Model Checking Prioritized Timed Automata

Shang-Wei Lin, Pao-Ann Hsiung, Chun-Hsian Huang, and Yean-Ru Chen

Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi, Taiwan−621, ROC
hpa@computer.org

Abstract. Priorities are often used to resolve conflicts in timed systems. However, priorities are not directly supported by state-of-art model checkers. Often, a designer has to either abstract the priorities leading to a high degree of non-determinism or model the priorities using existing primitives. In this work, it is shown how prioritized timed automata can make modelling prioritized timed systems easier through the support for priority specification and model checking. The verification of prioritized timed automata requires a subtraction operation to be performed on two clock zones, represented by DBMs, for which we propose an algorithm to generate the minimal number of zones partitioned. After the application of a series of DBM subtraction operations, the number of zones generated become large. We thus propose an algorithm to reduce the final number of zones partitioned by merging some of them. A typical bus arbitration example is used to illustrate the benefits of the proposed algorithms. Due to the support for prioritization and zone reduction, we observe that there is a 50% reduction in the number of modes and 44% reduction in the number of transitions.

Keywords: Prioritized timed automata, DBM subtraction, zone merging, zone reduction.

1 Introduction

Concurrency results in conflicts when resources are shared such as two or more processes trying to use the same processor or the same peripheral device in real-time embedded systems. To resolve such conflicts, scheduling, synchronization, and arbitration are some well-known solutions that have been popularly used in operating systems and in hardware designs. A common artifact of these solutions is the prioritization of contending parties. A low priority process is allowed to execute only when all processes with higher priorities are disabled.

System models used for design and verification such as timed automata, statecharts, and others allow non-determinisms which arise out of concurrency, interleaving, and information hiding. However, non-determinisms often result in unmanageably large state-spaces. Prioritization of transitions not only models real systems more accurately but also removes non-determinisms and thus reduces the size of state-spaces. Several modeling frameworks have been proposed for modeling and designing systems with priorities. However, their verification techniques are still very limited. All the above mentioned reasons have motivated us to model check timed systems with priorities.

D.A. Peled and Y.-K. Tsay (Eds.): ATVA 2005, LNCS 3707, pp. 370–384, 2005.

The target model for prioritization in this work will be timed automata (TA) [4], because it is widely used in most model checkers for real-time systems such as SGM [15,22], RED [21], UPPAAL [6], and Kronos [23]. In model checking timed automata with priorities, a subtraction operation between two clock zones, represented by *Difference Bound Matrices* (DBMs), is required. We have proposed an algorithm to perform the subtraction operation on two DBMs [14]. This algorithm generates the minimal number of zones partitioned. If there are multiple priorities in timed automata, a series of DBM subtraction operations are needed. After a series of DBM subtraction operations, the final number of zones partitioned may become very large resulting in a significant increase in state-space sizes. In this work, we propose another DBM subtraction operation that also generates the minimal number of zones partitioned. For handling the large number of zones generated after a series of DBM subtraction, we also propose a DBM merging algorithm to reduce the final number of zones partitioned by merging some of the zones.

The remaining portion is organized as follows. Section 2 describes previous work related to priority modeling and verification. Basic definitions used in our work are given in Section 3. Section 4 will formulate the solutions for solving the above described issues in prioritizing timed automata and then verifying them. Section 5 gives an example. The article is concluded and future research directions are given in Section 6.

2 Related Work

Several work of Joseph Sifakis [1,2,13,20,19] have focused on modeling timed systems with priorities. A solid theoretical basis has been laid by these work for modeling schedulers based on priorities. Several well-known scheduling methods such FIFO, rate-monotonic, earliest deadline first, least laxity first, priority ceiling protocol were modeled by priority rules. Deadlock-free controllers were also synthesized to meet safety properties expressed as priority rules [13]. A real-time process with arrival time, execution time, and period or deadline was formally modeled using different time urgencies such as *delayable* (must transit before the transition condition expires) and *eager* (must transit as soon as the transition condition holds) [2]. The compositionality of priorities was handled by priority choice operators [19]. In spite of these solid work on modeling and synthesis of schedulers for timed systems with priorities, little has been investigated on the verification of such systems.

Priorities have also been added to other modeling formalisms such as the work on a priority language for Timed CSP [16], in which some operators were refined into biased ones and several algebraic laws were given. Actions in process algebra have been prioritized by Cleaveland and Hennessy [11]. A prioritized version of ACP was proposed by Baeten et al. [5]. Camilleri proposed a prioritized version of CCS [17] with a left biased choice operator [8]. Priorities have not received as much focus in the model checking field as that in process algebra. The work here is an initial step in the verification direction. Some background on the model checking paradigm is given in the rest of this Section.

Model checking [9,10,18] is a technique for verifying finite state concurrent systems. One benefit of this restriction is that verification can be performed automatically.

