The Journal of Systems and Software 82 (2009) 1627–1641

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Modeling and verification of real-time embedded systems with urgency

Pao-Ann Hsiung [a], Shang-Wei Lin [a], Yean-Ru Chen [b], Chun-Hsian Huang [a], Chihhsiong Shih [c], William C. Chu [c,*]

[a] National Chung Cheng University, Chiayi-62102, Taiwan, ROC
[b] National Taiwan University, Taipei-10617, Taiwan, ROC
[c] Tunghai University, Taichung 40704, Taiwan, ROC

ARTICLE INFO

Article history:
Available online 19 March 2009

Keywords:
Modeling
Formal verification
Urgency

ABSTRACT

Real-time embedded systems are often designed with different types of urgencies such as delayable or eager, that are modeled by several urgency variants of the timed automata model. However, most model checkers do not support such urgency semantics, except for the IF toolset that model checks timed automata with urgency against observers. This work proposes an *Urgent Timed Automata* (UTA) model with zone-based urgency semantics that gives the same model checking results as absolute urgency semantics of other existing urgency variants of the timed automata model, including timed automata with deadlines and timed automata with urgent transitions. A necessary and sufficient condition, called *complete urgency*, is formulated and proved for avoiding zone partitioning so that the system state graphs are simpler and model checking is faster. A novel zone capping method is proposed that is time-reactive, preserves complete urgency, satisfies all deadlines, and does not need zone partitioning. The proposed verification methods were implemented in the SGM CTL model checker and applied to real-time and embedded systems. Several experiments, comparing the state space sizes produced by SGM with that by the IF toolset, show that SGM produces much smaller state-spaces.

1. Introduction

A popular model for real-time embedded systems is *Timed Automata* (TA) (Alur and Dill, 1994), for which several model checkers such as SGM (Wang and Hsiung, 2002), RED (Wang, 2001), UPPAAL (Bengtsson et al., 1995), and Kronos (Yovine, 1997) have been developed to verify them formally. However, timed automata models assume a *lazy* semantics, that is, an enabled state transition need not be taken as long as the invariant condition of the state is not violated. Lazy transition semantics are too general to model the urgent behavior found in many real-world systems such as medical devices, home appliances, robotics, and others. Thus, the TA model was extended with urgency semantics such as the *Timed Automata with Deadlines* (TAD) (Bornot et al., 1997), *Timed Automata with Urgent Transitions* (TAUT) (Barbuti and Tesei, 2004), *Timed I/O Automata with Stopping Condition* (Kaynar et al., 2003), and *Timed I/O Automata with Urgency* (Gebremichael and Vaandrager, 2005). These extended variants incorporate different syntax for accurately modeling urgency. However, system verification using such extended variants has not received as much attention in the area of *Computation Tree Logic* (CTL) model checking (Clarke and Emerson, 1981). This work focuses on proposing a class of TA with urgencies called *Urgent Timed Automata* (UTA), its corresponding *zone-based urgency semantics*, and how an urgent timed system state graph can be model checked against CTL properties.

Before urgency semantics were defined for timed automata, state invariants were used to model urgent behavior by forcing a TA to transit to successor states before the invariants are violated due to time elapse. However, the invariant-based method was only applicable to hard deadlines, where the stopping of time due to urgency and the non-existence of any transition to take when time is stopped resulted in a *timelock*. Stopping conditions associated with timed I/O automata also result in similar timelocks. Different methods were proposed to avoid timelocks such as associating a transition with a *deadline predicate* (Bornot and Sifakis, 2000; Bornot et al., 1997; Sifakis and Yovine, 1996), with an *urgency predicate* (Gebremichael and Vaandrager, 2005), or with a positive rational parameter representing deadline (Barbuti and Tesei, 2004). However, there is very little research on how such models with urgent semantics are to be verified using CTL model checking (Clarke and Emerson, 1981). There is also no CTL model checker that can *directly* model check these models without workarounds. The IF toolset (Bozga et al., 1999) can model check timed automata with urgency against properties written as *observers*, which are IF processes that monitor and guide simulation.

The expressiveness of deadline predicates, urgency predicates, and deadline parameters are all same (Gebremichael and

* Corresponding author.
E-mail address: cchu@thu.edu.tw (W.C. Chu).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.03.013

Vaandrager, 2005; Barbuti and Tesei, 2004). Further, it has also been shown that deadline predicates can be simplified into urgency types, namely *lazy*, *delayable*, and *eager*. We thus decided that we need only address the model checking of timed automata having transitions associated with urgency types. We call this model as *Urgent Timed Automata* (UTA). The major issue in this work is how we restrict time progress so that the enabled urgent transitions are taken as required by their semantics and the models can be model checked.

The issues to be resolved in this work are as follows. The first is a *soundness* issue, which means we need to determine an urgency semantics for UTA models that is consistent with the conventional TA model checking. As a solution, we propose a *zone-based* urgency semantics that gives the same model checking results as the urgency semantics of TAD and TAUT. The second is a *completeness* issue, which means we need to determine the class of UTA that can be model checked by a conventional TA model checker. As a solution, we found that UTA under the proposed complete urgency restriction can be model checked. The third is a *construction* issue, which means we need to find a method for enforcing urgencies while guaranteeing time-reactivity, preserving complete urgency, satisfying all deadlines, and not needing zone partitioning. As a solution, we propose a novel *zone-capping* operation that is proved to possess all the above characteristics.

In summary, our major contribution in this work is the proposal of solutions to the above three issues, the theoretical proofs and analysis of the solutions, and their implementation in the SGM model checker along with application to several examples from the real-time and embedded systems domain. The proposed solutions mainly include the UTA model, the zone-based urgency semantics, the complete urgency restriction, and the zone capping operation, which result in time-reactive state graphs, satisfying all deadlines, and without the need for zone partitioning.

The article is organized as follows. Section 2 describes previous work related to urgency modeling and verification. Basic definitions used in our work are given in Section 3. Section 4 will formulate the solutions to solve the above described issues in model checking UTA. Section 5 describes the algorithm, the theoretical analysis, and its application to several examples. The article is concluded and future research directions are given in Section 6.

2. Related work

Most works that extend the timed automata model with urgency semantics (Barbuti and Tesei, 2004; Bornot and Sifakis, 2000; Bornot et al., 1997; Gebremichael and Vaandrager, 2005; Sifakis and Yovine, 1996) are focused on the modeling aspects such as expressiveness and compositionality. Except for the IF toolset (Bozga et al., 1999), little attention has been paid to the verification of systems modeled by these urgency extended models. In this section, we first discuss the differences among the various model extensions and then summarize on the state-of-the-art verification techniques for these models.

Timed automata with deadlines (TAD) (Bornot and Sifakis, 2000; Bornot et al., 1997; Sifakis and Yovine, 1996) proposed by Sifakis et al. was among the first models that extended TA with urgency. An urgent transition was associated with a *deadline predicate*, which represents the condition when time progress must stop to allow for the urgent transition to be taken. Once the urgent transition is taken, time progress can continue. TADs are *time-reactive* or *timelock-free*, that is, the system never comes to a complete halt due to the violation of a deadline and some enabled transition can always be taken when time progress is stopped. Semantically, a TAD state s is associated with a *time progress condition* (TPC) $c_s = \neg\bigvee_{i\in I} d_i$, where d_i is the deadline predicate of transition $i \in I$ and I is the set of all outgoing transitions from state s (Bornot et al., 1997). However, TPC is not suitable for model checking because it results in non-convex clock zones which require further processing such as zone partitioning (Lin et al., 2005). It was also shown that any TAD can be transformed into an equivalent TAD with only eager and lazy transitions (Bornot et al., 1997).

An extension is called *Timed Automata with Urgent Transitions* (TAUT) (Barbuti and Tesei, 2004), which associates with a TA a small rational number called *deadline parameter*, $l \in \mathcal{Q}_{>0}$, such that urgent transitions must be taken within l time units after they are enabled. The expressiveness of TAUT is the same as that of TAD, but TAUT allows shorter deadline specifications. Another improvement is that TAUT allows right-closed TPC, which cannot be handled by TAD. A right-closed TPC is derived when we have an eager transition with a left-open deadline predicate and since time is dense we do not know when to take that eager transition in a TAD; however, with a deadline parameter l we can take the eager transition at $x_t + l$, where x_t is the lower bound of the deadline predicate and $l \in \mathcal{Q}_{>0}$ is made as small as possible.

The deadline predicates have also been applied to *Timed I/O Automata* (TIOA) (Kaynar et al., 2003) which originally had a stopping condition for specifying deadlines. Similar to state invariants, stopping conditions may also result in timelocks. This extension of TIOA associated urgent transitions with an *urgency predicate*, which made them time-reactive by construction and closed under composition. Invariant properties are proved by constructing time progress predicates for each urgent transition and then taking the conjunction of these time progress predicates as the condition for time progress. However, a time progress predicate is the negation of urgency predicate, which would result in non-convex clock zones and thus make model checking difficult as it further requires complex zone partitioning to keep them convex. The authors of (Gebremichael and Vaandrager, 2005) remarked that by restricting the clock zones in urgency predicates one can avoid non-convex time progress predicates, however this is too strict a restriction.

From the above descriptions, we can observe that TAD and TIOA with urgency use time progress conditions (predicates) that can result in non-convex clock zones, while TAUT adopts a TA transformation approach. Our work is similar to the transformation approach of TAUT, however we do not need the deadline parameter l and our approach is much simpler in terms of conformance with the original TA model and region semantics. Similar to TAUT, we allow zones with left-open transition enabling time intervals, which are not allowed by TAD and TIOA with urgency. Further, unlike all the other models, TAD, TAUT, and TIOA with urgency, we separate prioritization from urgency, which constitutes a more general and useful semantics. Our previous work on prioritization of TA transitions (Lin et al., 2005) is applicable to the UTA model in this work.

Support for modeling urgency in systems and verifying them has been incorporated in tools such as IF (Bozga et al., 1999) and UPPAAL (Bengtsson et al., 1995). The IF toolset is an environment for modeling and validation of heterogeneous real-time systems using TAD. It consists of two parts: a syntactic transformer, which provides language level access to IF descriptions and has been used to implement static analysis and optimization techniques, and an open exploration platform, which gives access to the graph of possible executions. IF has been connected to some state-of-the-art model checkers and test-case generators. IF can also model check directly using observers. UPPAAL uses urgent channels that are taken as soon as they are enabled, however time constraints cannot be associated with urgent channels. We pose no such restriction on urgent transitions.

Fig. 1 shows how an eager transition is enforced using invariants, TPC, and the newly proposed zone capping. We find that only zone capping succeeds in associating the model with a correct and

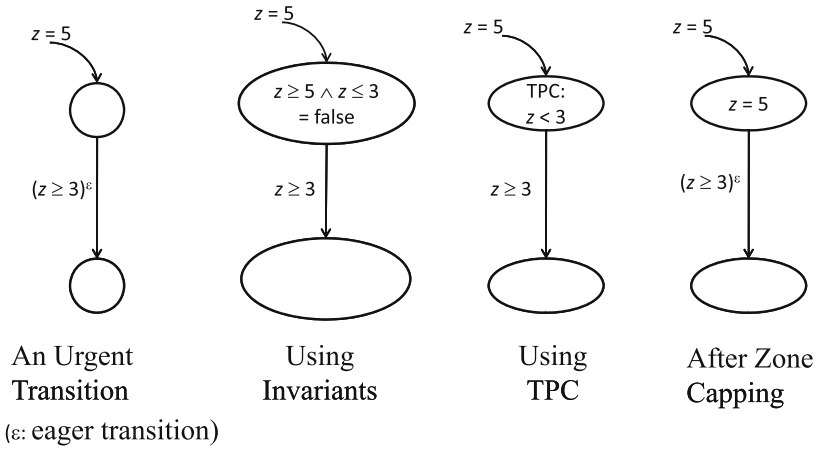


Fig. 1. Enforcing urgency using different methods.

intuitive semantics, in terms of time-reactivity and model checking, as explained in the following. If a user specified an invariant of $z \leqslant 3$ to enforce the eager transition with trigger $z \geqslant 3$ to be taken as soon as enabled, then we might end up with some runs being eliminated because if the mode clock zone is $z \geqslant 5$, then when conjuncted with the invariant $z \leqslant 3$, would be false. Thus, invariants fail to enforce correct urgency semantics. Since a TPC is constructed from the deadline predicates in transition triggers, in this example, the TPC would be $z < 3$. However, when time stops progressing at $z = 3$, the mode is not yet entered so there is a timelock. Originally, TPC guaranteed time-reactivity only under the condition that when time stops the transition is enabled. We have violated this assumption to produce a timelock. Zone capping takes the mode clock zone $z \geqslant 5$ also into consideration when stopping time progress (bounding or capping the zone), thus no such assumption is required. Zone capping thus provides a correct and intuitive urgency semantics.

