing. This relation will be useful for verifying the correctness of implementations of systems with multiple blocks where each individual block needs to finish execution within a specified time. The verification will conclude whether the timing requirement for each individual block as mentioned in the specification is satisfied in the implementation. An important consideration here is that both the states should be capable of performing the same set of actions at any point in time and hence they should be time abstracted bisimilar. Thus timed performance prebisimulation is stronger than time abstracted bisimulation but weaker than timed bisimulation. The automata shown in figure 1 illustrate the idea. The automaton in the left is *at least as fast as* the automaton on the right, since the second a action should be performed within a time interval of one time unit after the first a action whereas in the second timed automaton, the second a can be performed within an interval of two time units after the first action. Some *speed sensitive* preorder relations have been defined in process algebraic framework [7][18][15], though each of these relations is significantly different from our timed performance prebisimulation which is defined for timed automata states. In [12] too a similiar relation based on traces is proposed. In our approach we can capture non-determinism in terms of both time and action and the relation captures branching in time as in bisimulation rather than traces. Also, our relation is particular to the widely studied timed automata formalism whereas the formalism used in [12] is timed actor interfaces. As in the work in [12], in our approach too, a smaller nondeterministic delay against a greater constant time delay is considered to be a refinement. Our main contribution is proving the decidability of timed performance prebisimulation using a zone [4][9] based construction. This approach can also be used to prove the decidability of timed bisimulations. For timed bisimulation, our construction can lead to a simpler proof than the product construction technique used on regions in [6] or the zone based technique used in [21]. In section 2, we give a brief description of timed automata. In section 3, we describe strong timed and time-abstracted bisimulations while in section 4, we introduce *timed performance prebisimulation*. In section 5, we briefly describe zone and zone graph [14][22]. Our algorithm for proving decidability uses *zone valuation graph* which is a zone graph starting from a specific timed state and satisfies certain properties as described later. We describe a method for creating zone valuation graph in section 5 and provide an algorithm for deciding this prebisimulation and a proof of correctness for it. We conclude in section 6 where we give an example comparing two complex systems using this performance prebisimulation. We also mention about another timed prebisimulation which we consider to be of significant importance and whose decidability we would like to investigate in future.

2 Timed Automata

Timed automata [2] is an approach to model time critical systems where the system is modeled with *clocks* that track elapsed time. Timing of actions and time invariants on states can be specified using this model.

A timed automaton is a finite-state structure which can manipulate real-valued clock variables. Corresponding to every transition, a subset of the clocks can be specified that can be *reset* to zero. In this paper, the clocks that are reset in a transition are shown as being enclosed in braces. Clock constraints also specify the condition for actions being enabled. If the constraints are not satisfied, the actions will be disabled. Constraints can also be used to specify the amount of time that may be spent in a location. The clock constraints $\mathcal{B}(C)$ over a set of clocks C is given by the following grammar:

$$g ::= \ x < c \mid x \leq c \mid x > c \mid x \geq c \mid g \wedge g$$

where $c \in \mathbb{N}$ and $x \in C$. A *timed automaton* over a finite set of clocks C and a finite set of actions *Act* is a quadruple (L, l_0, E, I) [1] where L is a finite set of locations, ranged over by l, $l_0 \in L$ is the initial location, $E \subseteq L \times \mathcal{B}(C) \times Act \times 2^C \times L$ is a finite set of *edges*, and $I : L \to \mathcal{B}(C)$ assigns invariants to locations.

2.1 Semantics

The semantics of a timed automaton can be described with a *timed labeled transition system*(TLTS). Let $A = (L, l_0, E, I)$ be a timed automaton over a set of clocks C and a set of visible actions *Act*. The timed transition system $T(A)$ generated by A can be defined as $T(A) = (Proc, Lab, \{\xrightarrow{\alpha} | \alpha \in Lab\})$, where $Proc = \{(l, v) \,|\, (l, v) \in L \times (C \to \mathbb{R}_{\geq 0})$ and $v \models I(l)\}$, i.e. *states* are of the form (l, v), where l is a location of the timed automaton and v is a valuation that satisfies the invariant of l. We use the terms *process* and *state* interchangeably in this text. $Lab = Act \cup \mathbb{R}_{\geq 0}$ is the set of labels; and the transition relation is defined by $(l, v) \xrightarrow{a} (l', v')$ if for an edge $(l \xrightarrow{g,a,r} l') \in E$, $v \models g, v' = v[r]$ and $v' \models I(l')$, where an edge $(l \xrightarrow{g,a,r} l')$ denotes that l is the source location, g is the guard, a is the action, r is the set of clocks to be reset and l' is the target location. $(l, v) \xrightarrow{d} (l, v + d)$ for all $d \in \mathbb{R}_{\geq 0}$ such that $v \models I(l)$ and $v + d \models I(l)$ where $v + d$ is the valuation in which every clock value is incremented by d. Let v_0 denote the valuation such that $v_0(x) = 0$ for all $x \in C$. If v_0 satisfies the invariant condition of the initial location l_0, then (l_0, v_0) is the initial state or the initial configuration of $T(A)$.

3 Bisimulations for Timed Systems

We discuss here only the strong form of the bisimulations.

Definition 1. ***Timed bisimilarity****: A binary symmetric relation $\mathcal{R}$ over the set of states of a TLTS is a* timed bisimulation relation *if whenever $s_1 \mathcal{R} s_2$, for each action $a \in Act$ and time delay $d \in \mathbb{R}_{\geq 0}$*

if $s_1 \xrightarrow{a} s_1'$ then there is a transition $s_2 \xrightarrow{a} s_2'$ such that $s_1' \,\mathcal{R}\, s_2'$, and
if $s_1 \xrightarrow{d} s_1'$ then there is a transition $s_2 \xrightarrow{d} s_2'$ such that $s_1' \,\mathcal{R}\, s_2'$.
Timed bisimilarity $\sim_t$ is the largest timed bisimulation relation.

Timed automata A_1 and A_2 are *timed bisimilar* if the initial states in the corresponding TLTS are timed bisimilar. Matching each time delay in one automaton with identical delays in another automaton may be too strict a requirement. Time abstracted bisimilarity is the relation obtained by a relaxation of this requirement and the second clause of definition 1 is replaced by
if $s_1 \xrightarrow{d} s_1'$ then there is a transition $s_2 \xrightarrow{d'} s_2'$, such that $s_1' \mathcal{R} s_2'$. The delay d can be different from d'.

Timed automata A_1 and A_2 are *time abstracted bisimilar* if the initial states in the corresponding TLTS are time abstracted bisimilar.