The procedure normally uses an exhaustive search of the state space of a system to determine if some specification is true or not. Given sufficient resources, the procedure will always terminate with a *yes/no* answer. Moreover, it can be implemented by algorithms with reasonable efficiency, which can be run on moderate-sized machines. The process of model checking includes three parts: modeling, specification, and verification. *Modeling* is to convert a design into a formalism accepted by a model checking tool. Before verification, *specification*, which is usually given in some logical formalism, is necessary to state the properties that the design must satisfy. The *verification* is completely automated. However, in practice it often involves human assistance. One such manual activity is the analysis of the verification results. In case of a negative result, the user is often provided with an error trace. This can be used as a counterexample for the checked property and can help the designer in tracking down where the error occurred. In this case, analyzing the error trace may require a modification to the system and a re-application of the model checking algorithm.

When model checking is applied to real-time system verification, the model checker verifies if a system modeled by a set of concurrent *Timed Automata* (TA) satisfies a set of user-given specification properties expressed in the *Timed Computation Tree Logic* (TCTL). TA [3,4] is a timed extension of the conventional automata, which was proposed by Alur, Courcoubetis, and Dill in 1990 . TCTL [3] is a *timed* extension of the temporal logic called *Computation Tree Logic* (CTL) [9].

Our model checking procedures for prioritized timed automata are implemented in the *State-Graph Manipulators* (SGM) model checker [15,22], which is a high-level compositional model checker for real-time systems. Now, with the enhancement of prioritizations, SGM can also be used to model check real-time embedded systems such as *System-on-Chip* (SoC) architectures.

3 Preliminaries

Definition 1. **Mode Predicate**
Given a set C of clock variables and a set D of discrete variables, the syntax of a *mode predicate* η over C and D is defined as: $\eta := \mathit{false} \mid x \sim c \mid x - y \sim c \mid d \sim c \mid \eta_1 \wedge \eta_2 \mid \neg\beta_3$, where $x, y \in C$, $\sim \in \{\leq, <, =, \geq, >\}$, $c \in \mathcal{N}$, the set of non-negative integers, $d \in D$, η_1, η_2 are mode predicates, and β_3 is a discrete variable constraint. A mode predicate η can be expressed as a conjunction of a clock constraint ζ and a Boolean condition β on the discrete variables, that is, $\eta = \zeta \wedge \beta$. □

Let $B(C, D)$ be the set of all mode predicates over C and D. We extend the conventional definition of TA by prioritizing some of the transitions in a TA, as defined in Definition 2.

Definition 2. **Prioritized Timed Automaton**
A *Prioritized Timed Automaton* (PTA) is a tuple $\mathcal{A}_i = (M_i, m_i^0, C_i, D_i, L_i, \chi_i, T_i, \lambda_i, \tau_i, \pi_i, \rho_i)$ such that:

- M_i is a finite set of modes,
- $m_i^0 \in M$ is the initial mode,

- C_i is a set of clock variables,
- D_i is a set of discrete variables,
- L_i is a set of synchronization labels, and $\epsilon \in L_i$ is a special label that represents asynchronous behavior (i.e. no need of synchronization),
- $\chi_i : M_i \mapsto B(C_i, D_i)$ is an *invariance* function that labels each mode with a condition true in that mode,
- $T_i \subseteq M_i \times M_i$ is a set of transitions,
- $\lambda_i : T_i \mapsto L_i$ associates a synchronization label with a transition,
- $\tau_i : T_i \mapsto B(C_i, D_i)$ defines the transition triggering conditions,
- $\pi_i : T_i \mapsto \mathcal{N}$ associates an integer priority with a transition, where a larger positive value implies higher priority and a zero value implies no prioritization, and
- $\rho_i : T_i \mapsto 2^{C_i \cup (D_i \times \mathcal{N})}$ is an *assignment* function that maps each transition to a set of assignments such as resetting some clock variables and setting some discrete variables to specific integer values. □

A system state space is represented by a *system state graph* as defined in Definition 3.

Definition 3. Prioritized System State Graph

Given a system $\mathcal{S}$ with n components modelled by PTA $\mathcal{A}_i = (M_i, m_i^0, C_i, D_i, L_i, \chi_i, T_i, \lambda_i, \tau_i, \pi_i, \rho_i)$, $1 \leq i \leq n$, the system model is defined as a state graph represented by $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n = \mathcal{A}_\mathcal{S} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \pi, \rho)$, where:

- $M = M_1 \times M_2 \times \ldots \times M_n$ is a finite set of system modes, $m = m_1.m_2.\ldots.m_n \in M$,
- $m^0 = m_1^0.m_2^0.\ldots.m_n^0 \in M$ is the initial system mode,
- $C = \bigcup_i C_i$ is the union of all sets of clock variables in the system,
- $D = \bigcup_i D_i$ is the union of all sets of discrete variables in the system,
- $L = \bigcup_i L_i$ is the union of all sets of synchronization labels in the system,
- $\chi : M \mapsto B(\bigcup_i C_i, \bigcup_i D_i)$, $\chi(m) = \wedge_i \chi_i(m_i)$, where $m = m_1.m_2.\ldots.m_n \in M$.
- $T \subseteq M \times M$ is a set of system transitions which consists of two types of transitions:
  - *Asynchronous transitions*: $\exists i, 1 \leq i \leq n, e_i \in T_i$ such that $e_i = e \in T$
  - *Synchronized transitions*: $\exists i, j, 1 \leq i \neq j \leq n, e_i \in T_i, e_j \in T_j$ such that $\lambda_i(e_i) = (l, in)$, $\lambda_j(e_j) = (l, out)$, $l \in L_i \cap L_j \neq \emptyset$, $e \in T$ is synchronization of e_i and e_j with conjuncted triggering conditions and union of all transitions assignments (defined later in this definition)
- $\lambda : T \mapsto L$ associates a synchronization label with a transition, which represents a blocking signal that was synchronized, except for $\epsilon \in L$, ϵ is a special label that represents asynchronous behavior (i.e. no need of synchronization),
- $\tau : T \mapsto B(\bigcup_i C_i, \bigcup_i D_i)$, $\tau(e) = \tau_i(e_i)$ for an asynchronous transition and $\tau(e) = \tau_i(e_i) \wedge \tau_j(e_j)$ for a synchronous transition,
- $\pi : T \mapsto \mathcal{N}$, where $\pi(e) = \pi_i(e_i)$ for an asynchronous transition and $\pi(e) = \max\{\pi_i(e_i), \pi_j(e_j)\}$ for a synchronous transition, and
- $\rho : T \mapsto 2^{\bigcup_i C_i \cup (\bigcup_i D_i \times \mathcal{N})}$, $\rho(e) = \rho_i(e_i)$ for an asynchronous transition and $\rho(e) = \rho_i(e_i) \cup \rho_j(e_j)$ for a synchronous transition. □

In a mode predicate, there are Boolean and clock constraints. In most model checkers, the Boolean constraints are represented by *Binary Decision Diagrams* (BDD) [7] proposed by Bryant and the clock constraints are represented by *Difference Bound Matrices* (DBM) [12] proposed by Dill.

4 Model Checking Real-Time Embedded Systems

Our target problem is to model and verify real-time embedded systems with priority. A set of prioritized timed automata is used to model such a system and model checking is used to verify if the prioritized system state graph, obtained by merging the set of PTA, satisfies user-given TCTL properties. In this section, we will propose solutions to the issues that were introduced in Section 1. A precise definition of the semantics of prioritized timed automata will be given in Section 4.1. A major extension to the conventional semantics involves the negation of clock zones and its implementation using DBMs, which will be covered in Section 4.2.

4.1 Semantics of Prioritized Timed Automata

The syntax of priorities was given as non-negative integers associated with transitions such that a larger positive value implied higher priority, as defined in Definition 2. Besides an integer priority, a transition $t \in T_i$ also has a triggering condition $\tau_i(t) \in B(C_i, D_i)$, which, being a mode predicate (Definition 1), can be segregated into two parts, namely a clock constraint (or clock zone) $\zeta(t)$ and a Boolean condition $\beta(t)$ such that $\tau_i(t) = \zeta(t) \wedge \beta(t)$. Prioritization semantics require the negation of transition triggering conditions because a transition t can be executed only if all transitions t' with priorities higher than that of t cannot be executed, that is, they are disabled or their triggering conditions $\tau_i(t')$ do not hold. In other words, transition t can fire only when the following condition holds.

$$\tau_i(t) \wedge \left(\bigwedge_{\pi_i(t') > \pi_i(t)} \neg\tau_i(t') \right), \tag{1}$$

where transitions t and t' all originate from the same source mode.

Given a triggering condition $\tau_i(t) = \zeta(t) \wedge \beta(t)$, its negation is defined as follows.

$$\begin{aligned}
\neg\tau_i(t) &= \overline{\zeta(t)} \vee \neg\beta(t) \\
\overline{\zeta(t)} &= x \sim' c, \quad \text{if } \zeta(t) = x \sim c, \\
&= x - y \sim c, \quad \text{if } \zeta(t) = x - y \sim c, \\
&= \overline{\zeta_1} \wedge \overline{\zeta_2}, \quad \text{if } \zeta(t) = \zeta_1 \wedge \zeta_2, \\
\neg\beta(t) &= d \sim' c, \quad \text{if } \beta(t) = d \sim c \\
&= \neg\beta_1 \vee \neg\beta_2, \quad \text{if } \beta(t) = \beta_1 \wedge \beta_2, \\
&= \beta_1, \quad \text{if } \beta(t) = \neg\beta_1,
\end{aligned} \tag{2}$$

where $x, y \in C_i$, $d \in D_i$, $c \in \mathcal{N}$, $\sim' \in \{>, \geq, \neq, \leq, <\}$ corresponding respectively to $\sim \in \{\leq, <, =, >, \geq\}$, ζ_1, ζ_2 are clock constraints, and β_1, β_2 are Boolean conditions on discrete variables in D_i.

In the above definition for negation of a triggering condition in $B(C_i, D_i)$, the result of negation no longer belongs to the set $B(C_i, D_i)$, that is, the set of mode predicates is not closed under the negation operator. This is because all clock constraints, also called *clock zones*, in $B(C_i, D_i)$ are n-dimensional convex polyhedra for a system with n clocks. However, the result of negation may not be *convex*. This non-closure of negation has adverse effects on model checking because all operators on transition triggers and mode invariants need to guarantee closure so that the timed automata can be composed into state-graphs for model checking. Closure is also required to guarantee termination of the composition procedures for timed automata.