Several other work related to deadline specification include the use of probabilistic timed automata to verify soft deadlines (Kwiatkowska et al., 2000), the timed extension for AltaRica – a language for specifying constraint automata (Cassez et al., 2002), and the timed extensions for SDL (Bozga et al., 2001), for MSC (Hogrefe et al., 2001), and for TTCN (Hogrefe et al., 2001) specification languages in the INTERVAL project (INTERVAL, 1999). These are only extensions in specification syntax and semantics, without addressing the verification of systems specified by those timed extensions.

3. Preliminaries

We first introduce the basic definitions required for the proposed work. Given a set C of clock variables and a set D of discrete variables over integers, a *mode predicate* η over C and D is defined as: $\eta := false|\zeta \wedge \beta$, where ζ is a clock constraint over C and β is a Boolean constraint over D. A clock constraint ζ is defined as $\zeta := x \sim c|x - y \sim c|\zeta_1 \wedge \zeta_2$, where $x, y \in C, c \in \mathcal{N}, \sim \in \{\leqslant, <, =, \geqslant, >\}$, and ζ_1, ζ_2 are all clock constraints. A Boolean constraint β is defined as $\beta := d \sim c|\beta_1 \wedge \beta_2|\neg\beta_3$, where $d \in D$ and $\beta_1, \beta_2, \beta_3$ are discrete variable constraints. Let $B(C, D)$ represent the set of all mode predicates over C and D.

Definition 1 (*Urgent Timed Automaton*). An *Urgent Timed Automaton* (UTA) is a tuple $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \Omega)$ such that: M is a finite set of modes. $m^0 \in M$ is the initial mode. C is a set of clock variables. D is a set of discrete variables. L is a set of synchronization labels, and $\alpha \in L$ is a special label that represents asynchronous behavior (i.e. no need of synchronization). $\chi : M \mapsto B(C, D)$ is an *invariance* function that labels each mode with a condition true in that mode. $T \subseteq M \times M$ is a set of mode transitions. $\Omega : T \mapsto \langle L, B(C, D), \{\lambda, \delta, \varepsilon\}, \gamma\rangle$ is a description of T. For ease of notations, we use the following short forms: $\Omega(t) = \langle \psi(t), \tau(t), \mu(t), \rho(t)\rangle$, where $t \in T$. $\psi : T \mapsto L$ associates a synchronization label with a transition. $\tau : T \mapsto B(C, D)$ defines the transition triggering conditions, where $\tau(t) = \zeta_{\tau(t)} \wedge \beta_{\tau(t)}$ is the conjunction of a clock zone and a Boolean condition associated with the transition. $\mu : T \mapsto \{\lambda, \delta, \varepsilon\}$ associates an urgency type with a transition, including *lazy*, *delayable*, and *eager*, respectively, whose semantics are as follows. An enabled lazy transition needs not to fire as long as the invariant condition $\chi(m)$ of the transition's source mode m is not violated. An enabled delayable transition must fire before it becomes disabled due to time elapse. An eager transition must fire as soon as it is enabled. $\rho : T \mapsto \gamma$, where γ is a partial function mapping $(C \cup D)$ to $\mathcal{N}$, restricted to 0 for clocks in C, i.e., ρ is an *assignment* function associated with each transition with clock resetting.

The semantics of a UTA can be defined by its state and computation run as follows.

Definition 2 (*State and run*). A pair $s = (m, v)$ is called a *state* of a UTA $\mathcal{A}_i = (M_i, m_i^0, C_i, D_i, L_i, \chi_i, T_i, \Omega_i)$ if $m \in M_i$ and v maps each clock from C_i to a non-negative real number in $\mathcal{R}_{\geqslant 0}$ and each discrete variable from D_i to an integer in $\mathcal{N}$ such that v satisfies the invariant $\chi_i(m)$. A sequence of state-transition pairs $\langle s_0 \xrightarrow{t_0} s_1 \ldots s_n\rangle$ is called a *run* if s_{i+1} is a successor state of s_i due to a mode transition $t_i = (s_i, s_{i+1}) \in T_i$ or due to a time transition that represents the elapse of time without changing mode. A state s is said to be *reachable* if there exists a computation run $\langle s_0 \xrightarrow{t_0} s_1 \ldots s\rangle$, where $s_0 = (m_i^0, v_0)$ is an initial state of $\mathcal{A}_i$.

Definition 3 (*Time progress and time reactivity*). Given a state $s = (m, v)$, if $\exists t \in \mathcal{R}, t > 0$ such that $s_t = (m, v + t)$ is a reachable state from s, then we say that *time progress* is possible in s, where $v + t$ maps each clock x to $v(x) + t$. If time progress is not possible in s and there is no mode transition enabled in s, then we say that there is a *time lock* in s. When there is no time lock in a UTA, we say it is *time reactive*.

States can be grouped into zones and the infinite number of states can be classified into a finite number of regions as defined in the following.

Definition 4 (*Region*). Let c_{max} be the maximal constant integer appearing in any clock constraint. Two states $s = (m, v)$ and

$s' = (m, v')$ are said to be in the same *region* if either $v(x) > c_{max}$ and $v'(x) > c_{max}$ for all $x \in C_i$ or $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and $(v(x) - \lfloor v(x) \rfloor > v(y) - \lfloor v(y) \rfloor) \leftrightarrow (v'(x) - \lfloor v'(x) \rfloor > v'(y) - \lfloor v'(y) \rfloor)$, for all $x, y \in C_i$, and $v(d) = v'(d)$, for all $d \in D_i$. Let $[s]$ denote the region to which s belongs.

Definition 5 (*Clock zone and zone).* A convex union of regions is called a *clock zone*. Given a mode m, a clock zone ζ_m, and a Boolean constraint β_m, the tuple (m, ζ_m, β_m) is called a *zone* if both ζ_m and β_m satisfy the invariant of m. A state $s = (m, v)$ is in a zone $z = (m, \zeta_m, \beta_m)$ if v satisfies both ζ_m and β_m.

In most model checkers, the clock constraints are represented by *Difference Bound Matrices* (DBM) (Dill, 1989).

Definition 6 (*Difference Bound Matrix (DBM)*). A clock zone ζ that represents a clock constraint on n clocks in $C_i = \{x_1, x_2, \ldots, x_n\}$ can be implemented as a $(n+1) \times (n+1)$ matrix Δ, where $\Delta(i,j) = (\sim, c),\ \sim \in \{<, \leqslant\}, c \in \mathcal{N} \cup \{\infty\}$, represents the constraint $x_i - x_j \sim c,\ 0 \leqslant i, j \leqslant n$. Here, $x_0 = 0$.

Example 1 shows how a clock zone is represented by a DBM.

Example 1. Given two clock zones z_1 and z_2 such that $z_1 = (x > 3) \wedge (y > 3) \wedge (x \leqslant 6) \wedge (y \leqslant 6) \wedge (x - y \leqslant 2) \wedge (y - x \leqslant 1)$ and $z_2 = (x \geqslant 1) \wedge (y \geqslant 1) \wedge (x \leqslant 7) \wedge (y \leqslant 7) \wedge (x - y \leqslant 3) \wedge (y - x \leqslant 3)$. The two DBMs Δ_{z_1} and Δ_{z_2} representing z_1 and z_2, respectively, are as follows.

$$\Delta_{z_1}: \begin{bmatrix} & 0 & x & y \\ 0 & \leqslant 0 & < -3 & < -3 \\ x & \leqslant 6 & \leqslant 0 & \leqslant 2 \\ y & \leqslant 6 & \leqslant 1 & \leqslant 0 \end{bmatrix}, \quad \Delta_{z_2}: \begin{bmatrix} & 0 & x & y \\ 0 & \leqslant 0 & \leqslant -1 & \leqslant -1 \\ x & \leqslant 7 & \leqslant 0 & \leqslant 3 \\ y & \leqslant 7 & \leqslant 3 & \leqslant 0 \end{bmatrix}$$

Given a DBM Δ representing a clock zone ζ, $\Delta(0,i)$ and $\Delta(i,0)$ are respectively the lower and upper bounds for clock x_i. Given two DBM elements $\Delta(i,j) = (\sim, c)$ and $\Delta(i',j') = (\sim', c')$, we can compare them by saying $\Delta(i,j) < \Delta(i',j')$ if either (1) $c < c'$ or (2) $c = c'$ and $\sim$ is $<$ and $\sim'$ is $\leqslant$. The other relational comparisons between $\Delta(i,j)$ and $\Delta(i',j')$ can be similarly defined.

Given two lower bound DBM elements $\Delta(0,j) = (\sim, c)$ and $\Delta'(0,j') = (\sim', c')$, a difference operator between the two DBM elements can be defined as follows:

$$\mathrm{diff}(\Delta(0,j), \Delta'(0,j')) = \begin{cases} c' - c & \text{if } \sim \text{ and } \sim' \text{ are the same,} \\ (c' - c)^+ & \text{if } \sim \in \{<\},\ \sim' \in \{\leqslant\}, \\ (c' - c)^- & \text{if } \sim \in \{\leqslant\},\ \sim' \in \{<\}. \end{cases}$$

Here the exponents + and − are used to represent infinitesimally larger and smaller numbers than those in the brackets, respectively.

For an integer c, we have the following relation: $c^- < c < c^+$. Similarly, given two upper bound DBM elements $\Delta(i,0) = (\sim, c)$ and $\Delta'(i',0) = (\sim', c')$, a difference operator can be defined as follows.

$$\mathrm{diff}(\Delta(i,0), \Delta'(i',0)) = \begin{cases} c - c' & \text{if } \sim \text{ and } \sim' \text{ are the same,} \\ (c - c')^+ & \text{if } \sim \in \{\leqslant\},\ \sim' \in \{<\}, \\ (c - c')^- & \text{if } \sim \in \{<\},\ \sim' \in \{\leqslant\}. \end{cases}$$

Similarly, given an upper bound DBM element $\Delta(i,0) = (\sim, c)$ and a lower bound DBM element $\Delta'(0,j') = (\sim', c')$, a difference operator can be defined as follows:

$$\mathrm{diff}(\Delta(i,0), \Delta'(0,j')) = \begin{cases} c + c' & \text{if } \sim \in \{\leqslant\} \text{ and } \sim' \in \{\leqslant\}, \\ (c + c')^- & \text{otherwise.} \end{cases}$$

Example 2 illustrates the relational comparisons between two DBM elements.

Example 2. For the two DBMs Δ_{z_1} and Δz_2 in Example 1, we can calculate the following:

$$\mathrm{diff}(\Delta_{z_1}(0,1), \Delta_{z_2}(0,1)) = \mathrm{diff}(\Delta_{z_1}(0,2), \Delta_{z_2}(0,2)) = 2^+,$$
$$\mathrm{diff}(\Delta_{z_1}(1,0), \Delta_{z_2}(1,0)) = \mathrm{diff}(\Delta_{z_1}(2,0), \Delta_{z_2}(2,0)) = -1$$
$$\text{and} \quad \mathrm{diff}(\Delta_{z_1}(1,0), \Delta_{z_2}(0,2)) = 5.$$

Since clocks progress at the same speed, the difference operator between two lower bound DBM elements represents the time lag between *entering* the two zones represented by the DBMs and the difference operator between two upper bound DBM elements represent the time lag between *leaving* the two zones. The difference operator between an upper and a lower bound DBM elements represent the time lag between exiting the first zone and entering the second zone. The difference definitions consider only a single clock at a time, so for actually entering or leaving a zone, we need to define zone lags that consider all clocks. Before defining zone lags, we need to ensure that they are well defined, thus a deadline restriction is enforced as given in Definition 7.

Definition 7 (*Deadline restriction).* Given two clock zones z_1 and z_2 over the same set of clocks C, represented by DBMs Δ_{z_1} and Δ_{z_2}, respectively, we say the deadline of z_1 is not later than that of z_2, denoted by $z_1 \preceq z_2$, if for each clock $x_i \in C, \mathrm{diff}(\Delta_{z_1}(i,0), \Delta_{z_2}(i,0)) \leqslant 0$.