4 Timed Performance Prebisimulation

Deciding whether one automaton is at least as fast as another is interesting when they can perform the same visible actions at every stage. Thus they should be time abstracted bisimilar. This prebisimulation is similar to efficiency preorder [3] where the efficiency of two systems is compared using the number of internal moves made by them.

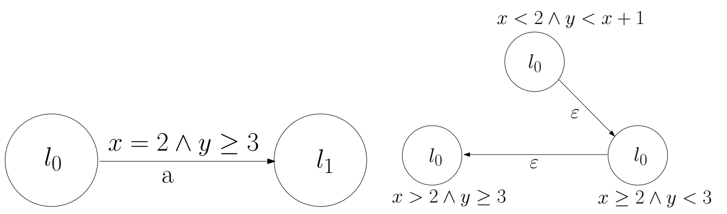
Definition 2. ***Timed performance prebisimilarity****: A binary relation $\mathcal{B}$ over the set of states of a TLTS is a* timed performance prebisimulation relation *if whenever $s_1 \mathcal{B} s_2$, for each action a and time delay d*

if $s_1 \xrightarrow{a} s_1'$ then there is a transition $s_2 \xrightarrow{a} s_2'$ such that $s_1' \mathcal{B} s_2'$, and
if $s_2 \xrightarrow{a} s_2'$ then there is a transition $s_1 \xrightarrow{a} s_1'$ such that $s_1' \mathcal{B} s_2'$, and
if $s_1 \xrightarrow{d} s_1'$ then there is a transition $s_2 \xrightarrow{d'} s_2'$ for $d \leq d'$ such that $s_1' \mathcal{B} s_2'$,and
if $s_2 \xrightarrow{d} s_2'$ then there is a transition $s_1 \xrightarrow{d'} s_1'$ for $d \geq d'$ such that $s_1' \mathcal{B} s_2'$.
Timed performance prebisimilarity $\precsim$ is the largest timed performance prebisimulation relation.

Clearly $\sim_t \subset \precsim \subset \sim_u$. In this paper, for brevity, we use the term *timed performance prebisimilarity* and *performance prebisimilarity* interchangeably.

5 Zone Valuation Graph and Deciding Prebisimulation

We present a zone graph based algorithm which, given two timed automata states s_1 and s_2 with rational clock valuations, decides whether or not $(s_1, s_2) \in \precsim$.



(a) A timed automaton with guards (b) Zone graph as produced by *minim*

Fig. 2. A timed automaton in (a) and its zone valuation graph corresponding to process $\langle l_1, (0, 0.6)\rangle$ in (b)

We first discuss the definition of a zone graph and the properties that must hold for it to be applicable in our algorithm. A zone graph that satisfies these properties is referred to as a *zone valuation graph.* The concept is introduced in subsection 5.1 and is used in subsection 5.2 to prove that the prebisimulation relation is decidable. In subsection 5.3, we give an algorithm for constructing *zone valuation graphs* and discuss its complexity. After this we present an algorithm in subsection 5.4 which uses this *zone valuation graph* to check the existence of a performance prebisimulation relation between two states of timed automata.

5.1 Zone Valuation Graph

The following two definitions are from [21].

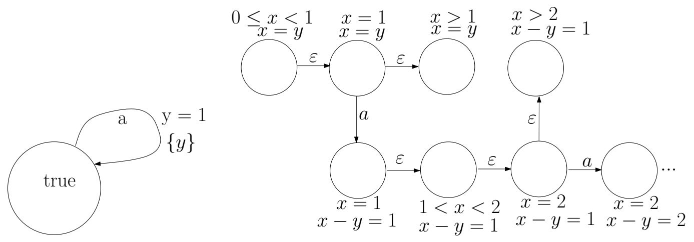
Definition 3. ***zone:*** *The characteristic set of a linear formula ϕ, a clock constraint of the form $x \smile c$ or a diagonal constraint of the form $x - y \smile c$, where $x, y \in C$, is the set of all valuations for which ϕ holds. A zone is a finite union of characteristic sets.*

A *zone graph* is similar to a *region graph*[1] with the difference that each node consists of a timed automaton location and a zone.

Definition 4. ***zone graph:*** *For a timed automaton $P = (L, l_0, E, I)$, a zone graph is a transition system $(S, s_0, Lep, \rightarrow)$, where $Lep = Act \cup \{\varepsilon\}$, ε is an action corresponding to delay transitions of the processes of the zone, $S \subseteq L \times \Phi_\vee(C)$ is the set of nodes, $s_0 = (l_0, \phi_0(C))$, $\rightarrow \subseteq S \times Lep \times S$ is connected, $\phi_0(C)$ is the formula where all the clocks in C are 0 and $\Phi_\vee(C)$ denotes the set of all zones.*

Definition 5. ***Bisimulation between zone graphs***
For two zone graphs, $Z_1 = (S_1, s_1, Lep, \rightarrow_1)$ and $Z_2 = (S_2,\ s_2,\ Lep,\ \rightarrow_2)$, Z_1 is strongly bisimilar[17] to Z_2, denoted as $Z_1 \sim Z_2$, iff the nodes s_1 and s_2 are strongly bisimilar, denoted by $s_1 \sim s_2$. While checking strong bisimulation between the two zone graphs, ε is considered visible similar to an action in Act.

It is possible to have more than one zone graph corresponding to a timed automaton. For a timed automaton $A = (L, l_0, E, I)$ and a process $r = (l_j, v_{l_j}) \in T(A)$,



(a) Timed automaton producing infinite zone graph

(b) An infinite zone graph

Fig. 3. An automaton in (a) with its infinite zone graph in (b)

we are interested in a particular form of zone graph $Z_{(A,r)}$=$(S, s_r, Lep, \rightarrow)$ which satisfies the following properties:

1. set S is finite.
2. For every node $s \in S$ the zone corresponding to the constraints ϕ_s is convex.
3. $v_{l_j} \models \phi_{s_r}$. Note that v_{l_j} may or may not satisfy $\phi_0(C)$.
4. For any two processes $p, q \in T(A)$, if their valuation satisfies the formula ϕ_r for the same node $r \in S$ then $p \sim_u q$, i.e. p is time abstracted bisimilar to q.
5. For two timed automata A_1, A_2 and two processes $p \in T(A_1)$ and $q \in T(A_2)$, $Z_{(A_1,p)} \sim Z_{(A_2,q)} \Leftrightarrow p \sim_u q$.
6. It should be minimal to the extent of preserving convexity of the zones and gives a canonical form for checking performance prebisimulation.