Conventional operators on clock zones such as *intersection*, *time elapse*, and *reset* all guarantee closure as their results are still clock zones (convex polyhedra). Nevertheless, the possibly non-convex polyhedron generated by negation can be converted into a set of convex polyhedra. In Section 4.2, we propose an algorithm for the optimal partitioning of non-convex polyhedra so that priorities can be modeled and verified for real-time embedded systems. Here, optimality means the least number of convex polyhedra is generated.

Returning to the condition for execution of a low priority transition t given in Equation (1), it can be expressed now more precisely as the following.

$$\beta_i(t) \wedge \zeta_i(t) \wedge \left[\left(\bigwedge_{\pi_i(t') > \pi_i(t)} \neg \beta_i(t') \right) \vee \left(\bigwedge_{\pi_i(t') > \pi_i(t)} \overline{\zeta_i(t')} \right) \right] \tag{3}$$

4.2 Optimal DBM Subtraction Algorithm

A *clock zone* is a clock constraint consisting of conjunctions of $x \sim c$ and $x - y \sim c$, where x, y are clock variables in C_i and $c \in \mathcal{N}$. A clock zone is also restricted to be a convex polyhedron. It is often implemented as a *Difference Bound Matrix* (DBM) [12], which is defined as follows.

Definition 4. **Difference Bound Matrix (DBM)**
Given a clock zone z that represents clock constraints on n clocks in $C_i = \{x_1, x_2, \ldots, x_n\}$, it can be implemented as a $(n + 1) \times (n + 1)$ matrix D, where the element $D(i, j) = \sim c, \sim \in \{<, \leq\}, c \in \mathcal{N}$, represents the constraint $x_i - x_j \sim c, 0 \leq i, j \leq n$. It is assumed $x_0 = 0$. □

Geometrically, a clock zone over n clocks is an n-dimensional convex polyhedron. In the model checking of timed automata, three operations on clock zones are required, namely intersection, time elapse, and reset. The set of clock zones is closed under these three operations.

Definition 5. **Unbounded and Bounded Element in DBM**
Given a DBM D on n clocks, an element $D(i, j)$ of D is *unbounded* if $D(i, j) = "< \infty"$, where $i \neq j$; otherwise we call $D(i, j)$ *bounded*, where $i \neq j$. □

Definition 6. **Unrestricted DBM**
Given a DBM D on n clocks, we call D *unrestricted*, if it satisfies the following: $D(i, j)$ is *unbounded* for all $0 \leq i, j \leq n$. □

Definition 7. Complement Clock Constraint
Given a clock constraint $c_1 = x - y \sim c$ (or $x \sim c$), the *Complement Clock Constraint* $\overline{c}_1$ of c_1 is defined as : $\overline{c}_1 = x - y \sim' c$ (or $x \sim' c$), where $x, y \in C$, $c \in \mathcal{N}$, $\sim' \in \{\leq, <, \neq, \geq, >\}$ corresponding respectively to $\sim \in \{>, \geq, =, <, \leq\}$. We can call that c_1 and $\overline{c}_1$ are *complement or* $\overline{c}_1$ *is complement to* c_1. □

Definition 8. Reduced DBM
Given a DBM D on n clocks representing a clock zone z, D is *reduced* if D has the minimal number of *bounded* elements in all of the DBMs representing the same clock zone z. □

In the verification of real-time embedded systems with priority, we need to subtract the clock zone representing the time a higher priority transition is enabled (trigger satisfied) from the clock zone representing the time a lower priority transition is enabled. We will call this the *subtraction* operator. In Section 4.1, for a given clock zone ζ, Equations (1, 2, 3) defined the negation of clock zones $\overline{\zeta}$. Using this negation, given two clock zones z_1 and z_2, we can calculate their difference as follows.

$$z_1 - z_2 = z_1 \cap \overline{z_2} \tag{4}$$

where $\cap$ is the intersection operator between two zones.

As mentioned before in Section 4.1, $\overline{z_2}$ may not be a clock zone anymore as it may not be convex. We propose an algorithm here so that we can partition the possibly non-convex polyhedron $\overline{z_2}$ into a set of convex polyhedra. For ease of illustration, we will focus on the 2-dimensional case. It can be easily extended to n-dimensional zones, for any $n > 2$.

The optimal DBM subtraction algorithm, with $O(n^4)$ complexity for n clocks, is given in Algorithm 1 and described as follows. Given two clock zones represented by DBM z_1 and z_2, we can obtain the difference $z_1 - z_2$ by the following steps.

- Find $z_{intersect} = z_1 \cap z_2$ (Step 2)
- Reduce $z_{intersect}$ to obtain a DBM with minimal bounded elements so as to generate the minimal number of partitions (zones). (Step 3)
- Let z_{remain} record the remainder unpartitioned zone, which is initially assigned as z_1 (Step 4)
- For each non-diagonal and bounded element in the DBM $z_{intersect}$, we obtain a corresponding complement zone by reversing the relational operator. (Steps 5–7)
- For each complement zone, we find its intersection with z_{remain}, denoted as z_{tmp}, which is a zone. (Step 8)
- If z_{tmp} is not NULL, it means that we have subtracted a zone from z_{remain}. Then z_{tmp} is included into the set Z. (Step 10)
- After that, we have to recompute the remainder zone z_{remain}. (Step 11–13)

We can prove that a minimal number of partitions (zones) are generated after subtraction. Before we prove this, we give Theorem 1 as follows.

