Model Checking Timed Systems with Priorities

Pao-Ann Hsiung and Shang-Wei Lin
Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi, Taiwan−621, ROC
E-mail: hpa@computer.org

Abstract

Priorities are used to resolve conflicts such as in resource sharing and in safety designs. The use of priorities has become indispensable in real-time system design such as in scheduling, synchronization, arbitration, and fairness guaranteeing. There are several modeling frameworks that show how timed systems with priorities are to be designed and how priority schedulers can be automatically synthesized. However, the verification of timed systems with priorities using model checking is still a relatively untouched area. We show what the issues are in model checking timed systems with priorities and how the issues are solved in this work. In the process, we propose an optimal zone subtraction algorithm. The method has been implemented into the SGM model checker and successfully applied to real-time embedded systems and safety-critical systems, which illustrate the feasibility and advantages of the proposed verification method.

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. Priorities may take different forms in different methods such as the process arrival time in FIFO scheduling, the task period in rate monotonic scheduling, the task deadline in earliest deadline scheduling, or even as simple as an integer value assigned by a user to a real-time task. Priorities are also necessary for modeling interrupts in embedded systems.

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.

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 [18, 23], RED [22], UPPAAL [6], and Kronos [24]. The main issue here is how to extend the syntax and semantics of TA without losing its original theoretical basis for model checking.

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 to solving the above described issues in prioritizing timed automata and then verifying them. An application example is given in Section 5 to show how priority helps in model checking. The article is concluded and future research directions are given in Section 6.

2. Related Work

Several work of Joseph Sifakis [1, 2, 16] 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 as 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 [16]. 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

Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)
1533-2306/05 $20.00 © 2005 **IEEE**

transition condition expires) and *eager* (must transit as soon as the transition condition holds) [2]. 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 [19], 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 [12]. A prioritized version of ACP was proposed by Baeten et al. [5]. Camilleri proposed a prioritized version of CCS [20] with a left biased choice operator [9]. 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 [10, 11, 21] is an automatic technique for verifying finite state concurrent systems. 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. 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) [10].

Our model checking procedures for prioritized timed automata are implemented in the *State-Graph Manipulators* (SGM) model checker [18, 23], 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

Before we discuss solutions to the issues introduced in Section 1 on model checking prioritized timed automata, we give some basic definitions.

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 can be expressed as $\eta = \zeta \wedge \beta$, where ζ is a clock constraint and β is a Boolean condition on the discrete variables. □

Let $B(C, D)$ be the set of all mode predicates over C and D. We extend the conventional definition of TA by prioritizing the transitions, 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) [8] and the clock constraints are represented by *Difference Bound Matrices* (DBM) [14].

For verifying a real-time embedded system modeled by a set of prioritized timed automata, the system properties can be specified in a *temporal logic*. The SGM model checker chooses TCTL as its logical formalism, as defined below.

Definition 4 Timed Computation Tree Logic (TCTL)
A *timed computation tree logic* formula has the following syntax: $\phi ::= \eta \mid EG\phi' \mid E\phi' U_{\sim c} \phi'' \mid \neg\phi' \mid \phi' \vee \phi''$, where η is a mode predicate, ϕ' and ϕ'' are TCTL formulae, $\sim \in \{<, \leq, =, \geq, >\}$, and c $\in \mathcal{N}$. $EG\phi'$ means there is a computation from the current state, along which ϕ' is always true. $E\phi' U_{\sim c} \phi''$ means there exists a computation from the current state, along which ϕ' is true until ϕ'' becomes true, within the time constraint of $\sim c$. Shorthands like EF, AF, AG, AU, $\wedge$, $\rightarrow$ can all be defined [17]. □

Definition 5 Model Checking
Given a prioritized system state graph $\mathcal{A}_\mathcal{S}$ that represents a real-time embedded system $\mathcal{S}$ with priority and a TCTL formula, ϕ, expressing some desired specification, model checking verifies if $\mathcal{A}_\mathcal{S}$ satisfies ϕ, denoted by $\mathcal{A}_\mathcal{S} \models \phi$. Model checking can be either explicit using a labeling algorithm or symbolic using a fixpoint algorithm. □

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. The transformation of PTA to TA 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. Model Transformation

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.

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

$$\begin{aligned}
\neg\tau_i(t) &= \overline{\zeta(t)} \vee \neg\beta(t) \\
\overline{\zeta(t)} &= x \sim' c, \quad \text{if } \zeta(t) = x \sim c, \\
&= x - y \sim c, \quad \text{if } \zeta(t) = x - y \sim c, \\
&= \overline{\zeta_1} \vee \overline{\zeta_2}, \quad \text{if } \zeta(t) = \zeta_1 \wedge \zeta_2, \\
\neg\beta(t) &= d \sim' c, \quad \text{if } \beta(t) = d \sim c \\
&= \neg\beta_1 \vee \neg\beta_2, \quad \text{if } \beta(t) = \beta_1 \wedge \beta_2, \\
&= \beta_1, \quad \text{if } \beta(t) = \neg\beta_1,
\end{aligned} \tag{1}$$

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.