Zone graph created with the algorithm given in section 5.3 of [20] satisfies most of these properties. It is finite by construction and does not require any abstraction mechanism to ensure finiteness of the zone graph. All the zones are convex and any two processes satisfying the same zone formula in the same node are time abstracted bisimilar. The algorithm described there has been implemented in the tool *minim*. Though for most of the timed automata the zone graph constructed by *minim* preserves all the properties mentioned above, we found that it does not produce the minimal graph preserving time abstracted bisimulations for certain timed automata where all locations are not reachable. This is due to the fact that in *minim* the zones are split based on canonical decomposition of the guards on the outgoing edges. For example, let us look at the automaton in figure 2(a). Due to the guards on the outgoing edges the initial zone of location l_0 is split into sub-zones by *minim* as shown in figure 2(b). However, no valuation in l_0 can satisfy the guard and therefore cannot make an a transition. Therefore a zone graph algorithm producing the minimal graph should not split the zone.

While minim uses a backward method, in our approach, while generating the zone valuation graph, we use a combination of both forward and backward analysis. One must note that forward analysis may cause a zone graph to become infinite [10]. Let us consider the automaton in figure 3(a). Since clock y is reset while x keeps increasing, the number of zones becomes infinite as shown in figure

3(b). To ensure finiteness of the zone graph, we require an abstraction on the zones. Several zone abstractions have been suggested in the literature [8][10][11]. We consider location dependent maximal constants abstraction [10]. For each clock $x \in C$ and each location $l \in L$, a maximum constant max_x^l is determined beyond which the actual value of x in l is irrelevant. For a location l and a clock x, $max_x^l \leq c_x$, the global maximal constant with which clock x is compared. This reduces the number of zones compared to the one obtained using region graph abstraction.

Our algorithm constructs a zone graph whose nodes are the time abstracted bisimulation classes of the automaton preserving the convexity of zones. Thus it is possible to have two or more nodes in the zone graph which are time abstracted bisimilar to each other. Similar to *minim* [20], we use the guards on transitions to split a zone corresponding to a location but unlike minim, it is split only when it is found to be reachable from the initial location using a forward analysis [10][5] approach. One location can be reached through multiple paths and thus a split of a zone in a location also causes its *ancestors* to be split accordingly. This is done using the backward method as in minim. Like minim, the notion of *stability* is used for the purpose of splitting a zone. We present below a brief description of stability as given in [20]. Given an edge $v \to u$, where v and u are zones in a zone graph, v is a *predecessor* of u and u is a successor of v where u and v are nodes in a zone graph. Function $preds(v)$ (resp. $succs(v)$) denotes the set of predecessors (resp. successors) of v.

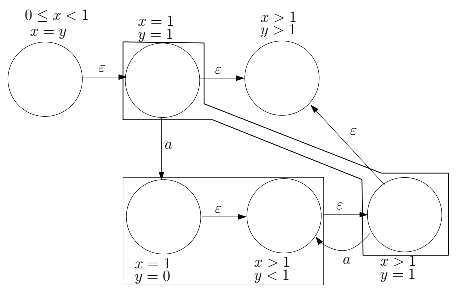
Definition 6. ***Stable partitions****: Given two classes $C_1, C_2 \in \mathcal{C}$, C_1 is said to be* pre-stable *with respect to C_2 if either $C_1 \subseteq preds(C_2)$ or $C_1 \cap preds(C_2) = \emptyset$. C_2 is said to be* post-stable *with respect to C_1 if either $C_2 \subseteq succs(C_1)$ or $succs(C_1) \cap C_2 = \emptyset$. In this paper, we use the term stability which implies either pre-stability or post-stability depending on the particular situation. We say that C_1 and C_2 are stable with respect to each other if C_1 is pre-stable with respect to C_2 and C_2 is post-stable with respect to C_1.*

Figure 4(a) shows the zone graph after using abstraction with location dependent maximal constants for the timed automaton of figure 3(a). The nodes in the polygonal enclosures are strongly bisimilar to each other such that their union gives a convex zone. Such nodes in the enclosures are combined in the next phase to obtain the *canonical* form of the zone graph shown in figure 4(b). The nodes in the canonical zone graph represent the time abstracted bisimilar classes of the zones preserving convexity.

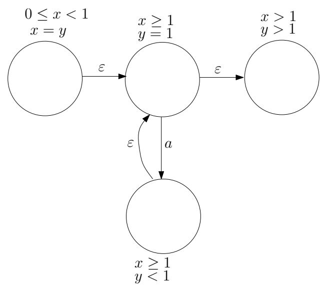
5.2 Decidability of Timed Performance Prebisimulation

We now discuss how zone valuation graph can be used in checking the existence of timed performance prebisimulation relation between two TLTS states. For this purpose, we first introduce a few notations and definitions which will be used in proving the correctness of our approach.

For timed automaton A and process $p \in T(A)$, let $Z_{(A,p)} = (S, s_0, Lep, \to)$ be the zone valuation graph of p. For any state $q \in T(A)$, let $\mathcal{N}(q)$ represent the



(a) Zone graph with abstraction for timed automaton 3(a)



(b) Canonical zone graph for timed automaton 3(a)

Fig. 4. (a) Zone abstraction for automaton in 3(a) and its canonical representation in (b)

node of the same location as that of q and whose clock valuation range includes the valuation of q. Similarly, for any node $s \in S$, let $\mathcal{G}(s)$ represent the set of all processes reachable from p with the same location as that of s and whose valuations are satisfied by ϕ_s.

Based on the clock constraints of the nodes in the *zone valuation graph*, we define **span** as below,

Definition 7. ***Span:*** *For a given node $s \in S$ and a clock $x \in C$, $min_x(s)$ and $max_x(s)$ represent the minimum and the maximum clock valuations of a clock x across all processes in the node s. For $x \ge c$, $min_x(s) = c$, for $x > c$, $min_x(s) = c + \delta$. For $x \le c$, $max_x(s) = c$, for $x < c$, $max_x(s) = c - \delta$, where δ is a symbolic value representing an infinitesimally small value. We define $range(x, s)$ as $max_x(s) - min_x(s)$. The span of a node $s \in S$ is defined as $\mathcal{M}(s) = min\{range(x, s) \mid x \in C\}$, i.e. minimum of all clocks' ranges. We define a clock y belonging to the set $\{y \mid range(y, s) = \mathcal{M}(s)\}$ to be a critical clock.*

It is to be noted that for a given node s, $range(x, s)$ is the same for all clock variables $x \in C$ if the zone corresponding to node s is not abstracted with respect to any clock variable. If the zone corresponding to s is abstracted with respect to one or more clock variables then for each such variable $x \in C$, $range(x, s) = \infty$. For example, in a zone valuation graph with two clocks x and y, the span for a node s with $\phi_s = x > 3$ and $y < 1$ is $min(\infty, 1 - \delta) = 1 - \delta$ whereas span for a node with $\phi_s = x > 1$ and $y = 2$ is $min(\infty, 0) = 0$.