To successfully apply our proposed zone capping method, there is a *deadline restriction* on the clock zones associated with delayable transitions. The set of clock zones specified on *delayable* transitions in a UTA model should be a *partially ordered set of zones*, as defined in Definition 8.

Definition 8 (*Partially ordered set of zones).* Given a set of clock zones $Z = \{z_1, z_2, \ldots, z_n\}$, we say Z is a *partially ordered set of zones* if for every two zones $z_i, z_j \in Z$, either $z_i \preceq z_j$ or $z_j \preceq z_i$. Thus, there exists a partial order among the zones such as $z_{i_1} \preceq z_{i_2} \preceq \cdots \preceq z_{i_n}$, where $i_1, \ldots, i_n \in \{1, \ldots, n\}$.

The deadline restriction ensures that we can use the deadline of a transition to cap the zone of a mode, without the need to calculate the intersection of two transition deadlines.

The difference operators will be used to define zone lags in Definition 9, which are well-defined due to the deadline restriction.

Definition 9 (*Zone entry lag, zone exit lag, and zone entry/exit lag).* Given two clock zones ζ_1 and ζ_2 for clocks in C, represented by DBMs Δ_1 and Δ_2, respectively, the zone entry lag, zone exit lag, and zone exit/entry lag between the two zones are denoted, respectively, by $\mathrm{enlag}(\zeta_1, \zeta_2)$, $\mathrm{exlag}(\zeta_1, \zeta_2)$, and $\mathrm{eelag}(\zeta_1, \zeta_2)$, and are defined as follows:

$$\mathrm{enlag}(\zeta_1, \zeta_2) = \max_{1 \leqslant j \leqslant |C|} \{\mathrm{diff}(\Delta_1(0,j), \Delta_2(0,j))\},$$
$$\mathrm{exlag}(\zeta_1, \zeta_2) = \min_{1 \leqslant i \leqslant |C|} \{\mathrm{diff}(\Delta_1(i,0), \Delta_2(i,0))\},$$
$$\mathrm{eelag}(\zeta_1, \zeta_2) = \min_{1 \leqslant i \leqslant |C|} \{\mathrm{diff}(\Delta_1(i,0), \Delta_2(0,i))\}.$$

When $\mathrm{enlag}(\zeta_1, \zeta_2)$ is positive, it means ζ_1 is entered later than ζ_2; when zero, it means they are entered at the same time; and when negative, it means ζ_1 is entered earlier than ζ_2. When $\mathrm{exlag}(\zeta_1, \zeta_2)$ is positive, it means ζ_1 is exited later than ζ_2; when zero, it means they are exited at the same time; and when negative, it means ζ_1 is exited earlier than ζ_2. When $\mathrm{eelag}(\zeta_1, \zeta_2)$ is positive, it means ζ_1 is exited after ζ_2 is entered; when zero, it means they are exited and entered at the same time; and when negative, it means ζ_1 is exited before ζ_2 is entered. Note that there is no need for the clock zones to intersect.

Example 3. In Example 1, $\text{enlag}(z_1, z_2) = (-1-(-3))^+ = 2^+, \text{exlag}(z_1, z_2) = 6-7 = -1$, and $\text{eelag}(z_1, z_2) = 6+(-1) = 5$, which shows that z_1 is entered later than z_2 by at most 2^+ time units, z_1 is exited earlier than z_2 by at least 1 time unit, and z_1 is exited after z_2 is entered by at most 5 time units.

For defining zone-based urgency semantics of a set of concurrent UTA, we first define *complete urgency*, which is a type of composition of UTA models.

Definition 10 (*Complete urgency*). During the composition of UTA models that obey the deadline restriction, if an urgent transition outgoing from a mode is eventually enabled, then *complete urgency* requires that it is not disabled due to time elapse in the mode or in any predecessor mode of that mode.

Definition 11 (*Urgent timed system state graph (model composition)*). Given an urgent timed system $\mathscr{S}$ with n components modeled by UTA $\mathscr{A}_i = (M_i, m_i^0, C_i, D_i, L_i, \chi_i, T_i, \Omega_i)$, where $1 \leqslant i \leqslant n$, the system model is defined as a state graph represented by $\mathscr{A}_1 \times \cdots \times \mathscr{A}_n = \mathscr{A}_{\mathscr{S}} = (M, m^0, C, D, L, \chi, T, \Omega)$, where

- $M = M_1 \times M_2 \times \cdots \times M_n$ is a finite set of system modes such that $m = m_1 m_2 \ldots m_n$ is a system mode in M corresponding to UTA modes $m_i \in M_i, 1 \leqslant i \leqslant n$,
- $m^0 = m_1^0 m_2^0 \ldots m_n^0 \in M$ is the initial system mode,
- $C = \bigcup_i C_i$,
- $D = \bigcup_i D_i$,
- $L = \bigcup_i L_i$,
- $\chi : M \mapsto B(C, D)$ gives the invariant of a system mode such that $\chi(m) = \wedge_i \chi_i(m_i)$, where $m = m_1 m_2 \ldots m_n \in M$,
- $T \subseteq M \times M$ is a set of system transitions that preserves *complete urgency* and is segregated into two types:

  1. *asynchronous transitions*: a transition $e \in T$ is asynchronous iff $\exists i, 1 \leqslant i \leqslant n, e_i \in T_i$ such that $e_i = e$, and
  2. *synchronized transitions*: a transition $e \in T$ is synchronous iff $\exists i, j, 1 \leqslant i \neq j \leqslant n, e_i \in T_i, e_j \in T_j$ such that $\psi_i(e_i) = \psi_j(e_j) = l \in L_i \cap L_j$, $e \in T$ is the synchronization of ETA transitions e_i and e_j, denoted as $e = e_i \| e_j$, with conjuncted triggering conditions and union of all transitions assignments as defined later in this definition.

- $\Omega : T \mapsto \langle L, B(\bigcup_i C_i, \bigcup_i D_i), \{\lambda, \delta, \varepsilon\}, 2^{\bigcup_i C_i \cup (\bigcup_i D_i \times \mathscr{N})} \rangle$ is a description of T. For ease of notations, we use the following short forms: $\Omega(t) = \langle \psi(t), \tau(t), \mu(t), \rho(t) \rangle$, where $t \in T$. Ω is defined as follows:
  - $\psi : T \mapsto L$ associates a synchronization label with a system transition such that $\psi(e) = \psi_i(e)$ for $e = e_i$ or $e = e_i \| e_j$,
  - $\tau : T \mapsto B(\bigcup_i C_i, \bigcup_i D_i)$ gives the triggering condition of a system transition such that $\tau(e) = \tau_i(e_i)$ for an asynchronous transition $e = e_i$ and $\tau(e) = \tau_i(e_i) \wedge \tau_j(e_j)$ for a synchronized transition $e = e_i \| e_j$,
  - $\mu : T \mapsto \{\lambda, \delta, \varepsilon\}$ gives the urgency type of a system transition, such that $\mu(e) = \mu_i(e_i)$ for an asynchronous transition $e = e_i$ and $\mu(e)$ is as defined in Table 1 for a synchronized transition $e = e_i \| e_j$,
  - $\rho : T \mapsto \gamma$, where γ is a partial function mapping $\bigcup_i (C_i \cup D_i)$ to $\mathscr{N}$, gives the assignments on a system transition such that $\rho(e) = \rho_i(e_i)$ for an asynchronous transition $e = e_i$ and $\rho(e) = \rho_i(e_i) \cup \rho_j(e_j)$ for a synchronized transition $e = e_i \| e_j$.

Table 1
Urgency resolution for synchronized transitions.

$\mu_i(e_i)$	λ	λ	δ	λ	δ	ε
$\mu_j(e_j)$	λ	δ	δ	ε	ε	ε
$\mu(e)$	λ	δ	δ	ε	ε	ε

Complete urgency, in other words, also means that there is no urgent transition that could have been enabled, but was disabled due to time elapse in an ancestor mode of the source mode of that urgent transition. For example, suppose there is a lazy transition with trigger $x \geqslant 0$ going from mode m to m' and suppose m' has an outgoing delayable transition with deadline $x \leqslant 8$, then we must take the lazy transition and enter m' no later than $x = 8$, otherwise complete urgency restriction will not hold.

One might wonder if complete urgency is a very strict restriction. We argue that though complete urgency is a restriction; however, real system models often obey this restriction because if deadline violations are to be handled then they are explicitly modeled by a user. All the examples in Section 5.4 satisfy the complete urgency restriction. Complete urgency holds for delayable transitions because the violation of a deadline (reaching the source mode after the deadline of the transition has passed) can always be modeled as a branching transition in some predecessor mode of the source mode. Fig. 2 shows how the satisfaction and the violation of a deadline can be modeled under the complete urgency. Thus, we need only consider the satisfaction of the deadline in the source mode, which basically is complete urgency. Further, complete urgency also holds for eager transitions because eagerness is similar to the *as soon as possible* semantics, which means no behavior beyond the enabling time of the eager transition need to be considered and this is what complete urgency covers.

In Section 4, our zone-based urgency semantics will be proposed on the system model that satisfies both the complete urgency and the deadline restrictions, as described above. For simplicity, henceforth we will assume the deadline restriction always holds, and simply talk about complete urgency.

Our model checking procedures for urgent timed automata are implemented in the *State-Graph Manipulators* (SGM) model checker (Hsiung and Wang, 1998; Wang and Hsiung, 2002), which is a high-level compositional model checker for real-time systems. SGM model checks extended timed automata against temporal logic properties. For verifying an urgent timed system modeled by a set of urgent timed automata, the system properties can be specified in some *temporal logic*. SGM uses the *Computation Tree Logic* (Alur et al., 1990) as its logical formalism.

Definition 12 (*Computation tree logic (CTL)*). A *computation tree logic* formula has the following syntax: $\phi ::= \eta | EG\phi' | E\phi' U \phi'' | \neq \phi' | \phi' \vee \phi''$, where η is a mode predicate, ϕ' and ϕ'' are CTL formulae. $EG\phi'$ means there is a computation from the current state, along

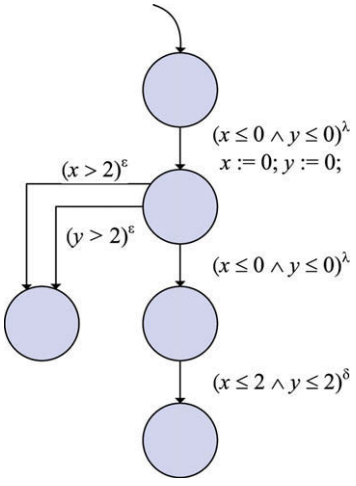


Fig. 2. Modeling under complete urgency.

which ϕ' is always true. $E\phi' U \phi''$ means there exists a computation from the current state, along which ϕ' is true until ϕ'' becomes true. Shorthands like $EF, AF, AG, AU, \wedge,$ and $\rightarrow$ are standard (Henzinger et al., 1992).

Definition 13 (*Model checking*). Given an urgent timed system state graph $\mathscr{A}_{\mathscr{S}}$ that represents an urgent timed system $\mathscr{S}$, a system state (m, v), and a CTL formula, ϕ, expressing some desired specification, model checking verifies if $\mathscr{A}_{\mathscr{S}}$ satisfies ϕ at (m, v), denoted by $(\mathscr{A}_{\mathscr{S}}, m, v) \models \phi$. Model checking can be either explicit using a labeling algorithm on the system state graph or symbolic using a fixpoint algorithm on BDDs and DBMs.

4. Model checking urgent timed systems

Our target problem is to model and verify urgent timed systems such as real-time embedded systems. A set of urgent timed automata is used to model such a system and model checking is used to verify if the urgent timed system state graph, obtained by merging the set of UTA, satisfies user-given CTL properties. In this section, we will propose solutions to the issues that were introduced in Section 1. A precise definition of the semantics of urgent timed automaton will be given in Section 4.1. A major extension to the conventional semantics involves setting the upper bound of clock zones and its implementation using DBMs, which is called zone capping, will be covered in Section 4.2. In Section 4.3, we will show how to cap mode zones for different compositions of urgent outgoing transitions.

4.1. Absolute semantics of urgent timed automata

As defined in Definition 1, the urgency of a transition t is classified as *lazy* $(\mu_i(t) = \lambda)$, *delayable* $(\mu_i(t) = \delta)$, and *eager* $(\mu_i(t) = \varepsilon)$. According to the definition of these transition urgencies in Definition 11, we formalize their *absolute* semantics in this subsection. As shown in Fig. 3, consider a non-initial mode $m \in M_i$ of an urgent timed automaton $\mathscr{A}_i = (M_i, m_i^0, C_i, D_i, L_i, \chi_i, T_i, \Omega_i)$, which has an incoming mode transition t_{in} and an outgoing mode transition t_{out}. For simplicity, we first assume that m has a single incoming and a single outgoing transition. From the two computation runs in Fig. 3, we can make the following observations, where each state $s_j = (m_j, v_j)$.