Theorem 1. *Given two DBMs namely* z_1 *and* z_2*, and* $z_{intersect} = z_1 \cap z_2$*, if the reduced DBM of* $z_{intersect}$ *has* m *unbounded elements, then* $z_1 - z_2$ *will generate at least m zones(DBMs).*

input : DBM: z_1, z_2
output: DBM*: Z //set of DBMs
DBM*: $z_{intersect}, z_{tmp}, z_{remain}$

$z_{intersect} \leftarrow z_2 \cap z_1$;
REDUCE($z_{intersect}$);
$z_{remain} \leftarrow z_1$;
for *each* $z_{intersect}(i,j)$ *and* $i \neq j$ *and* $z_{intersect}(i,j)$ *is bounded* **do**
  INIT(z_{tmp}); //set z_{tmp} *unrestricted*
  $z_{tmp}(i,j) \leftarrow\sim' c$; //$z_{intersect}(i,j) = '' \sim c''$,$\sim' \in \{>, \geq\}$ for $\sim \in \{\leq, <\}$, respectively
  $z_{tmp} \leftarrow z_{tmp} \cap z_{remain}$
  if $z_{tmp} \neq NULL$ **then**
    $Z \leftarrow Z \cup \{z_{tmp}\}$
    INIT(z_{tmp}); //set z_{tmp} *unrestricted*
    $z_{tmp}(i,j) \leftarrow\sim c$; //$z_{intersect}(i,j) = '' \sim c''$
    $z_{remain} \leftarrow z_{remain} \cap z_{tmp}$;
  end
end
return Z

Algorithm 1: DBM Subtraction $z_1 - z_2$

Proof. Let $S = \{S_1, S_2, ..., S_n\}$ be the set of DBMs generated by $z_1 - z_2$, and $\mid S \mid$ is n. Let $\{c_1, c_2, ..., c_m\}$ be the m clock constraints corresponding to the m unbounded elements of $z_{intersect}$. Let $\{\overline{c}_1, \overline{c}_2, ..., \overline{c_m}\}$ be the m complement clock constraints with respect to $\{c_1, c_2, ..., c_m\}$. Because we want to subtract $z_{intersect}$ from z_1, therefore $\overline{c}_i \in S_j$, where $1 \leq i \leq m$, for some $j \in \{1, 2, ..., n\}$. Given every two element $\overline{c}_p$ and $\overline{c}_q \in \{\overline{c}_1, \overline{c}_2, ..., \overline{c}_m\}$, $\overline{c}_p$ and $\overline{c}_q$ will not belong to the same S_j for some $j \in \{1, 2, ..., n\}$, otherwise S_j will not be a zone (if so, S_j will not be a convex). So, $n \geq m$, that is, $z_1 - z_2$ will generate at least m zones (DBMs).

Theorem 2. *The number of zones generated by the DBM subtraction algorithm is minimal.*

Proof. Let z_1 and z_2 are two DBMs, and we want to do $z_1 - z_2$. Let $z_{intersect} = z_1 \cap z_2$. Let m be the number of bounded elements of the reduced DBM of $z_{intersect}$. By Theorem 1, we know that the number of zones generated by $z_1 - z_2$ is at least m. By Algorithm 1, we use each bound element of the reduced DBM of $z_{intersect} \Rightarrow$ the number of zones generated by Algorithm 1 is at most m. So, Algorithm 1 generates the minimal number of zones of $z_1 - z_2$.

Consider the example shown in Figure 1, we want to subtract the smaller zone from the bigger zone. Figure 2 shows the snap shots of Algorithm 1 operating on this example.

4.3 DBM Merging Algorithm

From Theorem 1 and Theorem 2, we know that we can obtain the minimal number of zones using Algorithm 1 when we want to subtract z_2 from z_1. Algorithm 1 operating on

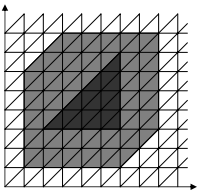

Fig. 1. An Example of DBM Subtraction

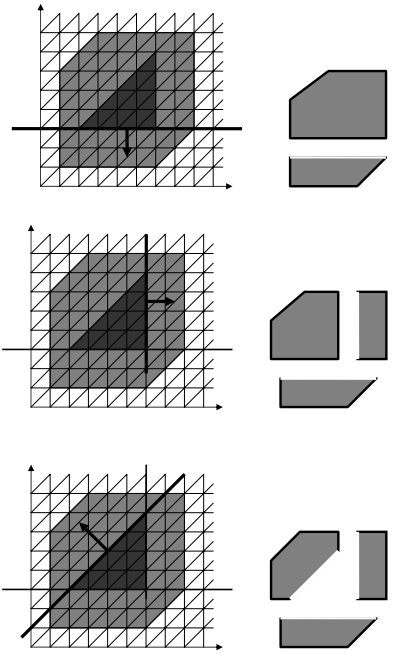

Fig. 2. Steps of DBM Subtraction

two zones is optimal, but if we want to do $z_1 - z_2 - z_3$, the number of zones generated may not be optimal. In other words, there may exist more than two zones which can be merged into a zone in the zone set after $z_1 - z_2 - z_3$.