Definition 8. *Flip in delay: For timed automata A_1 and A_2 and processes $p \in T(A_1)$, $q \in T(A_2)$, let $Z_{(A_1,p)} = (S_1, s_1, Lep, \rightarrow_1)$ and $Z_{(A_2,q)} = (S_2,\ s_2,\ Lep, \rightarrow_2)$ be the corresponding zone valuation graphs. If $Z_{(A_1,p)} \sim Z_{(A_2,q)}$ then flip in delay, denoted by $FID(Z_{(A_1,p)}, Z_{(A_2,q)})$ is true iff there does not exist any strong bisimulation relation $\mathcal{B}$ between $Z_{(A_1,p)}$ and $Z_{(A_2,q)}$, such that the following hold,*

– $\forall (s,t) \in \mathcal{B}\ :\ s \in S_1, t \in S_2$, *$\mathcal{M}(s) \le \mathcal{M}(t)$ or*

– $\forall (s,t) \in \mathcal{B}\ :\ s \in S_1, t \in S_2,\ \mathcal{M}(s) \geq \mathcal{M}(t)$

i.e. in none of the bisimulation relations, all bisimilar nodes maintain the same relation (less or greater or equal) between their spans.

Definition 9. *Given a timed automaton A, let $Z_{(A,p)}$ be the zone valuation graph corresponding to process $p \in T(A)$. Let $p' \in T(A)$ be a process reachable from p and s be the node of $Z_{(A,p)}$ such that $p' \in \mathcal{G}(s)$. Let x be a critical clock of s and $v_{p'}(x)$ denote the valuation of clock x for process p'. We define maximum admissible delay for p' in s as $max_x(s) - v_{p'}(x)$.*

For example, for the process $\langle l_0, 3\rangle$ in 5, the maximum admissible delay is $5-3=2$.

Now we are ready to prove the decidability of checking prebisimilarity. Intuitively, the approach consists of checking the strong bisimilarity between two zone valuation graphs and simultaneously checking for the flip in delay between bisimilar nodes. If zone valuation graphs are strongly bisimilar and no flip in delay is found then the processes are prebisimilar, otherwise they are not prebisimilar. The following lemmas prove this intuition.

Lemma 1. *For $p \in T(A_1)$ and $q \in T(A_2)$, $FID(Z_{(A_1,p)}, Z_{(A_2,q)}) \wedge p \sim_u q \Rightarrow (p \not\precsim q \wedge q \not\precsim p)$*

Proof. $p \sim_u q$ and $FID(Z_{(A_1,p)}, Z_{(A_2,q)})$ imply that in each strong bisimulaton relation $\mathcal{B} \subseteq S_1 \times S_2$, such that $(s_{p_1}, s_{q_1}) \in \mathcal{B}$ and $(s_{p_2}, s_{q_2}) \in \mathcal{B}$, where $s_{p1}, s_{p2} \in Z_{(A_1,p)}$ and $s_{q1}, s_{q2} \in Z_{(A_2,q)}$, either

– $\mathcal{M}(s_{p1}) > \mathcal{M}(s_{q1})$ and $\mathcal{M}(s_{p2}) < \mathcal{M}(s_{q2})$, or
– $\mathcal{M}(s_{p1}) < \mathcal{M}(s_{q1})$ and $\mathcal{M}(s_{p2}) > \mathcal{M}(s_{q2})$.

We first show that $p \sim_u q$ and $\mathcal{M}(s_{p_1}) > \mathcal{M}(s_{q_1}) \Rightarrow p \not\precsim q$. Consider the process $p_1 \in \mathcal{G}(s_{p_1})$ reachable from p such that $v_{p_1}(x) = min_x(s_{p_1})$, where x is a critical clock of s_{p_1}. Consider the delay $p_1 \xrightarrow{\mathcal{M}(s_{p_1})} p'_1$. So $p'_1 \in \mathcal{G}(s_{p_1})$. There does not exist a $q_1 \in \mathcal{G}(s_{q_1})$ such that $q_1 \xrightarrow{d} q'_1$ where $d > \mathcal{M}(s_{p1})$ and $q'_1 \in \mathcal{G}(s_{q_1})$. This implies that there does not exist a process q_1 such that $p_1 \precsim q_1$ implying $p \not\precsim q$. Similarly, we can show that $\mathcal{M}(s_{p2}) < \mathcal{M}(s_{q2})$ implies $q \not\precsim p$.

For the second case too we can similarly show that neither $p \precsim q$ nor $q \precsim p$ holds. □

Lemma 2. *For $p \in T(A_1)$ and $q \in T(A_2)$, $p \sim_u q \wedge\ \neg FID(Z_{(A_1,p)}, Z_{(A_2,q)}) \Rightarrow\ p \precsim q \vee q \precsim p$.*

Proof. Without loss of generality, say in a strong bisimulation relation $\mathcal{B} \subseteq S_1 \times S_2$, span of all nodes in $Z_{(A_2,q)}$ is less than the span of their corresponding bisimilar nodes in $Z_{(A_1,p)}$. We show that this implies $q \precsim p$. Consider some node s_{q_1} of $Z_{(A_2,q)}$. Let s_{p_1} be a node of $Z_{(A_1,p)}$ such that $s_{p_1} \sim s_{q_1}$ and q_1 be any process in $\mathcal{G}(s_{q_1})$ reachable from q. We will show that there exists a process $p_1 \in \mathcal{G}(s_{p_1})$ reachable from p such that $q_1 \precsim p_1$.

Let $d_1 = v_{q_1}(x) - min_x(s_{q_1})$, where x is a critical clock of s_{q_1} and $d_2 = d_1 \times (\mathcal{M}(s_{p_1})/\mathcal{M}(s_{q_1}))$. If y is a critical clock of s_{p_1}, then we define $p_1 \in \mathcal{G}(s_{p_1})$ to be a process such that $v_{p_1}(y) = min_y(s_{p_1}) + d_2$. It is easy to see that there exists such a p_1 reachable from p and $q_1 \precsim p_1$.