State sequence $\langle s_0, \ldots, s_{k-1} \rangle$ leads to zone $z = (m, \zeta_m, \beta_m)$. s_k is the first reachable state in zone z, that is, $m_j \neq m, \forall j, 0 \leqslant j < k$ and $m_k = m$. s_e is the first reachable state in zone z in which transition t_{out} is enabled. Transition t_{out} may be taken anytime starting from state s_e before it is disabled, as in run π', and s_d is the first reachable state in zone z in which transition t_{out} is disabled before it is taken, as in π.

In Fig. 3, there are two kinds of computation runs π and π' as follows.

- Transition t_{out} is enabled and then disabled, without being taken: $\pi = \langle s_0 \xrightarrow{t_0} s_1 \cdots \xrightarrow{t_{in}} s_k \xrightarrow{t_k} s_{k+1} \xrightarrow{t_{k+1}} \cdots s_e \xrightarrow{t_e} s_{e+1} \xrightarrow{t_{e+1}} \cdots s_d \xrightarrow{t_d} s_{d+1} \xrightarrow{t_{d+1}} \cdots \rangle$, where starting from state s_k, all states are in the zone $z = (m, \zeta_m, \beta_m)$, that is, $v_j(C_i) \rightarrow \zeta_m$ and $v_j(D_i) \rightarrow \beta_m$ for all $j \geqslant k$. Let Π be set of all such computation runs π, where t_{out} is never taken.
- Transition t_{out} is enabled and is taken before being disabled: $\pi' = \langle s_0 \xrightarrow{t_0} s_1 \cdots \xrightarrow{t_{in}} s_k \xrightarrow{t_k} s_{k+1} \xrightarrow{t_{k+1}} \cdots s_e \xrightarrow{t_e} s_{e+1} \xrightarrow{t_{e+1}} \cdots s_t \xrightarrow{t_{out}} \cdots \rangle$, where the states $s_k, \ldots, s_{t-1}$ are in the zone $z = (m, \zeta_m, \beta_m)$. Let Π' be the set of all such computation runs, where t_{out} is enabled and taken before being disabled.

Since there are infinite number of states in which transition t_{out} may be enabled and taken, let us consider clock regions as defined in Definition 4. Given a region R, let $\Pi'_R \subseteq \Pi'$ be the subset of computation runs where t_{out} is taken in region R, that is, $s_t \in R$.

The semantics of the transition behavior differ according to the urgency $\mu_i(t_{out})$ associated with the transition as follows.

- Lazy transition (λ): If t_{out} is a lazy transition, that is, $\mu_i(t_{out}) = \lambda$, then the set of reachable computation runs that passes through mode m is $\Pi(m, t_{out}) = \Pi \cup \Pi'$, which means all runs, where t_{out} after being enabled is either taken or not taken before being disabled, are reachable.
- Delayable transition (δ): If t_{out} is a delayable transition, that is, $\mu_i(t_{out}) = \delta$, then the set of runs through mode m is $\Pi(m, t_{out}) = \Pi'$, which means all runs where t_{out} after being enabled is taken no later than being disabled.
- Eager transition (ε): If t_{out} is an eager transition, that is, $\mu_i(t_{out}) = \varepsilon$, then the set of reachable computation runs that passes through mode m is $\Pi(m, t_{out}) = \bigcup_R \Pi'_R, R = [s_e]$, where $[s_e]$ is the region in which t_{out} is enabled.

From above since $\bigcup_R \Pi'_R \subseteq \Pi' \subseteq \Pi \cup \Pi'$, we can observe that eager transitions are the most restrictive ones and lazy transitions are the most lenient ones, while the delayable transitions are in-between.

4.2. Capping zones

The absolute semantics for urgency, as described in Section 4.1, are not practical for model checking. We propose a novel *zone-based urgency semantics* that enforces urgency based on the symbolic representation of clock constraints, namely clock zones. As proved later in Section 5.2, zone-based urgency semantics give the same model checking results as the TAD or TAUT semantics. We will define a *zone capping* operation

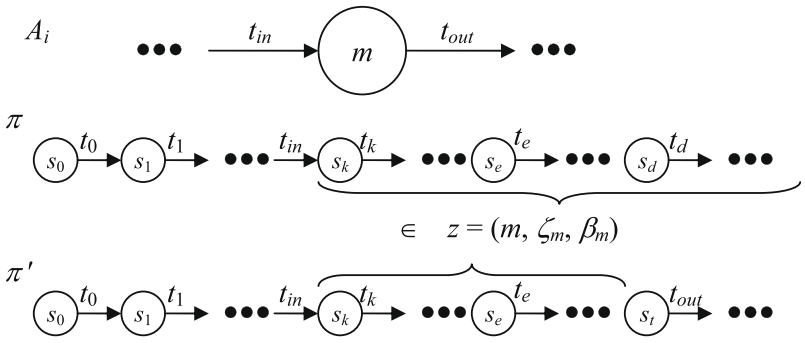


Fig. 3. Computation runs.

on the zones associated with modes that have urgent outgoing transitions in an urgent timed system state graph. Under the complete urgency restriction, our method is completely compatible with the conventional TA model checking, thus any model checker can be easily extended to model checking urgent timed systems, without the need for zone partitioning (proved in Section 5.3).

When a system is in a zone $z = (m, \zeta_m, \beta_m)$ such that the clock zone has no upper bound, that is, $\Delta_m(i,0) = (<, \infty)$ for all clocks $x_i \in C$, or the upper bound allows the system to stay in the mode m beyond that allowed by a delayable or an eager transition outgoing from the mode m, then we need to restrict the upper bound of the zone. This kind of restriction is called *zone capping*. By capping a zone, a system is forced to exit the mode before the upper bound is violated due to time elapse, otherwise the behavior of the system will be undefined. Before defining zone capping, since the upper bounds for delayable and eager transitions are different, we need to first define the *subzones* that will be used as upper bounds for zone capping. In the following, we define earliest subzone and final subzone that correspond to the upper bounds for taking the eager transition and the delayable transition, respectively.

Definition 14 (*Earliest subzone).* Given a clock zone ζ, represented by a DBM Δ, the earliest subzone $\mathrm{ESub}(\zeta)$ is a subspace of ζ such that the DBM Δ_e representing $\mathrm{ESub}(\zeta)$ is defined as follows:

$$\Delta_e(i,0) = \begin{cases} (\leqslant, c) & \text{if } \Delta(0,i) = (\leqslant, -c), \quad i > 0, \\ (<, c+1) & \text{if } \Delta(0,i) = (<, -c), \quad i > 0, \end{cases}$$
$$\Delta_e(0,j) = \Delta(0,j), \quad j \geqslant 0,$$
$$\Delta_e(i,j) = (\leqslant, 0), \quad i > 0,\ j > 0.$$

Definition 15 (*Final subzone).* Given a clock zone ζ, represented by a DBM Δ, the final subzone $\mathrm{FSub}(\zeta)$ is a subspace of ζ such that the DBM Δ_f representing $\mathrm{FSub}(\zeta)$ is defined as follows:

$$\Delta_f(0,j) = \begin{cases} (\leqslant, -c) & \text{if } \Delta(j,0) = (\leqslant, c), \quad j > 0, \\ (<, -c+1) & \text{if } \Delta(j,0) = (<, c), \quad j > 0, \\ \Delta(0,j) & \text{if } \Delta(j,0) = (<, \infty), \quad j > 0, \end{cases}$$
$$\Delta_f(i,0) = \Delta(i,0), \quad i \geqslant 0,$$
$$\Delta_f(i,j) = (\leqslant, 0), \quad i > 0,\ j > 0.$$

Note that the DBMs for both ESub and FSub must be canonicalized before using them in further operations, which may change the DBM elements besides the ones given in the above definitions. A zone is called a *subzone* when it is the earliest subzone or the final subzone for some clock zone. Zone capping can be defined using a subzone as upper bound for a clock zone.

Definition 16 (*Zone capping).* Given a clock zone ζ and a subzone ζ_s, represented respectively by DBMs Δ and Δ_s, the zone ζ can be capped by ζ_s into a new zone denoted by $\mathrm{ZCap}(\zeta, \zeta_s)$ which is defined by its DBM $\Delta_{(\zeta,\zeta_s)}$ as follows:

$$\begin{aligned} &\Delta_{(\zeta,\zeta_s)}(i,0) = \min(\Delta(i,0), \Delta_s(i,0)), \quad \forall i, \\ &\Delta_{(\zeta,\zeta_s)}(i,j) = \Delta(i,j), \quad \forall j \neq 0. \end{aligned} \tag{1}$$

Examples and intuitive illustrations of earliest subzone, final subzone, and zone capping are given in Fig. 4. It must be noted here that after zone capping, we need to canonicalize the DBM $\Delta_{(\zeta,\zeta_s)}$ before they can be used for further processing in model checking.

4.3. Enforcing urgencies

We now show how urgency is enforced in UTA by applying the zone capping operation using the earliest and final subzones. Given a mode m with zone (m, ζ_m, β_m) and outgoing transitions classified into three sets: a set of p delayable transitions $T_d^m = \{t_d | \mu(t_d) = \delta\}$, a set of q eager transitions $T_e^m = \{t_e | \mu(t_e) = \varepsilon\}$, and zero or more lazy transitions, urgency is enforced by modifying the clock zone ζ_m into a newly capped zone $\zeta'(m)$ as shown in Eqs. (2)–(5), where $p, q \in \mathcal{N}, p = |T_d^m|$, and $q = |T_e^m|$. If there is no urgent transition ($p = q = 0$), then the mode clock zone ζ_m is not modified, which is shown in Eq. (2):

$$\text{Case } p = q = 0: \quad \zeta'_m = \zeta_m, \tag{2}$$

$$\begin{aligned} &\text{Case } p > 0,\ q = 0: \\ &\quad \zeta'_m = \mathrm{ZCap}(\zeta_m, \mathrm{FSub}(\zeta_m \cap \zeta_{\tau(t_d)})), \\ &\quad \text{for some } t_d \in \{t_d | \mathrm{exlag}(\zeta_m \cap \zeta_{\tau(t_d)}, \zeta_m \cap \zeta_{\tau(t'_d)}) \leqslant 0, \forall t'_d \in T_d^m\}, \end{aligned} \tag{3}$$

$$\begin{aligned} &\text{Case } p = 0,\ q > 0: \\ &\quad \zeta'_m = \mathrm{ZCap}(\zeta_m, \mathrm{ESub}(\zeta_m \cap \zeta_{\tau(t_e)})), \\ &\quad \text{for some } t_e \in \{t_e | \mathrm{enlag}(\zeta_m \cap \zeta_{\tau(t_e)}, \zeta_m \cap \zeta_{\tau(t'_e)}) \leqslant 0, \forall t'_e \in T_e^m\}, \end{aligned} \tag{4}$$

$$\begin{aligned} &\text{Case } p > 0,\ q > 0: \\ &\quad \zeta'_m = \begin{cases} \mathrm{ZCap}(\zeta_m, \mathrm{ESub}(\zeta_m \cap \zeta_{\tau(t_e)})) \\ \quad \text{if } \mathrm{eelag}(\zeta_m \cap \zeta_{\tau(t_d)}, \zeta_m \cap \zeta_{\tau(t_e)}) \geqslant 0, \\ \mathrm{ZCap}(\zeta_m, \mathrm{FSub}(\zeta_m \cap \zeta_{\tau(t_d)})) \\ \quad \text{otherwise}, \end{cases} \\ &\quad \text{where } t_d \in \{t_d | \mathrm{exlag}(\zeta_m \cap \zeta_{\tau(t_d)}, \zeta_m \cap \zeta_{\tau(t'_d)}) \leqslant 0, \forall t'_d \in T_d^m\} \\ &\quad \text{and } t_e \in \{t_e | \mathrm{enlag}(\zeta_m \cap \zeta_{\tau(t_e)}, \zeta_m \cap \zeta_{\tau(t'_e)}) \leqslant 0, \forall t'_e \in T_e^m\}. \end{aligned} \tag{5}$$

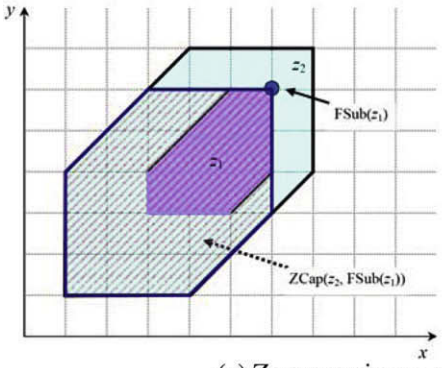
The intuitions behind the zone capping definitions are as follows. When there is only one delayable transition t_d, we need to force the system to leave mode m no later than the last subzone (before t_d becomes disabled) and this is the final subzone (FSub) of the intersection of ζ_m and $\zeta_{\tau(t_d)}$, which is $\mathrm{FSub}(\zeta_m \cap \zeta_{\tau(t_d)})$. This final subzone is used to cap ζ_m. However, for multiple delayable transitions, we need to select the one that becomes disabled the earliest, which is the one with non-positive zone exit lag with all other delayable transitions in T_d^m. This case is shown in Eq. (3).