A transition t can fire only when the following conditions holds.

$$\beta_i(t) \wedge \zeta_i(t) \wedge \left[\left(\bigwedge_{\pi_i(t') > \pi_i(t)} \neg \beta_i(t') \right) \vee \left(\bigwedge_{\pi_i(t') > \pi_i(t)} \overline{\zeta_i(t')} \right) \right] \quad (2)$$

4.2. Optimal DBM Subtraction

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) [14], which is defined as follows.

Definition 6 Difference Bound Matrix (DBM)
Given a clock zone z that represents a clock constraint 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 intersection of two convex polyhedra gives a convex polyhedron, hence the set of clock zones is closed under intersection. Time elapse removes the upper bounds on all clocks (all $D(i, 0), 1 \leq i \leq n$ elements are changed to $< \infty$), however it is still a convex polyhedron, hence the set of clock zones is also closed under time elapse. The reset operation sets the values of one or more clocks to zero and adjusts the clock value differences accordingly. Geometrically, reset amounts to a projection of the clock zone on the axes corresponding to the clocks that are reset. Projection of a convex polyhedron is still a convex polyhedron, hence the set of clock zones is also closed under reset.

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) 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.

$$z_1 - z_2 = z_1 \cap \overline{z_2} \quad (3)$$

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 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.

- For each non-diagonal element in the DBM z_2, we obtain its complement zone by reversing the relational operator. This complement zone does not intersect with z_2, but may intersect with z_1. (Steps 2–4)
- For each complement zone, we find its intersection with z_1, denoted as z_{tmp}, which is a zone. (Step 5)
- There is a merge procedure that tries to combine a zone z_{prev} generated in the previous iteration with the currently generated zone z_{tmp} (Steps 6, 7). If the merged DBM is indeed convex, i.e. a valid zone, then z_{prev} is discarded from the set z of zones to be returned (Step 8). Finally, the merged zone or z_{tmp} is added into z (Step 10).

The complexity of the algorithm is $O(n^4)$, where n is the number of clock variables. Since we work on each element of z_2, it gives a $O(n^2)$ complexity. In each iteration, we need to perform zone intersection, which is also $O(n^2)$ complexity. Hence, the $O(n^4)$ complexity of the DBM subtraction algorithm.

We can prove that a minimal number of partitions (zones) are generated after subtraction. Due to page limit, we only give the theorem here.

Theorem 1 *The number of zones generated by the DBM subtraction algorithm is minimal.* □

input : DBM: z_1, z_2 $\quad //z_2(i,j) = \ '' \sim c''$
output: DBM*: z $\quad$ //set of DBMs

1 DBM*: $z_{tmp}, z_{prev} = NULL$
2 **for** *each* $z_2(i,j), i \neq j$ **do**
3 $\quad$ init(z_{tmp}) $\quad //z_{tmp}(i,j) = ''< \infty''$, $i \neq j$ & $z_{tmp}(i,i) = ''\leq 0''$
4 $\quad z_{tmp}(i,j) = \sim' c$ $\quad //\sim' \in \{>, \geq\}$, $\sim \in \{\leq, <\}$
5 $\quad z_{tmp} = z_{tmp} \cap z_1$
6 $\quad$ **if** z_{prev} & mergeable(z_{tmp}, z_{prev}) **then**
7 $\quad\quad z_{tmp} =$ merge(z_{tmp}, z_{prev})
8 $\quad\quad z = z \backslash \{z_{prev}\}$
9 $\quad$ **end**
10 $\quad z = z \cup \{z_{tmp}\}$
11 $\quad$ **if** $z_{prev} \neq z_{tmp}$ **then**
12 $\quad\quad z_{prev} = z_{tmp}$
13 $\quad$ **end**
14 **end**
15 **return** z

Algorithm 1: DBM Subtraction $z_1 - z_2$

5. Application Example

The proposed verification method was implemented in the State-Graph Manipulators (SGM) model checker [18, 23]. The extensions included both syntax and semantics of priorities in timed automata. Clock zones had to be subtracted and the set of generated zones has to be split into different state graph modes, which were then pushed into the model checker stack.

The proposed verification method can be applied to any real-time embedded system with priority that can be modeled accurately using prioritized timed automata. Besides several smaller examples, we applied the method to a real-world large example, which is an embedded active structural control system (ASCS) used to prevent damages on buildings caused by earthquakes or strong wind [7, 15]. The ASCS system was modeled by three periodic processes, where we verified the system both with and without priorities. Figure 1 illustrates the three PTA: modeler, data, and pulser. The modeler computes for 2.3 to 2.5 ms and updates a shared buffer, which takes 1 ms. The pulser has a period between 3.2 and 14.5 ms. It reads the data in the buffer for 0.2 ms and writes them to actuators for 5.8 ms. The modeler and pulser share the same processor. It is required that the data read by the pulser is fresh, that is, stored by modeler at most 13 ms ago. Freshness is imposed by the data process. The updated transition in modeler is synchronized with the refresh transition in data. For the system to be schedulable, there should not be any dead state in the three process automata, where a dead state is one without any outgoing transition. To guarantee schedulability, priorities are necessary as described in the following.