Now consider a node s_{p_2} in $Z_{(A_1,p)}$. Let s_{q_2} be a node in $Z_{(A_2,q)}$ reachable from q such that $s_{q_2} \sim s_{p_2}$ and let $p_2 \in \mathcal{G}(s_{p_2})$ be a process reachable from p. Let $d_3 = v_{p_2}(z) - min_z(s_{p_2})$, where z is a critical clock of s_{p_2} and $d_4 = d_3 \times (\mathcal{M}(s_{q_2})/\mathcal{M}(s_{p_2}))$. If w be a critical clock of s_{q_2}, then we define $q_2 \in \mathcal{G}(s_{q_2})$ to be a process such that $v_{q_2}(w) = min_w(s_{q_2}) + d_4$. It is easy to see that there exists such a q_2 reachable from q such that $q_2 \precsim p_2$.

So we have proved that if there is a strong bisimulation relation in which the span of each node of $Z_{(A_2,q)}$ is less than the span of its corresponding bisimilar node of $Z_{(A_1,p)}$, then $q \precsim p$. Similarly, we can also show that if there exists a strong bisimulation relation in which if the span of each node of $Z_{(A_1,p)}$ is smaller than the span of its corresponding bisimilar node in $Z_{(A_2,q)}$ then $p \precsim q$. Thus for $p \in T(A_1)$ and $q \in T(A_2)$, $p \sim_u q \wedge \neg FID(Z_{(A_1,p)}, Z_{(A_2,q)}) \Rightarrow p \precsim q \vee q \precsim p$. □

Corollary 1. *For $p \in T(A_1)$ and $q \in T(A_2)$, $q \precsim p$ or $p \precsim q \Rightarrow p \sim_u q$ and $\neg FID(Z_{(A_1,p)}, Z_{(A_2,q)})$*

Proof. Immediate from the contrapositive of lemma 1. □

Theorem 1. *For $p \in T(A_1)$ and $q \in T(A_2)$, $q \precsim p$ or $p \precsim q \Leftrightarrow p \sim_u q$ and $\neg FID(Z_{(A_1,p)}, Z_{(A_2,q)})$*

Proof. From lemma 2 and corollary 1. □

5.3 Generating Zone Valuation Graph

Algorithm 1 describes the construction of the zone valuation graph for a timed automaton process. It uses a combination of both forward and backward traversal of the timed automata locations to create stable partitions of the zones. Initially the timed automaton is traversed so as to find max_x^l for each clock $x \in C$ and location $l \in L$. This step is used for abstraction purposes. The algorithm then splits a convex polyhedron [20] defined by all clocks with valuations $\mathbb{R}_{\geq 0}$ into multiple convex polyhedra according to the clock constraints in the timed automaton. To achieve this, the algorithm traverses the timed automaton in a breadth-first manner using the queue Q. Corresponding to each location l of the timed automaton, a height balanced sorted tree T_l is maintained that stores the locations whose zone splitting effect, based on a certain canonical decomposition of clock constraints as described below, have already been propagated to l. In other words, zones of l are stable with respect to the outgoing transitions from the locations present in T_l. Stability checking considers zones of those locations which are either in *preds* or in *succs* relation with zones in l. These relations are dynamically updated as edges are inserted between the nodes of the zone valuation graph. In the algorithm, l_p denotes the latest location added to T_l and the zones of l are split so as to make them *stable* with respect to the zones of l_p. A reachable location l is added to Q only if its zones are stable with respect to zones of all locations in T_{l_p}. Function $elements(L_l)$ returns the set of all elements present in L_l.

For each location l dequeued from Q, its zones are split based on a canonical decomposition of certain clock valuations. The canonical decomposition is obtained from the guards on the outgoing edges of l, the invariant on l and the invariant on the destination location corresponding to each outgoing transition. The splitting of zones of l is further propagated to zones of other locations in T_l. Each $l_j \in T_l$ is also added to the queue since the split of the zones of l_j can cause splits of zones of its descendants as well. The abstraction, if necessary, is done simultaneously during the creation of the zone. A location l_i which is a descendant or ancestor of location l is not added to the queue if its zones are stable with respect to the zones of all locations in T_l. This phase of the algorithm terminates when the queue becomes empty. While constructing zone valuation graph, after phase 1, we are left with a set of nodes in the zone valuation graph. The aim of the next phase is to identify the nodes that are strongly bisimilar to each other and then to combine them. Given a finite graph (labeled transition system), the Paige-Tarjan algorithm [19] produces the largest bisimulation relation. The nodes that are identified to be bisimilar are merged to produce a single node as long as the merged node preserves convexity. In the algorithm, to avoid clutter, we have not specifically mentioned how to add edges. Figure 5 (e) shows the zone valuation graph for the state $\langle l_0, 0\rangle$ corresponding to the timed automaton shown in figure 5 (a). All loops in the zone valuation graph have an implicit self loop labeled with ε.

Complexity: In the worst case, corresponding to the timed automaton for which region equivalence matches time abstracted bisimulation, the zone graph created is the same as the region graph. Hence the worst case complexity for zone graph creation is exponential in the number of clocks. We consider the complexity of zone graph creation in terms of a specific input automaton to be of particular interest and thus present the complexity in terms of the number of locations, transitions and clock variables of the automaton and the size of zone valuation graph created after abstraction.

For the purpose of abstraction, a preprocessing step is required to identify max_x^l for each clock $x \in C$ and each location $l \in L$. From [10], the complexity is $O(t^3)$ where $t = |C| \times n$ and n is the number of locations in the timed automaton. Since each node can be added to Q a maximum of n times, the total number of additions to and deletions from Q is bounded by $O(n^2)$. Let $|S|$ and m denote the number of zones and edges respectively in the zone valuation graph produced after abstraction. For each dequeue operation we list the sub-operations below and find their complexity. Let $l = dequeue(Q)$.

– For subsequent splits of zones of l, it is required to find the zones of the locations in T_l with respect to which, the zones of l are unstable. This can be done in $O(|E|)$ time.