When there is only one eager transition t_e, we need to force the system to leave mode m no later than the first subzone (when t_e becomes enabled), which is the earliest subzone (ESub) of the intersection of ζ_m and $\zeta_{\tau(t_e)}$, that is, $\mathrm{ESub}(\zeta_m \cap \zeta_{\tau(t_e)})$. This earliest subzone is used to cap ζ_m. However, for multiple eager transitions, we need to select the one that becomes enabled the earliest, which is the one with non-positive zone entry lag with all other eager transitions in T_e^m. This case is shown in Eq. (4).

When there are delayable and eager transitions, in order to ensure time-reactivity or to avoid timelocks, we need to find a transition that is either disabled or enabled the earliest. The condition $\mathrm{eelag}(\zeta_m \cap \zeta_{\tau(t_d)}, \zeta_m \cap \zeta_{\tau(t_e)}) \geqslant 0$ implies the earliest enabled eager transition t_e becomes enabled not later than the earliest disabled delayable transition t_d. If the condition holds, we use ESub to cap ζ_m. Otherwise, we use FSub. This case is shown in Eq. (5).

As an illustration of the various definitions used in zone capping, Fig. 4 shows the results of computing the final subzone and the earliest subzone of a zone z_1 and then these two subzones are used to cap another zone z_2. Fig. 4b clearly shows how the zone-based urgency semantics differs from the absolute urgency semantics in TAUT because the eager transition is allowed to be taken as late as in the earliest subzone of z_1, while it must be taken by $(x \leqslant 3 + l) \wedge (y \leqslant 3 + l)$, where l is the deadline parameter. Note that TAD does not allow left open time intervals, hence cannot handle this case.

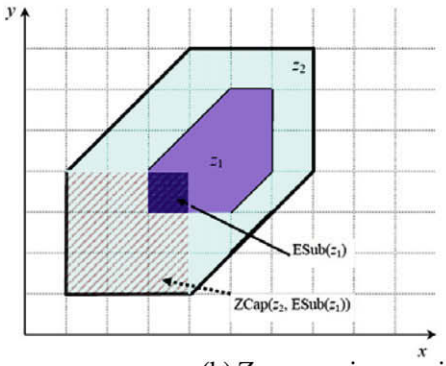
$$z_2: \begin{bmatrix} \le 0 & \le -1 & \le -1 \\ \le 7 & \le 0 & \le 3 \\ \le 7 & \le 3 & \le 0 \end{bmatrix} \qquad z_1: \begin{bmatrix} \le 0 & < -3 & < -3 \\ \le 6 & \le 0 & \le 2 \\ \le 6 & \le 1 & \le 0 \end{bmatrix}$$



$$\text{FSub}(z_1): \begin{bmatrix} \le 0 & \le -6 & \le -6 \\ \le 6 & \le 0 & \le 0 \\ \le 6 & \le 0 & \le 0 \end{bmatrix}$$

$$\text{ZCap}(z_2, \text{FSub}(z_1)): \begin{bmatrix} \le 0 & \le -1 & \le -1 \\ \le 6 & \le 0 & \le 3 \\ \le 6 & \le 3 & \le 0 \end{bmatrix}$$

(a) Zone capping z_2 with final subzone FSub(z_1)



$$\text{ESub}(z_1): \begin{bmatrix} \le 0 & < -3 & < -3 \\ < 4 & \le 0 & \le 0 \\ < 4 & \le 0 & \le 0 \end{bmatrix}$$

$$\text{ZCap}(z_2, \text{ESub}(z_1)): \begin{bmatrix} \le 0 & \le -1 & \le -1 \\ < 4 & \le 0 & \le 3 \\ < 4 & \le 3 & \le 0 \end{bmatrix}$$

(b) Zone capping z_2 with earliest subzone ESub(z_1)

Fig. 4. Illustration of zone capping.

5. Implementation, analysis, and application examples

The proposed method for model checking urgent timed systems modeled by UTA has been implemented in the *State-Graph Manipulators* (SGM) model checker (Wang and Hsiung, 2002), which is a high-level compositional model checker for real-time systems. The implementation of the algorithm for processing urgencies in a state graph using zone capping is described in Section 5.1. UTA can be input to SGM and model checked automatically against user-specified CTL properties. Theoretical results such as semantics equivalence and time reactivity are proved in Section 5.2. The proof of complete urgency with deadline restriction being a sufficient and necessary condition for avoiding zone partitioning is given in Section 5.3. It is also shown how zone capping preserves complete urgency and hence avoids zone partitioning, while satisfying all deadlines. Several application examples given in Section 5.4 show how our approach is superior to the existing state-of-the-art methods and tools.

This is the first known implementation and handling of urgency for timed automata in a CTL model checker itself. Other tools such as UPPAAL does not support the urgency semantics described in this work, while the IF toolset (Bozga et al., 1999) supports modeling of urgency, exhaustive simulation, and model checking using observers only. Labeled transition systems (LTS) generated by IF could blow up in size and even not terminate, as detailed in Section 5.4.

5.1. Urgency processing algorithm

The algorithm for processing urgency assumes that we already have a system state graph, which represents the concurrent behavior of a set of UTA and can be obtained by the *merge* manipulator in SGM. The urgency processing algorithm as shown in Algorithm 1 works as follows. For each mode m in M, we count the number of delayable and eager outgoing transitions (Steps 3 and 4). Then, using the zone exit lag computation between two delayable transitions, we find the delayable transition t_{minx} that will become disabled the earliest (Steps 5–12). Similarly, using the zone entry lag computation between two eager transitions, we also find the eager transition t_{mine} that becomes enabled the earliest (Steps 13–20). If both t_{minx} and t_{mine} exist, then we compute the zone exit/entry lag between them. If the lag is non negative, it means $Z_{t_{mine}}$ is entered as late as when $Z_{t_{minx}}$ is exited, in which case the mode zone is capped using t_{mine}. Otherwise, the mode zone is capped using t_{minx} (Steps 21–31). The zone operations `ZCap`, `ESub`, `FSub`, `Intersect` are used to modify the mode clock zone Z_m according to Eqs. (2)–(5).

Algorithm 1. Process urgencies in a state graph

input State Graph: G//Definition 11:
$G = (M, m^0, C, D, L, \chi, T, \Omega)$
1 $t_{minx} = t_{mine} = NULL$;
2 **for** each mode $m \in M$
3 $num\_delayable = count\_delayable(m)$;
4 $num\_eager = count\_eager(m)$;
5 **if** $num\_delayable > 0$ **then**
6 $t_{minx} = t_{d_1}$;
7 **for** each delayable transition $t_d \neq t_{minx}$ outgoing from m **do**
8 **if** `exlag(Intersect)`$(Z_m, Z_{t_{minx}})$, `Intersect` $(Z_m, Z_{t_d}) > 0$) then
9 $t_{minx} = t_d$;
10 **end**

```
11      end
12    end
13    if num_eager > 0 then
14      t_mine = t_e1;
15      for each eager transition t_e ≠ t_mine outgoing from m do
16        if (enlag(Intersect(Z_m, Z_t_mine),
          Intersect(Z_m, Z_t_e)) > 0) then
17          t_mine = t_e;
18        end
19      end
20    end
21    if num_delayable > 0 then
22      if num_eager > 0 and
        eelag(Intersect(Z_m, Z_t_minx), Intersect(Z_m, Z_t_mine)) ⩾ 0
        then
23        Z_m = ZCap(Z_m, ESub(Intersect(Z_m, Z_t_mine)));
24      end
25      else
26        Z_m = ZCap(Z_m, FSub(Intersect(Z_m, Z_t_minx)));
27      end
28    end
29    else if num_eager > 0 then
30      Z_m = ZCap(Z_m, ESub(Intersect(Z_m, Z_t_mine)));
31    end
32  end
33  return 0;//Result: stategraph G is modified through zone
    capping
```

For an urgent system state graph, the complexity of the algorithm is $\mathcal{O}(|M| \times |T| \times |C|^2)$. For each mode in M, we need to calculate the zone exit lag for each outgoing delayable transition and the zone entry lag for each outgoing eager transition. The zone lag computations have $\mathcal{O}(|C|)$ time complexity. However, the zone capping, the subzone computation, and the intersection operations all require $\mathcal{O}(|C|^2)$ time complexity. Hence, the complexity of the algorithm is $\mathcal{O}(|M| \times |T| \times |C|^2)$. Note that the state graph must be pruned after zone capping because some outgoing transitions may become invalid. As a result, the graph becomes smaller in size and thus accelerates model checking.

5.2. Theoretical analysis

We give some theoretical results pertaining to our proposed zone capping method for enforcing zone-based urgency semantics. We first state that zone-based urgency semantics is equivalent to absolute urgency semantics as advocated by the previous work on urgency modeling (Barbuti and Tesei, 2004; Bornot et al., 1997; Gebremichael and Vaandrager, 2005). Next, we state that our method preserves time-reactivity for urgent timed automata.

Theorem 1 (The same model-checking results). *The zone-based urgency semantics of UTA and the absolute urgency semantics of TAD and TAUT give the same model checking results, whenever the models allow urgency semantics.*

Proof. The proposed zone-based urgency semantics, as given in Eqs. (2)–(5), uses the operations of earliest subzone, final subzone, and zone capping from Definitions 14–16, respectively. One can easily observe that the mode zone capping operation is performed *individually* for each clock and each clock difference. This is valid because the clock zones are geometrically convex polytopes and capping amounts to restricting its bounding cells (hyperplanes) individually. Hence, we need only discuss the proof for a basic clock constraint such as $x \sim c$ or $x - y \sim c$, where x, y are clocks, c is an integer, and $\sim \in \{<, \leqslant, \geqslant, >\}$.

For delayable transitions, the zone-based urgency semantics described in Section 4.1 includes only those computation runs that enforce all enabled delayable transitions to be taken before being disabled, that is, mode zones are bounded by the final subzones of the outgoing delayable transition that is disabled earliest. From the definition of final subzone (Definition 15), one can observe that all upper bounds are not changed, which shows that the zone-based urgency semantics are the same as the absolute urgency semantics for right open ($x < c$) and right closed ($x \leqslant c$) time intervals.

For eager transitions, the zone-based urgency semantics described in Section 4.1 includes only those computation runs that enforce an eager transition to be taken within the earliest subzone in which it is enabled. For a left closed time interval, $x \geqslant c$, the zone capping is performed at $x = c$, which is semantically equivalent to the absolute urgency semantics. However, for a left open time interval, $x > c$, there is significant difference among UTA, TAD, and TAUT urgency semantics. UTA urgency semantics requires an eager transition with a left open time interval $x > c$ to be taken within the zone that is bounded by $(c, c+1)$ for clock x, that is, the eager transition can be taken as early as $x = c + \delta$ or as late as $x = c + 1 - \delta$, where $\delta > 0$ is an infinitesimally small amount of time. As far as TAD urgency semantics are concerned, left open time intervals are not allowed on eager transitions. TAUT urgency semantics require an eager transition with a left open time interval $x > c$ to be taken within the zone that is bounded by $(c, c+l)$ for clock x, where $l \in \mathcal{Q}_{>0}$ is the deadline parameter associated with the TAUT. Since l is made as small as possible in the TAUT urgency semantics, $l < 1$ always holds. Because $(c, c+l) \subset (c, c+1)$, the zone in which an eager transition is taken in the TAUT semantics is a subzone of the zone in which the same transition is taken in the UTA semantics. This means that all computation runs of a TAUT are also runs of UTA. The UTA semantics include some more runs that allow the eager transition to be taken in the zone with $x \in [c+l, c+1)$. However, it is well-known that the CTL model checking results are the same for all states in the same region (Alur and Dill, 1994). Since all states in $(c, c+l)$ and in $[c+l, c+1)$ belong to the same region $(c, c+1)$, the model checking results will be the same for UTA and TAUT semantics. Notationally, $(\mathscr{A}_{UTA}, m) \models \phi \iff (\mathscr{A}_{TAUT}, m) \models \phi$ and $(\mathscr{A}_{UTA}, m') \models \phi \iff (\mathscr{A}_{TAUT}, m') \models \phi$, where $\mathscr{A}_{UTA}$ and $\mathscr{A}_{TAUT}$ are the same model under UTA and TAUT semantics, respectively, m and $m\prime$ are respectively the source and destination modes of an eager transition with a left open time interval, and ϕ is a CTL property.