In order to obtain the less number of zones after a series of zone subtraction operations, we propose an algorithm, which can help to determine whether given two zones z_1 and z_2 are mergeable. If not, the algorithm will return an unrestricted DBM. If yes, the algorithm will return a DBM representing $z_1 \cup z_2$. Note that all zones in the zone set S, after a series of subtraction operations, are disjoint. The input zones, say z_1 and z_2, of the algorithm have one restriction, which is that, z_1 cannot intersect with z_2. The DBM merging algorithm given in Algorithm 2 has $O(n^3)$ complexity and is described as follows. Given two clock zones represented by DBM z_1 and z_2 on n clocks, we can obtain the DBM z generated by merging z_1 and z_2 by the following steps.

- Reduce z_1 and z_2 (Steps 5–6)
- Set z unrestricted (Step 7)
- See if z_1 and z_2 are mergeable. If z_1 and z_2 are mergeable, then there exists a clock constraint $z_1(i,j)$, whose complement constraint is $z_2(j,i)$, for some $i,j \in \{1,2,...,n\}$. If there is no $z_1(i,j)$, whose complement constraint is the same as $z_2(i,j)$ for some $i,j \in \{1,2,...,n\}$, then z_1 and z_2 are not mergeable and the algorithm terminates. If yes, z_1 and z_2 are possibly mergeable. (Steps 8–17)
- If a pair of complement constraints is found in Steps 8–17 when $i = m$ and $j = n$, then $z(i,j)$ will be set as $z_2(i,j)$ and $z(j,i)$ will be set as $z_1(j,i)$. (Step 11–14)
- Besides the complement constraints found in Steps 8-17, if every corresponding bounded element pair $(z_1(i,j), z_2(i,j))$ is equal for all $i,j \in \{1,2,...,n\}$, then z_1 and z_2 are mergeable; otherwise, z_1 and z_2 are not mergeable. If z_1 and z_2 are mergeable,we can obtain $z(i,j)$ from bounded $z_1(i,j)$ or $z_2(i,j)$. (Step 22–38)

After illustrating how Algorithm 2 works on two zones, we will now show how to use Algorithm 2 to merge several zones generated by a series of zone subtraction operations on a zone. Theorem 3 tells us that given a zone set $Z = \{z_1, z_2, ..., z_n\}$, which consists of n clock zones, if $(z_1 \cup z_2 \cup ... \cup z_n)$ is a zone, then there exist two zones z_i and z_j which are mergeable for some $i,j \in \{1,2,...,n\}$, that is, there exists a merge sequence merging two zones at a time, which can merge $\{z_1, z_2, ..., z_n\}$ into Z.

We propose an algorithm which helps to merge a zone set $Z = \{z_1, z_2, ..., z_n\}$ into another zone set $Z' = \{z'_1, z'_2, ..., z'_m\}$, for $m \leq n$. This algorithm uses Algorithm 2 as a subroutine. The DBM set merging algorithm is given in Algorithm 3 and described as follows.

- Set Z' an empty set (Step 1)
- Merge $z_i \in Z$ with each other zone z_j , $i \neq j$, $i,j \in \{1,2,...,n\}$ and modify z_i into the output zone of DBM-Merge(z_i, z_j). (Step 3–9)
- Add z_i to Z' (Step 10)
- Do the above steps iteratively until every z_i has been considered. (Step 2–11)

Theorem 3. *Given a zone set $Z = \{z_1, z_2, ..., z_n\}$ and $\mid Z \mid = n$. If no pair of zones z_i, $z_j \in Z$ is mergeable, then $(z_1 \cup z_2 \cup ... \cup z_n)$ is not a zone.*

Proof. Let us prove this using a contradiction. We assume that $(z_1 \cup z_2 \cup ... \cup z_n)$ is a zone. Thus there must exist a zone $z_i \in Z$, such that $(z_1 \cup z_2 \cup \ldots \cup z_{i-1} \cup z_{i+1} \cup ... \cup z_n)$ is a zone for some $i \in \{1,2,...,n\}$, and of course z_i and $(z_1 \cup z_2 \cup ... \cup z_{i-1} \cup z_{i+1} \cup ... \cup z_n)$ are mergeable. Otherwise,$(z_1 \cup z_2 \cup ... \cup z_n)$ will not be a zone. Use the above property iteratively, we can obtain a zone Z' which consists of only two subzones z_p and z_q, such that $Z' = z_p \cup z_q$ for some $p,q \in \{1,2,...,n\}$, that is, z_p and z_q are mergeable. This contradicts the fact that no pair of zones z_i and $z_j \in Z$ is mergeable.