- (p_1) The expire transition in data has priority over the read transition in pulser.
- (p_2) The compute transition in modeler has priority over the read transition in pulser.
- (p_3) The update transition in modeler has priority over the read transition in pulser.

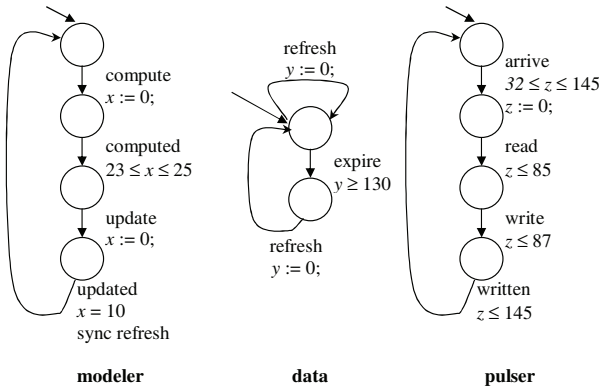

Figure 1. Active Structural Control System

The application results for ASCS were obtained using SGM running on a Linux machine with an Intel Pentium 4 2.8 GHz processor and 1 GB RAM. The results of model checking the ASCS real-time embedded system with and without priorities are given in Table 1. It is observed that there is a significant reduction in the sizes of the prioritized system state-graphs (rows 2–4) compared to the unprioritized one (row 1), where the size consists of the number of modes and transitions. We can also see that there are only 2 dead states in the fully prioritized state graph. After checking those dead states, we found it is due to the inability of the SGM model checker to model different transition urgencies such as *delayable* (must execute before trigger expires) and *eager* (must execute as soon as trigger holds). Ignoring the 2 dead states, we see that the system becomes schedulable only after all the 3 priority rules $\{p_1, p_2, p_3\}$ are incorporated into the 3 process model. Further, we can also observe that having priorities not only allowed us to check the schedulability of the system, but the whole verification process utilized much less computing resources such as CPU time and memory.

Besides real-time embedded systems, the proposed method was also applied to safety-critical systems that were modeled using Safecharts [13]. Due to page-limit, we could not cover this work here. Mainly, the priorities were used to resolve non-determinisms that could lead to unsafe states. Adding priorities ensured the models were safe and the

Table 1. Verifying ASCS With Priorities

Priorities	#Modes	#Trans	#Dead	MB	sec
$\{\}$	90,905	190,903	386	77.47	13.69
$\{p_1\}$	76,811	160,555	3,006	65.23	11.68
$\{p_1, p_2\}$	135	207	13	0.13	0.02
$\{p_1, p_2, p_3\}$	75	95	2	0.13	0.02

safety critical systems such as railway signaling system were model checked using SGM extended with priorities.

6. Conclusions

In this work, we have shown how the popular timed automata model for real-time systems can be extended with priorities for verifying real-time embedded systems. We have proposed a novel optimal algorithm for DBM subtraction, which takes care of the semantics of priorities in timed automata. We have shown through several examples how the algorithm generates least number of zones. A real application example also illustrates the benefits of our method for verifying real-time embedded systems with priority.

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. 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. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[5] J. Baeten, J. Bergstra, and J. 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] V. Braberman. *Modeling and Checking Real-Time System Designs*. PhD thesis, Department of Computation, Universidad de Buenos Aires, 2000.

[8] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[9] J. Camilleri. Introducing a priority operators to ccs. Technical Report 157, Cambridge, 1989.

[10] E. Clarke and E. 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.

[11] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[12] R. Cleaveland and M. Hennessy. Priorities in process algebra. In *Proceedings of the 3rd Symposium on Logic in Computer Science*, Edinburgh, 1988.

[13] H. Dammag and N. Nissanke. Safecharts for specifying and designing safety critical systems. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, pages 78–87, October 1999.

[14] D. 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.

[15] W. Elseaidy, R. Cleaveland, and J. J. Baugh. Modeling and verifying active structural control systems. *Science of Computer Programming*, 29(1–2):99–122, 1977.

[16] 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.

[17] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for reat-time systems. In *Proceedings of the IEEE International Conference on Logics in Computer Science (LICS)*, pages 394–406, June 1992.

[18] 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.

[19] G. Lowe. *Probabilities and Priorities in Timed CSP*. PhD thesis, St. Hugh's College, University of Oxford, Hilary Term, 1993.

[20] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[21] 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.

[22] 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.

[23] F. Wang and P.-A. Hsiung. Efficient and user-friendly verification. *IEEE Transactions on Computers*, 51(1):61–83, January 2002.

[24] 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.