We have proved that the zone-based urgency semantics of the proposed UTA model is the same as the absolute urgency semantics of TAD and TAUT for right open, right closed, and left closed time intervals. For left open time intervals, we have also proved that the zone-based urgency semantics of UTA and the urgency semantics of TAUT give the same model checking results. Thus, the model checking results are the same for both semantics. □

Theorem 2 (Time-Reactivity). *The proposed zone capping method for enforcing zone-based urgency semantics preserves time-reactivity for urgent timed automata.*

Proof. We prove by contradiction. Suppose UTA is not time reactive. By Definition 3, there exists at least one state s with time lock, which means time progress is not possible in s at some time t. Under the zone-based urgency semantics, modes without urgent transitions do not have upper bounded clock zones, which means time progress is always possible in such modes. For a mode with one or more urgent transitions, the clock zone is capped by either a final subzone or an earliest subzone. In a final subzone, $\mathrm{FSub}(\zeta_m \cap \zeta_{\tau(t_d)})$, the delayable transition t_d is enabled and is taken before being disabled. In an earliest subzone, $\mathrm{ESub}(\zeta_m \cap \zeta_{\tau(t_e)})$, the

eager transition t_e is enabled and taken. Thus, the zone capping operation always ensures that at least one transition is enabled at the time t when time progress is stopped in state s. However, this results in a contradiction because according to Definition 3, no transition is enabled when time progress is stopped in s. Thus, there is no such state s with time lock in UTA. Hence, the proposed zone capping method for enforcing zone-based urgency semantics preserves time-reactivity for UTA. □

5.3. Zone capping vs. zone partitioning

It is a common belief that model checking timed systems with urgency requires *zone partitioning* (Gebremichael and Vaandrager, 2005) because model checking requires clock zones to be convex, while the urgency processing (satisfaction) of deadline predicates generates non-convex zones due to disjunction ($\neq \bigvee_i d_i$). Zone partitioning transforms a non-convex zone into a set of convex zones. It was suggested in a previous work (Gebremichael and Vaandrager, 2005) that we should restrict clock zones in urgency predicates to avoid generating non-convex time progress predicates (zones). It was neither shown how the restrictions should be performed nor how complex the process could be. For urgent timed systems with complete urgency, it is shown here how the proposed zone capping method avoids non-convex zones and thus neither needs zone partitioning nor any restrictions on urgency predicates.

We use a simple example to illustrate our claims and then generalize upon the results. Given three concurrently enabled transitions as shown in Fig. 5, two of which are delayable transitions and one lazy transition. The set of deadline predicates of the two delayable transitions is $\{x = 2 \wedge y = 2, x = 3 \wedge y = 3\}$. To satisfy the deadline predicates, zone partitioning constructs five disjoint partitions of the 2-clock $\mathscr{R}_{\geqslant 0} \times \mathscr{R}_{\geqslant 0}$ zone space, for example, $\{x \leqslant 2 \wedge y \leqslant 2, 2 < x \leqslant 3 \wedge y \leqslant 3, 2 < y \leqslant 3 \wedge x \leqslant 2, 3 < y, x > 3 \wedge y \leqslant 3\}$ is one possible way of partitioning and is illustrated in Fig. 6.

To satisfy the same set of deadline predicates, zone capping bounds mode zones into three types of zones, namely $\{x \leqslant 2 \wedge y \leqslant 2, x \leqslant 3 \wedge y \leqslant 3, x \geqslant 0 \wedge y \geqslant 0\}$. The urgent timed system state graph of the three concurrent transitions in Fig. 5 is shown in Fig. 7, where the mode predicates are the results of zone capping. From Theorem 2, we know that the graph is time-reactive and we can also easily observe that all deadline predicates are satisfied. This shows that zone capping allows all deadline predicates to be satisfied without zone partitioning or zone restriction.

Further, as illustrated in Fig. 8, if we construct a state graph based on zone partitioning, it has 42 modes. The state graph is much larger in size than that produced by zone capping applied to a state graph, which has only 16 modes. This shows that zone capping is not only computationally less expensive by avoiding zone partitioning, but is also more memory efficient by producing smaller state graphs. This difference in state graph sizes occurs mainly due to the different composition semantics adopted in zone capping and in zone partitioning. Under the complete urgency composition restrictions, zone capping produces smaller state graphs that are time reactive and satisfies all deadlines. The smaller state graphs, in turn, make model checking more time and memory efficient. If we also enforce the complete urgency restriction on the zone partitioning based state graph (Fig. 8), in fact, we get a graph that is similar to the one obtained by zone capping (Fig. 7).

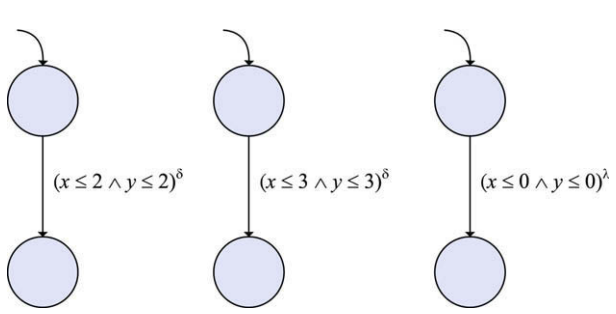


Fig. 5. Example for comparing zone partitioning and zone capping.

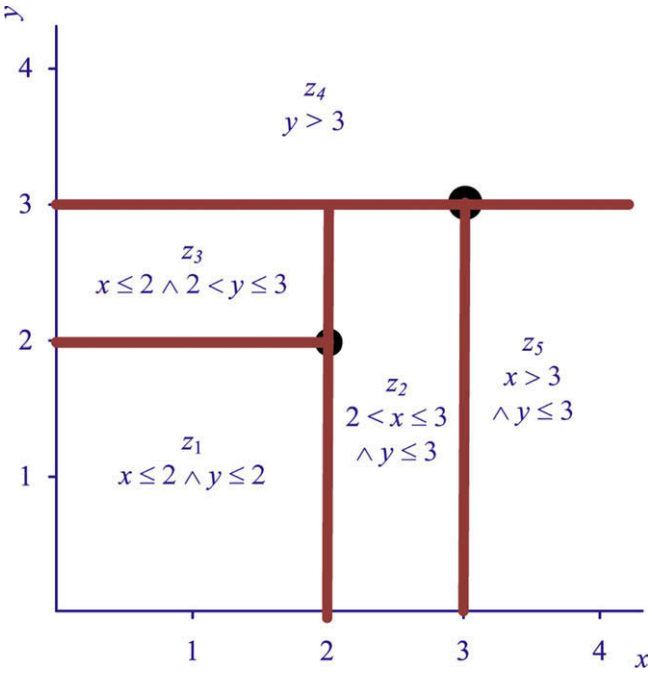


Fig. 6. Zone partitioning for deadlines $\{x = 2 \wedge y = 2, x = 3 \wedge y = 3\}$.

In general, we prove in Theorem 3 that complete urgency with deadline restriction is a necessary and sufficient condition for avoiding zone partitioning. Further, we prove in Theorem 4 that zone capping preserves and is the only operation required to preserve the complete urgency restriction and thus does not need zone partitioning. Further, it is also shown in Theorem 5 that zone capping satisfies all deadline predicates.

Theorem 3. Necessary and sufficient condition for avoiding zone partitioning

Zone partitioning can be avoided during the composition of urgent timed automata if and only if complete urgency composition restriction is maintained.

Proof. We first prove the sufficiency and then the necessity.

Sufficient: Deadline predicates can be represented by zones, thus depending on the urgency type we can use the earliest or final subzones to generate a partial order for the deadline predicates. Without loss in generality, assume $d_1 \preceq d_2 \preceq \cdots \preceq d_k$ for a mode m with k urgent outgoing transitions t_i, deadline predicates d_i, and $\preceq$ denoting the precedence relation.

To satisfy the first deadline predicate d_1, the corresponding zone within which t_1 must be taken is $z_1 = \mathrm{ZCap}(\zeta_m, \mathrm{Sub}(\zeta_m \cap \zeta_{\tau(t_1)}))$, where Sub could be ESub or FSub depending on the urgency type of t_1. Due to complete urgency, the satisfaction of all other deadline predicates $d_i, i > 1$, can be considered later in the runs starting from m, thus currently besides z_1, we need not consider the remaining part of the zone ζ_m, i.e., $\zeta_m - z_1$, which is beyond z_1 in time. Hence, we do not need zone partitioning, which is generally used to convert $\zeta_m - z_1$ into a set of convex zones. The satisfaction of $d_i, i > 1$ will be considered later in the runs from m.

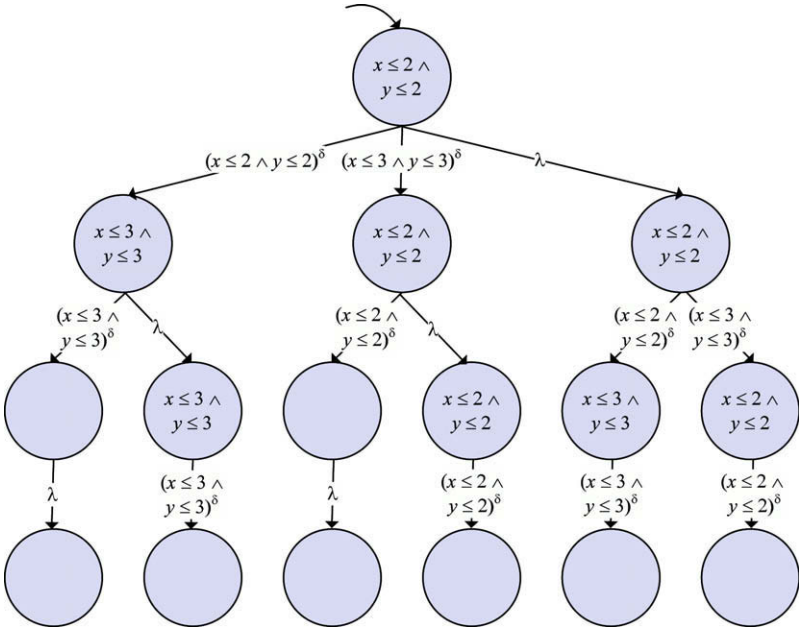


Fig. 7. State graph of comparison example after zone capping.

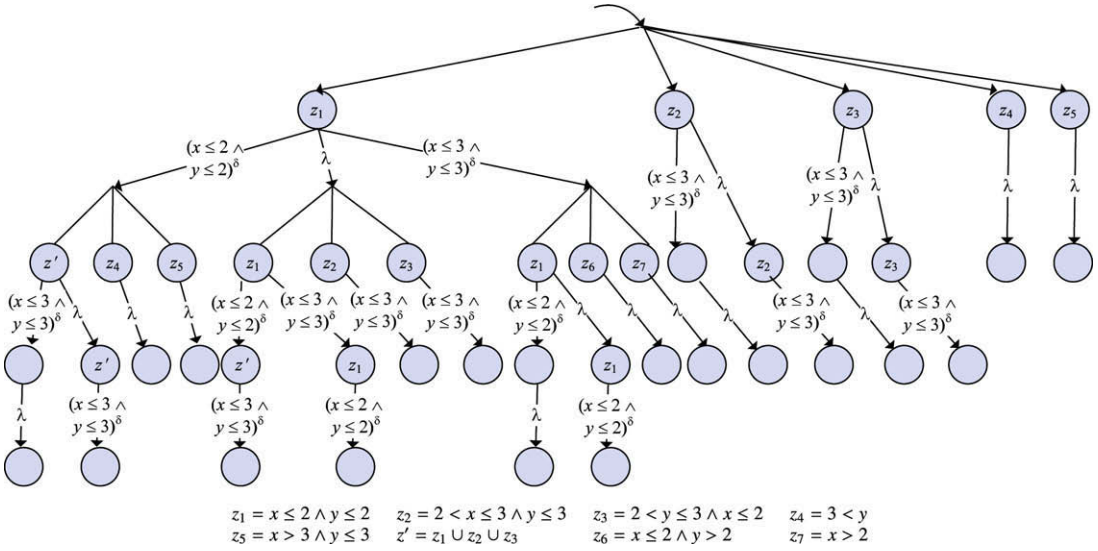


Fig. 8. State graph of comparison example using zone partitioning.