5 Application Example

In this section, we give a real example of bus arbitration. In a bus system, there are several masters and one arbiter attached to the bus. All masters on the bus will request the bus to do some data transfers, but the bus can only serve one master at a

input : DBM: z_1, z_2 //z_1 does not interset with z_2
output: DBM: z //the DBM generated by merging z_1 and z_2;If z_1 and z_2 cannot be merged, z will be unrestricted

```
1  bool : mergable
2  int : m, n;
3  unMergable ← TRUE;
4  m ← n ← 0;
5  REDUCE(z1);
6  REDUCE(z2);
7  INIT(z)    //set z unrestricted
8  for each z1(i, j) and i ≠ j do
9      if z1(i, j) is complement to z2(j, i) then
10         unMergable ← FALSE;
11         m ← i;
12         n ← j;
13         z(i, j) ← z2(i, j);
14         z(j, i) ← z1(j, i)
15         break;
16     end
17 end
18 if unMergable = TRUE then
19     return z;
20 end
21 else
22     for each z1(i, j) and i ≠ j and i ≠ m, n and j ≠ m, n do
23         if z1(i, j) =″< ∞″ and z2(i, j) ≠″< ∞″ then
24             z(i, j) ← z2(i, j);
25         end
26         else if z1(i, j) ≠″< ∞″ and z2(i, j) =″< ∞″ then
27             z(i, j) ← z1(i, j);
28         end
29         else if z1(i, j) ≠″< ∞″ and z2(i, j) ≠″< ∞″ then
30             if z1(i, j) = z2(i, j) then
31                 z(i, j) ← z1(i, j);
32             end
33             else
34                 unMergable ← TRUE;
35                 break;
36             end
37         end
38     end
39     if unMergable = TRUE then
40         INIT(z);    //set z unrestricted
41     end
42 end
43 return z
```

Algorithm 2: DBM Merging (z_1, z_2)

```
   input  : DBM: Z      //Z = {z_1, z_2, ..., z_n}
   output: DBM: Z'      //Z' = {z'_1, z'_2, ..., z'_m}
 1 Z' ← ∅
 2 for each z_i ∈ Z do
 3     for each z_j ∈ Z and z_j ≠ z_i do
 4         z_tmp = DBM-Merge(z_i, z_j);
 5         if z_tmp ≠ unrestricted DBM then
 6             z_i ← z_tmp;
 7             Z = Z − {z_j};
 8         end
 9     end
10     Z' = Z' ∪ {z_i};
11 end
12 return Z
```

Algorithm 3: DBM Set Merging (Z)

time. In order to resolve conflicts, masters will usually be prioritized. When requests from masters arrive at the same time, the arbiter will decide which master can use the bus according the priorities of the masters. In our example, there are three masters and one arbiter attached to the bus system. The masters are modelled as in Figure 3.

Each of the three masters, namely Master A, Master B, and Master C, will request the bus when it wants to transfer data on the bus by entering the "requesting" state. It will stay in the "requesting" state until the arbiter grants its request. The priorities of Master A, Master B, and Master C are respectively 5, 3, and 2 where a higher value implies a higher priority.

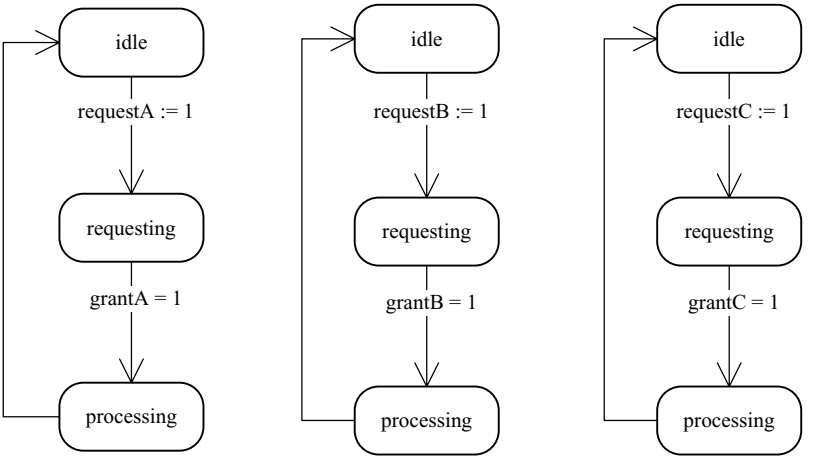

Fig. 3. PTA Model for Masters

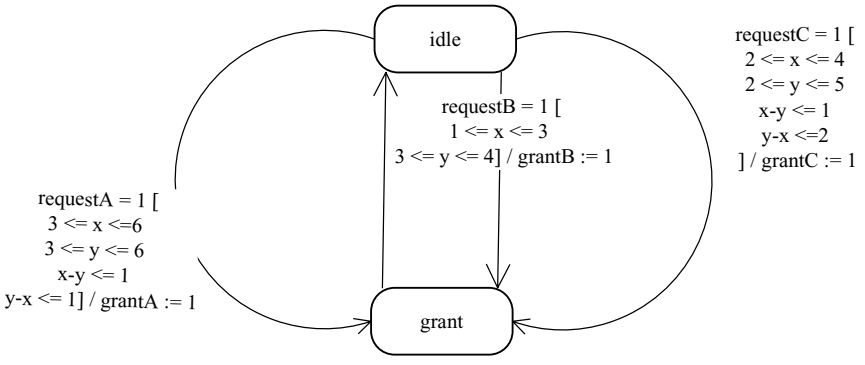

Fig. 4. PTA Model for Arbiter

Table 1. Prioritized Timed Automata of Arbiter

PTA	#Modes	#Trans
Without DBM Merging	2	7
With DBM Merging	2	5

Table 2. Verifying Bus-Arbitration Example (the whole state graph) With and Without DBM Merging