Algorithm 1 Construction of Zone Valuation Graph
Input: Process $p \in T(A)$ and description of a timed automaton A
Output: Zone valuation graph corresponding to p

```
1: Calculate $max_x^l$ for each location $l \in L$ and each clock $x \in C$. This is required for
   abstraction to ensure finite number of zones in the zone graph.
2: Initialize $Q$ to an empty queue. For each location $l$, create a zone $\mathcal{N}$ with clock
   valuations for each clock $\mathbb{R}_{\geq 0}$.
3: Associate a height balanced tree $T_l$ with each location $l$ of $A$ and initialize $T_l$ to be
   empty.
4: Let $l_0$ be the initial location of $A$.
5: $Enqueue(Q, l_0)$.
6: while Q not empty do
7:     $l = dequeue(Q)$
8:     if $T_l$ is not empty then,
9:         Let $l_p$ be the latest location added to $T_l$. /* parent of $l$ in current path */
10:        Split current zones of $l$ in order to make them stable with respect to the
   zones of $l_p$.                                   /* Forward method */
11:        Abstract each of the newly created zones if necessary.
12:    end if
13:    if $l \notin T_l$ then
14:        Find the canonical decomposition of constraint based on the guards of the
   outgoing transitions of $l$ along with the invariant of $l$ and invariant of destination
   location of each transition.
15:        Split $l$ further based on this canonical decomposition mentioned above.
16:        Abstract each of the newly created zones if necessary.
17:        Add $l$ to $T_l$
18:    end if
19:    for all location $l_j \in T_l$ do
20:        If zones of $l_j$ are unstable with respect to zones of $l$, then split zones of $l_j$
   to make them stable with respect to zones of $l$.
21:        Abstract each of the newly created zones if necessary.
22:        if $l \notin T_{l_j}$ then Add $l$ to $T_{l_j}$
23:        end if
24:        if $elements(T_{l_j}) \backslash elements(T_l) \neq \emptyset$ then
25:            $Enqueue(Q, l_j)$ /* An ancestor is enqueued: backward method */
26:        end if
27:    end for
28:    for all successor $l_i$ of $l$ do /* Successor in timed automaton */
29:        if $l_i$ is reachable from any of the current set of zones of $l$ then
30:            if $l \notin T_{l_i}$ then Add $l$ to $T_{l_i}$
31:            end if
32:            if $elements(T_{l_i}) \backslash elements(T_l) \neq \emptyset$ then
33:                $Enqueue(Q, l_i)$ /* A successor is enqueued */
34:            end if
35:        end if
36:    end for
37: end while
38: Obtain canonical form of the zone valuation graph by merging time abstracted
   bisimilar nodes of the zone graph while preserving convexity. /* Phase 2 */
```

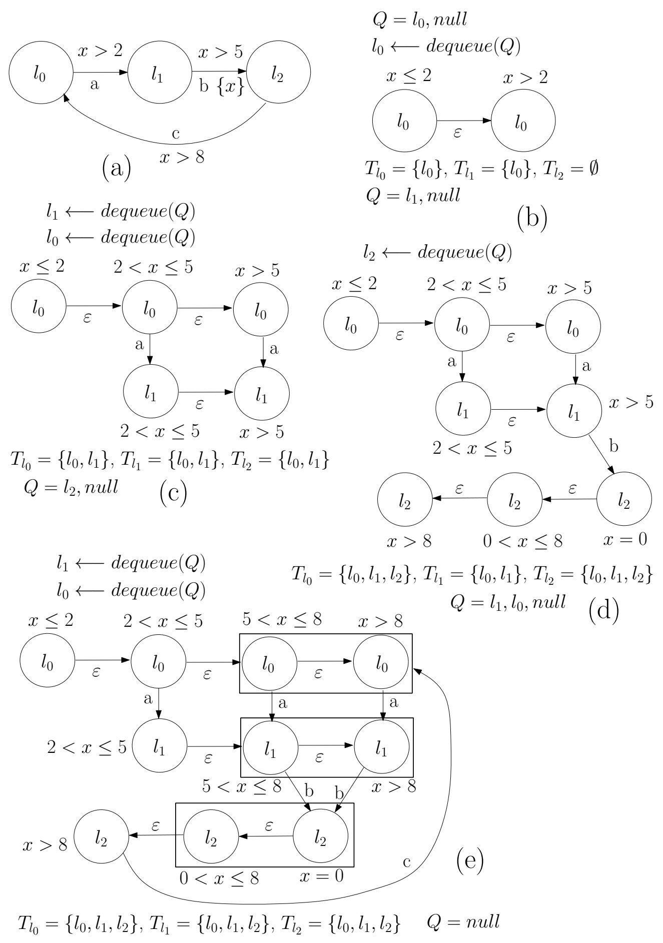
- For all locations in T_l, the number of zones that are unstable with respect to the zones of l and hence will be split are of order $O(|S|)$. Hence finding such zones, which will be split due to split of l based on the guards on outgoing edges of l, requires $O(|S| \times |C|)$ time, where the multiplier $|C|$ arises from the fact that each zone is defined by $|C|$ clocks. Abstraction of each zone also requires the same complexity.
- The complexity of split of l based on canonical decomposition is $O(|S| \times |C|)$.
- The number of locations that can be reached using the outgoing edges from l is of order $O(n)$. Before each such location l_i is enqueued to Q, its stability is checked with respect to the zones of all locations appearing in T_l which requires a set difference operation between $elements(T_{l_i})$ and $elements(T_l)$ that can be done in $O(n\,log\,n)$. If the corresponding zones are already stable with respect to zones belonging to all locations in T_l, then it is not added to Q. This condition along with the abstraction ensures termination of the zone graph construction procedure. As the number of locations in T_l is $O(n)$, for $O(n)$ operations this step has complexity of order $O(n^2 \times log\,n)$.

Corresponding to every dequeue operation, the cost incurred is $O(|E| + |S| \times |C| + n^2 \times log\,n) = O(|S| \times |C| + n^2 \times log\,n)$. Thus considering $O(n^2)$ dequeue operations, the total computation cost of phase 1 including the preprocessing phase to identify max_x^l for each clock $x \in C$ is $O(|S| \times |C| \times n^2 + n^4 \times log\,n)$.

Phase 2: Here we compute the complexity of the phase where the nodes in the zone graph obtained after phase 1 that are time abstracted bisimilar to each other are merged to produce the minimal zone graph preserving convexity of zones. We use Paige-Tarjan [19] partition refinement algorithm which has a cost of $O(|S| \times log\,m)$. This gives us the set of nodes that are bisimilar to each other and hence can be merged. The initial partition is decided in a way so as to ensure that nodes having non-convex zones are not merged.

Thus the total cost of computation for construction of the zone valuation graph is $O(n^2(|C|^3 n + |S| \times |C| + n^2 \times log\,n) + |S| \times log\,m)$.