Necessary: Suppose complete urgency does not hold, that is, there exists at least one urgent transition t_i in a computation run, which was disabled due to time elapse in some mode m. We thus need to consider the possibly non-convex zone $\zeta_m - \zeta_{\tau(t_i)}$, where the zone subtraction operator - in general requires zone partitioning. Hence, complete urgency is necessary for avoiding zone partitioning. □

Theorem 4 (Complete urgency semantics). *Zone capping preserves the complete urgency restriction in an urgent timed system state graph.*

Proof. From Eqs. (2)–(5), we observe that the proof is trivial because all zone capping is performed at the deadline of the earliest disabled delayable transition or of the earliest enabled eager transition. This means that no enabled urgent transition is disabled due to time elapse in a mode, in other words, complete urgency restriction is preserved. □

Theorem 5 (Deadline satisfaction). *Zone capping method allows all deadline predicates to be satisfied.*

Proof. We prove by contradiction. Suppose there exists a deadline predicate $x \sim c$ in some urgent timed system state graph that is not satisfied after zone capping is applied to the state graph. For delayable transitions, deadline violation means the transition is enabled and is not taken before it is disabled due to time progress. From Eqs. (3) and (5) (with $p > 0$), we can see that if there is a delayable transition, the source mode zone is capped either at the final subzone of the clock trigger of the earliest disabled transition or at the earliest subzone of an eager transition if the eager transition is enabled before the earliest disabled delayable transition is disabled (the eelag() condition). This means zone capping does not allow any delayable transition's deadline to be violated due to time progress. Hence, there is no such delayable transition. For eager transitions, deadline violation means the transition is enabled and is not taken within the subzone in which it is enabled. Similarly, from

Eqs. (4) and (5) (with $q > 0$), we can observe that if there is an eager transition, the source mode zone is capped either at the earliest subzone of the clock trigger of the earliest enabled transition or at the final subzone of a delayable transition if the delayable transition is disabled before the earliest enabled eager transition is enabled. This means zone capping does not allow any eager transition's deadline to be violated due to time progress. Hence, there is also no such eager transition.

From the above analysis, we can observe that there is neither a delayable nor an eager transition, whose deadline is violated by zone capping. Hence, our assumption is false, and all deadline predicates are satisfied after zone capping is applied. □

Corollary 1. *Zone capping produces time-reactive state graphs that satisfy all deadlines, preserves complete urgency, and does not need zone partitioning.*

Proof. The results can be trivially derived from Theorems 2–5. □

5.4. Application examples

Besides SGM, there are no other known CTL model checkers that have implemented the proposed urgency semantics. The closest work that we have found is the IF toolset (Bozga et al., 1999), which performs model checking using observers. The UPPAAL model checker has implemented urgent channels and committed locations, however these cannot be used to model the urgency semantics described in this work. We first show why urgent channels and state invariants cannot model the urgency semantics. Then, we compare SGM with IF using six examples from the embedded real-time systems domain, which show that our approach as implemented in SGM always terminates and is more efficient.

We found two problems while modeling urgency in UPPAAL, as follows: (1) an eager transition with time constraints could not be modeled by an urgent channel because an urgent channel cannot be associated with any time constraint, (2) a delayable transition when forced out of a state using invariants in UPPAAL could result in timelocks, and (3) the priority on transitions and processes cannot force a model to exit a state as required by delayable or eager transitions. Out of the six examples we tried, as shown in Table 2, only one could be modeled in UPPAAL using state invariants without resulting in timelocks, namely periodic processes, because there was no communication between the periodic processes.

We compared our urgency semantics implementation in SGM with that in IF using six typical examples as summarized in Table 2, which have various combinations of eager and delayable transitions. For each example, we also specified a few CTL properties to verify the urgency semantics, which could not have been possible

Table 2
Application examples.

No.	System	$n(\|M_i\|/\|T_i\|)$	Urgency	$\sum p$	$\sum q$
1	Water sprinkler	2 (2/2, 2/2)	Single delayable	1	0
2	Heating apparatus	2 (2/2, 2/2)	Concurrent eager	0	4
3	Error checker	2 (2/3, 2/2)	Branching eager	0	4
4	Priority arbiter	3 (3/4, 3/4, 3/4)	Branching eager/delayable	1	9
5	Periodic processes	2 (3/3, 3/3)	Single eager/delayable	2	4
6	Lip synchronization (VF)	4 (2/2, 1/1, 5/5, 4/5)	Complex eager/delayable	1	10
7	Lip synchronization (SF)	3 (1/1, 2/2, 6/7)	Complex eager/delayable	2	7

n: # of UTA, VF: Video first, SF: Sound first, $\sum p$: # delayable trans, $\sum q$: # eager trans, models and input files: http://embedded.cs.ccu.edu.tw/~esl_web/Project/Ch/SGM/.

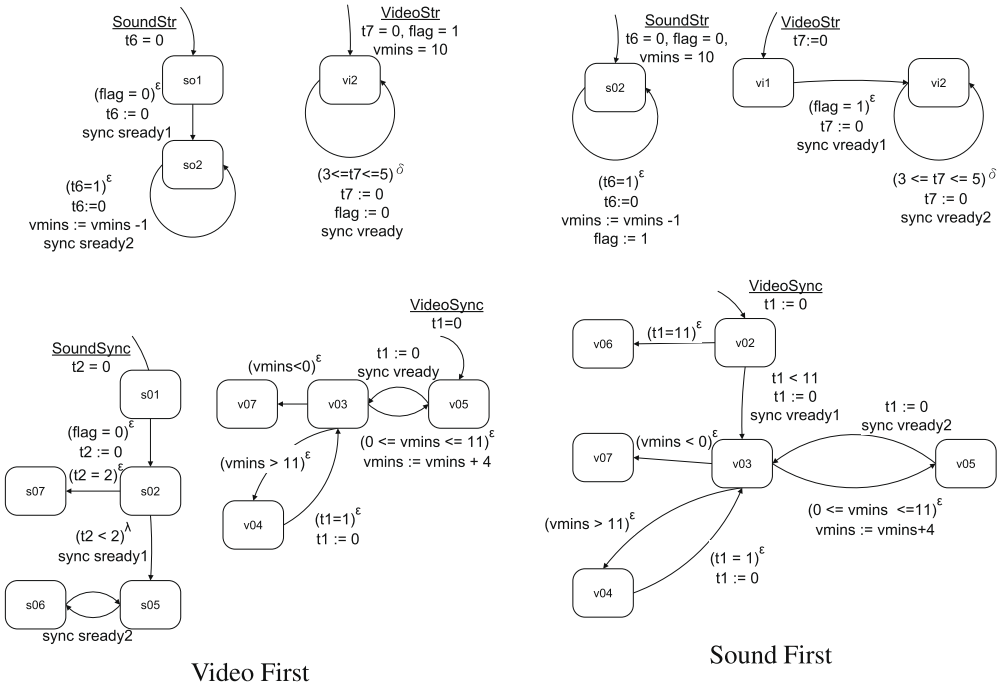


Fig. 9. Lip synchronization algorithm.

if we used UPPAAL or any other model checker without urgency semantics. Due to page-limits, only the most complex example, namely Lip Synchronization, is illustrated and described here, while the models and input files of all examples are available on the web (Table 2). A toy example is used to illustrate the differences between SGM, UPPAAL, and IF.

The Lip Synchronization algorithm was first described in the synchronous language Esterel (Stefani et al., 1992). Then specifications in a number of different formalisms were presented. The lip synchronization algorithm tries to synchronize audio and video streams as long as their arrival times are within certain time intervals. It is a typical real-time protocol for distributed multimedia systems. Bowman et al. (Bowman et al., 1998) verified the lip synchronization algorithm using the UPPAAL model checker. However, they also described the limitations in UPPAAL in detecting timelocks and in the "hand-wired" construction of timeout operators and watchdog timers, which could easily lead to timelocks. Since the lip synchronization algorithm distinguished between the initial arrival of video or sound, it was easy to partition the algorithm into two parts for verification. The models for initial arrival of video and that of sound are given in Fig. 9. The main job of lip synchronization is to compute *vmins*, the difference between the rate of the sound stream and the video stream. If *vmins* is out of some predefined range, it means that the streams are out of synchronization. We verified the following CTL property, where mode $\nu 07$ represents out of synchronization: AG($!mode(VideoSync) = \nu 07$).

In Fig. 9, we can see that it is much straightforward to model systems with urgency using UTA in SGM, compared to the construction of timeout operators and of watchdog timers, using UPPAAL committed locations and urgent channels as in (Bowman et al., 1998). As shown in Table 3, we experimented with different video input streams for the lip synchronization algorithm, by restricting the video input clock such as $t7 \in [3,4]$, that is a video frame comes every 3 to 4 time units. The results of comparing SGM with IF for the six examples are given in Table 3. We can observe that the state-graphs with urgency handling as generated by

Table 3
State graph sizes produced by SGM and IF.

Example	1		2		3		4		5		6				7			
											$t7 \in [3,4]$		$t7 \in [3,5]$		$t7 \in [3,4]$		$t7 \in [3,5]$	
Size	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$	$\|M\|$	$\|T\|$
IF[a]	31	65	12	17	8	14	N/T		341	759	N/T		N/T		N/T		N/T	
IF[b]	17	28	10	13	7	11	N/T		339	673	N/T		N/T		N/T		N/T	
SGM[c]	9	11	8	9	4	5	200	327	116	194	178	184	700	892	155	165	506	594

$t7$: Video Clock.
[a] Using DBM and DFS traversal with partial order reduction (`-dfs -po`).
[b] Using DBM and `-tf -dfs -po` parameters.
[c] No reduction, N/T: non-terminating.

Table 4
Experiment execution time (s).

Example	1	2	3	4	5	6		7	
						$t7 \in [3,4]$	$t7 \in [3,5]$	$t7 \in [3,4]$	$t7 \in [3,5]$
IF[a]	0.04	0.02	0.01	N/T	0.08	N/T	N/T	N/T	N/T
IF[b]	0.01	0.01	0.01	N/T	0.03	N/T	N/T	N/T	N/T
SGM[c]	0.02	0.02	0.01	6.16	0.27	2.14	4.55	2.58	3.36

$t7$: Video clock.
[a] Using DBM and DFS traversal with partial order reduction (`-dfs -po`).
[b] Using DBM and `-tf -dfs -po` parameters.
[c] No reduction, N/T: Non-Terminating.

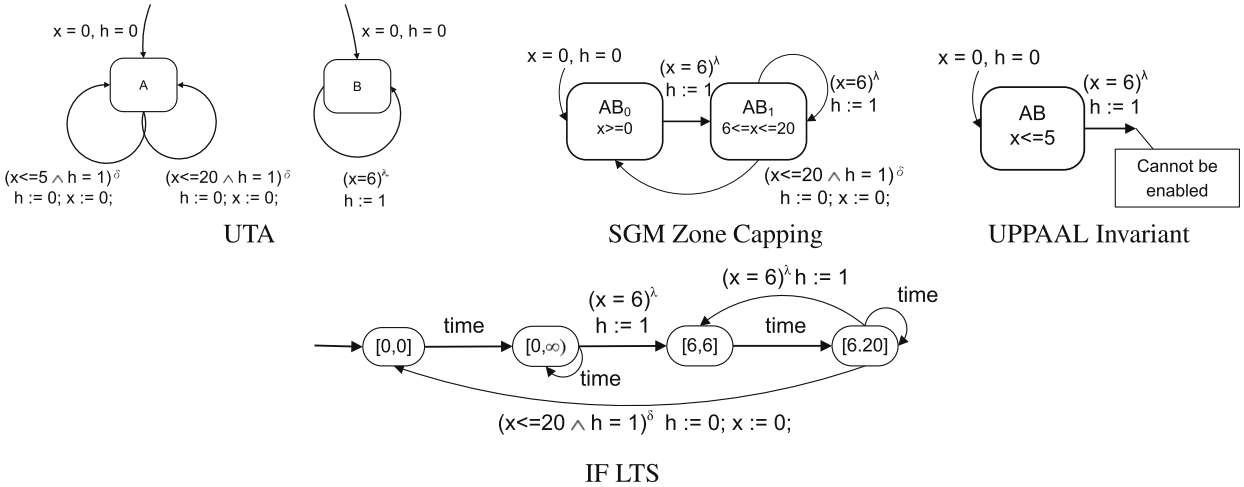


Fig. 10. Comparing between zone capping, state invariant, and IF LTS.