System State Graph	#Modes	#Trans	Mem (MB)	Time (sec)
Without DBM Merging	3,088	12,596	2.96	6.78
With DBM Merging (reduction)	1,524 (50%)	7,232 (44%)	1.08 (64%)	3.54 (48%)

The arbiter is modelled as in Figure 4. If there is any request from the masters, it will decide which master can use the bus according their priorities and trigger conditions, and then enter the ”grant” state.

The proposed DBM subtraction and merging algorithms were all implemented into the *State Graph Manipulators* (SGM) model checker [15,22] and applied to the bus arbitration example. Tables 1 and 2 show comparisons between applying and not applying the proposed DBM merging algorithm, for the arbiter alone and for the whole system graph, respectively. From Table 1, we can see that the arbiter PTA model illustrated in Fig. 4, which had 2 modes and 4 transitions, was transformed into a larger PTA model having 7 transitions when merging is not applied and having 5 transitions when merging is applied. This shows that merging significantly reduces the set of zones partitioned through a series of DBM subtraction. Further, it also shows that if priority is not supported, the user would have to model a larger PTA model. This increase in model size becomes significant with application complexity and will thus be a tedious task for a normal user. The burden of performing this tedious task is thus alleviated by our proposed methods.

The requirement for or the benefits of DBM merging after a series of DBM subtraction can be clearly observed from Table 2, which shows that there is a significant

reduction in the number of modes (50%) and transitions (44%) after applying DBM merging. The computing resources are also reduced by a factor of 64% and 48% for memory space and CPU time, respectively.

6 Conclusions and Future work

In this work, we have proposed *Prioritized Timed Automata* to model timed systems with multiple priorities. We have also developed the semantics of Prioritized Timed Automata for model checking. Three algorithms were proposed for DBM subtraction and merging. The DBM subtraction algorithm generates the minimal number of partitioned zones. After a series of DBM subtraction operations, the number of zones partitioned may be mergeable. The DBM merging algorithm helps to reduce the final number of partitioned zones. The proposed algorithm when applied to a simple bus arbiter showed significant reductions in model size and computing resources. Future work will include reduction in the complexities of the proposed algorithms and their application to larger applications.

References

1. K. Altisen, G. Gössler, and J. Sifakis. A methodology for the construction of scheduled systems. In *Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT), Lecture Notes in Computer Science*, volume 1926, pages 106–120. Springer Verlag, September 2000.
2. K. Altisen, G. Gössler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Real-Time Systems*, 23:55–84, 2002.
3. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.
4. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
5. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. Technical Report CS-R8503, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, 1985.
6. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and Y. Wang. UPPAAL: a tool suite for automatic verification of real-time systems. In *Proceedings of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, Oct 1996.
7. R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
8. J. Camilleri. Introducing a priority operators to ccs. Technical Report 157, Cambridge, 1989.
9. E.M. Clarke and E.A. Emerson. Design and sythesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Logics of Programs Workshop*, volume 131 of *LNCS*, pages 52–71. Springer Verlag, 1981.
10. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.
11. R. Cleaveland and M. Hennessy. Priorities in process algebra. In *Proceedings of the 3rd Symposium on Logic in Computer Science*, Edinburgh, 1988.
12. David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212. Springer-Verlag, 1989.

13. G. Gössler and J. Sifakis. Priority systems. In *Proceedings of the 2nd International Symposium on Formal Methods for Components and Objects (FMCO), Lecture Notes in Computer Science*, volume 3188, pages 314–329. Springer Verlag, November 2003.
14. P.-A. Hsiung and S.-W. Lin. Model checking timed systems with priorities. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA, Hong-Kong, China)*. IEEE Computer Society Press, August 2005. (accepted for publication).
15. P.-A. Hsiung and F. Wang. A state-graph manipulator tool for real-time system specification and verification. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA)*, October 1998.
16. G. Lowe. *Probabilities and Priorities in Timed CSP*. PhD thesis, St. Hugh's College, University of Oxford, Hilary Term, 1993.
17. R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.
18. J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer Verlag, 1982.
19. J. Sifakis. The compositional specification of timed systems. In *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV), Lecture Notes in Computer Science*, volume 1633, pages 2–7. Springer Verlag, July 1999.
20. J. Sifakis. Modeling real-time systems — challenges and work directions. In *Proceedings of the 1st International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science*, volume 2211, pages 373–389. Springer Verlag, October 2001.
21. F. Wang. RED: Model-checker for timed automata with clock-restriction diagram. In *Proceedings of the Workshop on Real-Time Tools*, August 2001. Technical Report 2001-014, ISSN 1404-3203, Department of Information Technology, Uppsala University.
22. F. Wang and P.-A. Hsiung. Efficient and user-friendly verification. *IEEE Transactions on Computers*, 51(1):61–83, January 2002.
23. S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.