An example of generating the zone valuation graph through stages: A timed automaton is shown in part (a) of figure 5. Parts (b) through (e) show steps in the generation of the corresponding zone valuation graph. The queue Q is initialized with location l_0, the initial location of the timed automaton. The zone valuation of location l_0 is split according to the canonical decomposition of the outgoing clock conditions of location l_0. After this split, T_{l_0} is initialized to $\{l_0\}$ and l_1 is enqueued to Q. Following a dequeue operation, l_1 zones are split to make it stable with respect to the zones of l_0. l_1 is also split according to the canonical decomposition of its outgoing transition. The effect of this split is again propagated to l_0 backwards which too is accordingly split. The process continues until the zones of the locations cannot be further split. We note here that the split happens in both forward and backward direction. The sequence of enqueue and dequeue operations for this particular example according to algorithm 1 are as follows:
$enqueue(l_0)$, $dequeue(l_0)$, $enqueue(l_1)$, $dequeue(l_1)$, $enqueue(l_0)$, $enqueue(l_2)$, $dequeue(l_0)$, $dequeue(l_2)$, $enqueue(l_1)$, $enqueue(l_0)$, $dequeue(l_1)$, $dequeue(l_0)$.



l_0 $\xrightarrow[a]{x > 2}$ l_1 $\xrightarrow[b\ \{x\}]{x > 5}$ l_2
c
$x > 8$
(a)
$Q = l_0, null$
$l_0 \longleftarrow dequeue(Q)$
$x \leq 2$
$x > 2$
ε
$T_{l_0} = \{l_0\}$, $T_{l_1} = \{l_0\}$, $T_{l_2} = \emptyset$
$Q = l_1, null$
(b)
$l_1 \longleftarrow dequeue(Q)$
$l_0 \longleftarrow dequeue(Q)$
$x \leq 2$
$2 < x \leq 5$
$x > 5$
$2 < x \leq 5$
$x > 5$
$T_{l_0} = \{l_0, l_1\}$, $T_{l_1} = \{l_0, l_1\}$, $T_{l_2} = \{l_0, l_1\}$
$Q = l_2, null$
(c)
$l_2 \longleftarrow dequeue(Q)$
$x \leq 2$
$2 < x \leq 5$
$x > 5$
$x > 5$
$2 < x \leq 5$
b
$x > 8$
$0 < x \leq 8$
$x = 0$
$T_{l_0} = \{l_0, l_1, l_2\}$, $T_{l_1} = \{l_0, l_1\}$, $T_{l_2} = \{l_0, l_1, l_2\}$
$Q = l_1, l_0, null$
(d)
$l_1 \longleftarrow dequeue(Q)$
$l_0 \longleftarrow dequeue(Q)$
$x \leq 2$
$2 < x \leq 5$
$5 < x \leq 8$
$x > 8$
$2 < x \leq 5$
$5 < x \leq 8$
$x > 8$
$x > 8$
$0 < x \leq 8$
$x = 0$
c
(e)
$T_{l_0} = \{l_0, l_1, l_2\}$, $T_{l_1} = \{l_0, l_1, l_2\}$, $T_{l_2} = \{l_0, l_1, l_2\}$ $Q = null$

Fig. 5. Timed automaton in part (a) and successive steps in creation of its zone valuation graph

The diagram shows the dequeue operations above each of the figures. Below each figure, the state of the queue is described after the operation shown in the figure gets completed. In part (e), the boxes enclose the strongly bisimilar zones that are combined in phase 2 to create the canonical zone valuation graph.

5.4 Algorithm: Deciding Timed Performance Prebisimulation Using Zone Valuation Graph

In algorithm 2, we present a method to decide if two states $p \in T(A)$ and $q \in T(B)$ are timed performance prebisimilar. After creating the zone valuation graphs for p and q, it invokes `checkTimedPrebisim` method with the initial nodes s_p and s_q of these zone valuation graphs and the binary comparison op-

Algorithm 2 Algorithm for deciding timed performance prebisimulation
Inputs: i) Two timed automata states p and q ii) description of the timed automata.
Output: Decision whether p and q are timed performance prebisimilar.

```
1: Create zone valuation graphs for $p$ and $q$
2: Let $s_p$ and $s_q$ be the initial nodes in these zone valuation graphs
3: if $\mathcal{M}(s_p) \leq \mathcal{M}(s_q)$ and checkTimedPrebisim($s_p$,$s_q$,$\leq$) then
4:     Declare $s_p \precsim s_q$
5: else
6:     if $\mathcal{M}(s_p) \geq \mathcal{M}(s_q)$ and checkTimedPrebisim($s_p$,$s_q$,$\geq$) then
7:         Declare $s_q \precsim s_p$
8:     else
9:         Declare $p$ and $q$ are *not* timed performance prebisimilar
10:    end if
11: end if

12: procedure checkTimedPrebisim($s_p$, $s_q$, $Rel$)
13:     if $(s_p, s_q) \in L_{pbisim}$ then return $true$
14:     end if
15:     if $(s_p, s_q) \in L_{notpbisim}$ then return $false$
16:     end if
17:     A := sort($s_p$); B := sort($s_q$);
18:     if $A \neq B$ then Add $(s_p, s_q)$ to $L_{notpbsim}$; return $false$
19:     end if
20:     Add $(s_p, s_q)$ to $L_{pbsim}$
21:     for all successors $s'_p$ of $s_p$ do
22:         if $s_p \overset{\alpha}{\rightarrow} s'_p$ then                    /* Here $\alpha \in Act \cup \{\varepsilon\}$ */
23:             r := false
24:             for all $s'_q$ such that $s_q \overset{\alpha}{\rightarrow} s'_q$ and $(\mathcal{M}(p')\ Rel\ \mathcal{M}(q'))$ do
25:                 $r$ := checkTimedPrebisim ($s'_p$, $s'_q$, $Rel$)
26:                 If r then  break; else continue EndIf
27:             end for
28:             if $not(r)$ then
29:                 Remove $(s_p, s_q)$ from $L_{pbsim}$; Add $(s_p, s_q)$ to $L_{notpbsim}$; return $false$
30:             else  continue
31:             end if
32:         end if
33:     end for
34:     for all successors $s'_q$ of $s_q$ do
35:         if $s_q \overset{\alpha}{\rightarrow} s'_q$ then
36:             r := false
37:             for all $s'_p$ such that $s_p \overset{\alpha}{\rightarrow} s'_p$ and $(\mathcal{M}(p')\ Rel\ \mathcal{M}(q'))$ do
38:                 $r$ := $r$ $\vee$ checkTimedPrebisim ($s'_p$, $s'_q$, $Rel$)
39:                 If r then  break; else continue EndIf
40:             end for
41:             if $not(r)$ then
42:                 Remove $(s_p, s_q)$ from $L_{pbsim}$; Add $(s_p, s_q)$ to $L_{notpbsim}$; return $false$
43:             else  continue
44:             end if
45:         end if
46:     end for
47:     return $true$
48: end procedure
```