SGM are all smaller in size than the labeled transition systems generated by IF. For the larger examples such as the priority arbiter and the lip synchronization algorithm, the exhaustive simulation in IF does not terminate, while SGM can generate manageable state graphs that can be model checked. Table 4 shows the running time of all the experiments.

To show how the LTS generated by IF differs from our zone-based urgency handling in SGM, we use a small example, as illustrated in Fig. 10, where the delayable transition with trigger $x \leqslant 20$ when simulated in IF, using `-tf -dfs -po` parameters, results in an LTS with five states and eight transitions (the initial state and initial transition are also taken into account by IF). In comparison, the urgency semantics in SGM produces a state-graph with only two states and three transitions. Moreover, using state invariants in UPPAAL results in only a single initial state due to timelock as shown in Fig. 10.

From the above experiments, we can observe that our zone-based urgency semantics as implemented in SGM has the following advantages. First, compared to the state-of-the-art CTL model checkers, modeling and verifying systems with urgency semantics has become feasible, straightforward, flexible, and consistent with model checking. Second, compared to the IF toolset, the urgency handling is more symbolic and the sizes of the state graphs are thus much reduced, which makes model checking more efficient. Third, timelocks are naturally avoided due to the UTA semantics and urgency handling, thus we need not use invariants for enforcing urgency now. A limitation of the proposed zone-based urgency semantics in SGM is that only one type of composition (complete urgency) is proposed and implemented for urgent transitions.

6. Conclusions

We have proposed the verification of urgent timed systems, modeled by urgent timed automata (UTA), using a zone-based urgency semantics for CTL model checking. We have proposed a novel zone capping operation, which enforces the semantics of urgency types in UTA and produces time-reactive state graphs that satisfy all urgency deadlines. A necessary and sufficient condition, called *complete urgency*, is also proved for avoiding zone partitioning. It is shown that complete urgency is preserved by the newly proposed zone capping method. Several application examples illustrate how our method for verifying real-time embedded systems with urgency is more symbolic and efficient compared to the state-of-the-art model checkers such as IF. In the future, we will consider different types of urgency compositions.

References

Alur, R., Dill, D.L., 1994. A theory of timed automata. Theoretical Computer Science 126 (2), 183–235.

Alur, R., Courcoubetis, C., Dill, D.L., 1990. Model-checking for real-time systems. In: Proceedings of the 5th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, pp. 414–425.

Barbuti, R., Tesei, L., 2004. Timed automata with urgent transitions. Acta Informatica 40 (5), 317–347.

Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Wang, Y., 1995. UPPAAL: a tool suite for automatic verification of real-time systems. In: Proceedings of Workshop on Verification and Control of Hybrid Systems III, LNCS, vol. 1066, pp. 232–243.

Bornot, S., Sifakis, J., 2000. An algebraic framework for urgency. Information and Computation 163 (1), 172–202.

Bornot, S., Sifakis, J., Tripakis, S., 1997. Modeling urgency in timed systems. In: Proceedings of the International Symposium on Compositionality: The Significant Difference, LNCS, vol. 1536, Springer-Verlag, pp. 103–129.

Bowman, H., Faconti, G., Katoen, J.-P., Latella, D., Massink, M., 1998. Automatic verification of a lip synchronisation algorithm using UPPAAL. In: Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems, pp. 97–124.

Bozga, M., Fernandez, J.Cl., Ghirvu, L., Graf, S., Krimm, J.P., Mounier, L., 1999. IF: an intermediate representation and validation environment for time asynchronous systems. In: Proceedings of the Formal Methods Conference (FM), September 1999, pp. 307–327.

Bozga, M., Graf, S., Mounier, L., Ober, I., Roux, J.-L., Vincent, D., 2001. Timed extensions for SDL. In: Proceedings of the 10th SDL Forum, LNCS, vol. 2078, Springer-Verlag, pp. 223–240.

Cassez, F., Pagetti, C., Roux, O., 2002. A timed extension for AltaRica. Research Report R 12002-13, IRCCyN/CNRS, Nantes, France. <http://www.irccyn.ec-nantes.fr>.

Clarke E.M., Emerson. E.A., 1981. Design and sythesis of synchronization skeletons using branching time temporal logic. In: Proceedings of the Logics of Programs Workshop, LNCS, vol. 131, Springer-Verlag, pp. 52–71.

Dill, D.L., 1989. Timing assumptions and verification of finite-state concurrent systems. In: Proceedings of Workshop on Automatic Verification Methods for Finite State Systems, LNCS, vol. 407, Springer-Verlag, pp. 197–212.

Gebremichael, B., Vaandrager, F., 2005. Specifying urgency in timed I/O automata. In: Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM), pp. 5–9.

Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S., 1992. Symbolic model checking for real-time systems. In: Proceedings of the IEEE International Conference on Logics in Computer Science (LICS), pp. 394–406.

Hogrefe, D., Koch, B., Neukirchen, H., Some implications of MSC, SDL, and TTCN timed extensions for computer aided test generation. In: Proceedings of the 10th SDL Forum, LNCS, vol. 2078, Springer-Verlag, pp. 168–181.

Hsiung, P.-A., Wang. F., 1998. 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).

INTERVAL Consortium, 1999–2001. Interval project: formal design, validation and testing of real-time telecommunication systems, (definition of the timed extensions to SDL, MSC, TTCN). IST-1999-11557. <http://www-interval.imag.fr>.

Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F., 2003. Timed I/O automata: a mathematical framework for modeling and analyzing real-time systems. In: Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS), IEEE CS Press, pp. 166–177.

Kwiatkowska, M., Norman, G., Segala, R., Sproston, J., 2000. Verifying soft deadlines with probablistic timed automata. In: Proceedings of the Workshop on Advances in Verification (WAVe).

Lin, S.-W., Hsiung, P.-A., Huang, C.-H., Chen, Y.-R., 2005. Model checking prioritized timed automata. In: Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA), LNCS, vol. 3707, Springer-Verlag, pp. 370–384.

Sifakis, J., Yovine, S., 1996. Compositional specification of timed systems. In: Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS), LNCS, vol. 1046, Springer-Verlag, pp. 347–359.

Stefani, J.-B., Hazard, L., Horn, F., 1992. Computational model for distributed multimedia application based on a synchronous programming language. Computer Communications (Special Issue on FDTs) 15 (2), 114–128.

Wang, F., 2001. 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, Department of Information Technology, Uppsala University, ISSN:1404-3203.

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

Yovine, S., 1997. Kronos: a verification tool for real-time systems. International Journal of Software Tools for Technology Transfer 1 (1/2), 123–133.

Pao-Ann Hsiung Ph.D., received his B.S. in Mathematics and his Ph.D. in Electrical Engineering from the National Taiwan University, Taipei, Taiwan, ROC, in 1991 and 1996, respectively. From 1996 to 2000, he was a post-doctoral researcher at the Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC. From February 2001 to July 2002, he was an assistant professor and from August 2002 to July 2007 he was an associate professor in the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, ROC. Since August 2007, he has been a full professor. Dr. Hsiung was the recipient of the 2001 ACM Taipei Chapter Kuo-Ting Li Young Researcher for his significant contributions to design automation of electronic systems. Dr. Hsiung was also a recipient of the 2004 Young Scholar Research Award given by National Chung Cheng University to five young faculty members per year. Dr. Hsiung is a senior member of the IEEE, a senior member of the ACM, and a life member of the IICM. He has been included in several professional listings such as Marquis' Who's Who in the World, Marquis' Who's Who in Asia, Outstanding People of the 20th Century by International Biographical Centre, Cambridge, England, Rifacimento International's Admirable Asian Achievers (2006), Afro/Asian Who's Who, and Asia/Pacific Who's Who. Dr. Hsiung is an editorial board member of the International Journal of Embedded Systems (IJES), Inderscience Publishers, USA; the International Journal of Multimedia and Ubiquitous Engineering (IJMUE), Science and Engineering Research Center (SERSC), USA; an associate editor of the Journal of Software Engineering (JSE), Academic Journals, Inc., USA; an editorial board member of the Open Software Engineering Journal (OSE), Bentham Science Publishers, Ltd., USA; an international editorial board member of the International Journal of Patterns (IJOP). Dr. Hsiung has been on the program committee of more than 50 international conferences. He served as session organizer and chair for PDPTA'99, and as workshop organizer and chair for RTC'99, DSVV'2000, and PDES'2005. He has published more than 160 papers in international journals and conferences. He has taken an active part in paper refereeing for international journals and conferences. His main research

interests include reconfigurable computing and system design, multi-core programming, cognitive radio architecture, System-on-Chip (SoC) design and verification, embedded software synthesis and verification, real-time system design and verification, hardware-software codesign and coverification, and component-based object-oriented application frameworks for real-time embedded systems.

Shang-Wei Lin received his B.S. in management information system from National Chung Cheng University, Chiayi, Taiwan, ROC, in 2002. He is currently working towards his Ph.D. in the Department of Computer Science and Information Engineering at National Chung Cheng University, Chiayi, Taiwan, ROC. He is a teaching and research assistant in the Department of Computer Science and Information Engineering at National Chung Cheng University. His research interests include formal verification, formal synthesis, scheduling, embedded system design, and object-oriented software synthesis.

Yean-Ru Chen received her B.S. in Computer Science and Information Engineering from the National Chiao Tung University, Hsinchu, Taiwan, ROC in 2002. From 2002 to 2003, she was employed as an engineer in SoC Technology Center, Industrial Technology Research Institute, Hsinchu, Taiwan, ROC. She received her M.S. in Computer Science and Information Engineering from the National Chung Cheng University, ChiaYi, Taiwan, ROC in 2006. She is currently a Ph.D. candidate in Graduate Institute of Electronics Engineering of National Taiwan University, Taipei, Taiwan, ROC. Her current research interests include model checking, safety-critical systems and Electronic System Level (ESL) Design.

Chun-Hsian Huang received his B.S. degree in Information and Computer Education from National TaiTung University, TaiTung, Taiwan, ROC, in 2004. He is currently working toward his Ph.D. in the Department of Computer Science and Information Engineering at National Chung Cheng University, Chiayi, Taiwan, ROC. He is a teaching and research assistant in the Department of Computer Science and Information Engineering at National Chung Cheng University. His research interests include dynamically partially reconfigurable systems, UML-based hardware/software co-design methodology, hardware/software co-verification, and formal verification.

Chihhsiong Shih received his BSc degree from Chung Yuan Christian University, Taiwan, in 1984 and his MSc in computer science, and PhD degree in mechanical engineering in 1997, all from Rensselaer Polytechnic Institute. Since then, he has been working in the CAD software industry. From 1997–2000, he has worked for a CAD simulation company, Simmetrix, while from 2000–2002, he worked for the EDA team of microelectronic division of IBM Corp. He has broad interests in the software-assisted CAD applications, including engineering simulation and electrical properties analysis. He is currently involved with graphics and vision assisted 3D applications, e.g., reverse engineering and embedded software engineering research in Tunghai University, Taiwan, as an assistant professor.

William Cheng-Chung Chu is the dean of Engineering College, a professor of the Department of Computer Science, and the Director of Software Engineering and Technologies Center of Tunghai University. He had served as the Dean of Research and Development office at Tunghai University from 2004 to 2007, Taiwan. From 1994 to 1998, he was an associate professor at the Department of Information Engineering and Computer Science at Feng Chia University. He was a research scientist at the Software Technology Center of the Lockheed Missiles and Space Company, Inc., where he received special contribution awards in both 1992 and 1993 and a PIP award in 1993. In 1992, he was also a visiting scholar at Stanford University. He is serving as the associate editor for Journal of Software Maintenance and Evolution (JSME) and Journal of Systems and Software (JSS). His current research interests include software engineering, embedded systems, and E-learning. Dr Chu received his MS and PhD degrees from Northwestern University in Evanston Illinois, in 1987 and 1989, respectively, both in computer science. He has edited several books and published over 100 referred papers and book chapters, as well as participating in many international activities, including organizing international conferences.