erator *Rel*. Passing $\leq$ as *Rel* is equivalent to checking if $p \precsim q$ and similarly passing $\geq$ is equivalent to checking if $q \precsim p$. If both calls return false then the states p and q are declared not timed performance prebisimilar. If either of them returns true then the states p and q are declared timed performance prebisimilar. It can be shown that by passing $=$ operator as *Rel*, we can check timed bisimilarity between p and q [13]. The function `checkTimedPrebisim` checks strong bisimilarity between two nodes whose spans are in the relations *Rel* with each other. Function $sort(s_p)$ returns the set of actions that can be performed by the node s_p. Two search data structures L_{pbsim} and $L_{notpbsim}$ are maintained by this function. All pairs of nodes which have been proved not to be prebisimilar are kept in $L_{notpbsim}$. L_{pbsim} maintains all those pair of nodes that have been encountered so far and either have been assumed to be prebisimilar or have been proved to be prebisimilar. These data structures are used to avoid checking the same pair of nodes repeatedly for prebisimilarity. They also make sure that the function `checkTimedPrebisim` is invoked at most $n_1 n_2$ times where n_1 and n_2 represent the number of nodes in the zone valuation graphs of p and q respectively. Complexity of one such invocation, because of two nested for loops, is $O(m_1 m_2 |C|)$ where m_1 and m_2 represent the number of edges in the zone valuation graphs of p and q respectively. The factor for C appears because of the complexity of checking the span. Similarly the complexity of searching, inserting and removing the elements from search data structures L_{pbsim} and $L_{notpbsim}$ is bounded by $O(n_1 n_2 log(n_1 n_2))$. Therefore the total complexity of this algorithm is $O(n_1 n_2 m_1 m_2 |C|) \times O(n_1 n_2 log(n_1 n_2)) = O(n_1^2 n_2^2 . m_1 m_2 |C| log(n_1 n_2))$.

Theorem 2. *Algorithm 2 determines whether a timed performance prebisimulation relation exists between two processes p and q, i.e. if $p \precsim q$ or $q \precsim p$.*

Proof. The algorithm declares two processes to be performance prebisimilar if there exists a strong bisimulation relation between their zone valuation graphs with no flip in delay otherwise the algorithm declares them as not performance prebisimilar. The correctness of the algorithm thus follows from theorem 1. □

6 Conclusion and Future Work

In the technical report [13], we show how timed performance prebisimulation can be used for showing that one system is at least as fast as another. We consider two protocols for reliable data transfer, alternating bit protocol and Stop-and-Wait ARQ and show that alternating bit protocol is 'at least as fast as' stop-and-wait ARQ. Each protocol model consists of a sender, receiver and a lossy channel. Each of sender, channel and receiver is modeled with timed automata and both the systems consist of the parallel composition of the three. Using this prebisimulation relation, we formally prove that Alternating bit protocol is at least as fast as Stop-and-Wait ARQ. We cannot provide the details of the examples due to space constraint. The interested reader is referred to [13] for details of modeling of the two systems using timed automata.

Though bisimulation is a standard notion of program equivalence, prebisimulations have not been studied much in the literature on timed automata. However we do believe that "faster than" preorders are important in real time systems where "responsiveness" is an important aspect of the specification and in the implementation. In this paper, we have defined timed performance prebisimulation and proved its decidability using zone valuation graph for two timed automata states. We also show the use of *zone valuation graph* for checking timed bisimilarity between two processes. In future, we plan to define a weaker prebisimulation relation in which one state can be defined to be at least as fast as the other state if the time elapsed is compared over sequence of actions instead of comparing delays at every stage as in timed performance prebisimulation. We also look forward to investigate the closure properties of these relations.

Acknowledgements.

We would like to thank the anonymous referees for their constructive comments and suggestions. We also thank Stavros Tripakis for pointing us to many of the state of the art works that are related to our contribution. This research was partly sponsored by EADS and partly by Microsoft Research India Travel Grants.

References

1. L. Aceto, A. Ingólfsdóttir, K.J. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, 2007.
2. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
3. S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 1992.
4. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. *In Lecture Notes on Concurrency and Petri Nets*, 3098:87–124, 2004.
5. Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods System Design*, 24(3):281–320, May 2004.
6. K. Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *Proceedings of the 4th International Workshop on Computer Aided Verification*, volume 663, pages 302–315, Montreal, Canada, June 1992. Springer-Verlag.
7. F. Corradini, R. Gorrieri, and M. Roccetti. Performance preorder: Ordering processes with respect to speed. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science.* Springer-Verlag, 1995.
8. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Proc of the 4th Intl Conf on Tools and Algorithms for Construction and Analysis of Systems*, pages 313–329. Springer-Verlag, 1998.
9. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the international workshop on Automatic verification methods for finite state systems.* Springer-Verlag, 1990.
10. E. Fleury G. Behrmann, P. Bouyer and K. G. Larsen. Static guard analysis in timed automata verification. In *Proceedings of the 9th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'03, pages 254–270, Berlin, Heidelberg, 2003. Springer-Verlag.

11. K. G. Larsen G. Behrmann, P. Bouyer and R. Pelanek. Lower and upper bounds in zone-based abstractions of timed automata. *Int. J. Softw. Tools Technol. Transf.*, 8:204–215, June 2006.
12. M. Geilen, S. Tripakis, and M. Wiggers. The earlier the better: a theory of timed actor interfaces. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 23–32, 2011.
13. S. Guha, C. Narayan, and S. Arun-Kumar. On decidability of prebisimulation for timed automata. *http://www.cse.iitd.ernet.in/ shibashis/webpage/prebisim.pdf, Technical Report, Indian Institute of Technology Delhi, New Delhi, India*, 2012.
14. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
15. Ichiro I. Satoh and M. Tokoro. A formalism for remotely interacting processes. In *Proceedings of the International Workshop on Theory and Practice of Parallel Programming*. Springer-Verlag, 1995.
16. K. G. Larsen and W. Yi. Time abstracted bisimulation: implicit specifications and decidability. In *Proc of the 9th Intl Conf on the Mathematical Foundations of Programming Semantics*, volume 802, pages 160–176. Springer-Verlag, April 1994.
17. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
18. F. Moller and C. Tofts. Relating processes with respect to speed. In *Proceedings of CONCUR'91*, volume 527. Springer-Verlag, August 1991.
19. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
20. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18:25–68, 2001.
21. C. Weise and D. Lenzkes. Efficient scaling-invariant checking of timed bisimulation. In *Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science*, volume 1200, pages 177–188, Lübeck, Germany, 1997. Springer, Berlin.
22. M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. *Form. Methods Syst. Des.*, 11:113–136, August 1997.