International Journal of Foundations of Computer Science

MODELING AND ANALYSIS OF REAL -TIME SYSTEMS WITH MUTEX COMPONENTS

Guoqiang Li Xiaojuan Cai
BASICS, School of Software,
MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems
Shanghai Jiao Tong University
Shanghai, 200240, China
{li.g, cxj}@sjtu.edu.cn

Shoji Yuen
Graduate School of Information Science, Nagoya University
Nagoya, 464-8601, Japan
yuen@is.nagoya-u.ac.jp

Communicated by (xxxxxxxxxx)

Timed automata are commonly recognized as a formal behavioral model for real-time systems. For compositional system design, parallel composition of timed automata as proposed by Larsen *et al.* [22] is useful. Although parallel composition provides a general method for system construction, in the low level behavior, components often behave sequentially by passing control via communication. This paper proposes a behavioral model, named *controller automata*, to combine timed automata by focusing on the control passing between components. In a controller automaton, to each state a timed automaton is assigned. A timed automaton at a state may be preempted by the control passing to another state by a global labeled transition. A controller automaton properly extends the expressive power because of the stack, but this can make the reachability problem undecidable. Given a strict partial order over states, we show that this problem can be avoided and a controller automaton can be faithfully translated into a timed automaton.

Keywords: Controller Automata; Timed Automata; Real-Time Systems; Mutex.

1991 Mathematics Subject Classification: 22E46, 53C35, 57S20

1. Introduction

Real-time systems, due to the requirements to complete their work and deliver their services on a timely basis, easily fall into pitfalls if improperly designed. In order to guarantee their correctness, lots of formal models [26], such as *timed automata* [3, 19], have been proposed and widely used.

A real-time system usually consists of functionally independent components. The control mechanism of these components is implemented by *synchronization* and *mutexes*. In the design of real-time systems, it is often the case that the behavior is considered at two levels: in the lower level, the components behaves sequentially by passing controls between

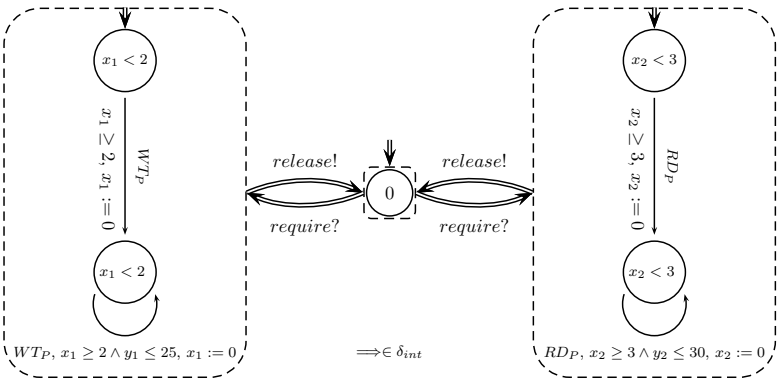
components, such as task switching and interrupt handling. In the upper level, the components behave concurrently through synchronizations and communications implemented in the lower level. In order to reason about timed behavior of such a composite timed system, it is desirable to have a simple formal treatment for the lower-level sequential behavior in addition to the upper-level concurrent behavior.

As for synchronization, we refer to the pioneering work [22], where a *parallel composition* of timed automata is proposed by Larsen *et al.* to analyze real-time systems with synchronized components. It uses the communication mechanism in CCS [28]. External symbols of timed automata are categorized into two disjoint sets, *triggered symbols* and *triggering symbols*, denoted by $a?, b?, c?, \ldots$, and $a!, b!, c!, \ldots$, respectively. Two automata with corresponding symbols synchronize via a channel and run simultaneously.

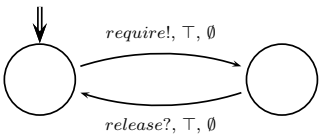
Mutexes provide a form of sequential execution in a composite timed system. Traditionally, mutexes can be encoded by synchronization [28]. In real-time systems, however, even if the execution time is controlled by a mutex of the processor, clocks proceed at the same rate regardless of the execution status. The priority control among components is one common way to choose which component is to be executed. We represent the require/release operations of control directly by meta-level transitions. Although the synchronization gives a basic operation for a mutex, the direct treatment of mutexes has an advantage for reasoning about timed behavior of the whole system, in that it leads to a simpler representation of time passage.

This paper proposes *controller automata*, generalized from the model in our previous work [24]. In a controller automaton, a timed automaton is assigned to each state. The timed automaton at the current state is enabled to make a transition; A labeled relation is adopted to show control passing among the timed automata, including *push* (for preemption relations), *pop* (for resumption relations), and *internal* (for competition relations). Essentially, a controller automaton controls a set of timed automata sequentially with possible preemption. A parallel composition between a controller automaton and a timed automaton is also defined for synchronizing the system with other systems.

Let's illustrate the model by two simple examples. Firstly, assume there are two processes that compete with a shared buffer P. One intends to write data to P continuously during 25 time units, and needs 2 time units to prepare each datum before writes through the action WT_P; The other intends to read the datum from P within 30 time units, and it needs 3 time units to read through the action RD_P. A controller automaton for the buffer allocator is shown in (a) of Figure 1. Two timed automata describe two processes. Actions WT_P and RD_P are represented as input symbols of the automata. An empty timed automaton (in the middle) is used to describe the situation where no processes are invoked. One of them is nondeterministically chosen when its $require?$ action is triggered by corresponding symbol $require!$. The unchosen one has to wait for $require!$ being available again. After finishing the writing or reading, the processes generate the symbol $release!$, and return back to the empty automaton. Furthermore, another timed automaton is needed to represent a semaphore of the buffer, shown in (b) of Figure 1, generating $require!$ and $release?$ alternately. Hence the system is represented as the parallel composition between the controller automaton in Figure 1(a) and the timed automaton in Figure 1(b).



(a) A Buffer Allocator



(b) Semaphore

Fig. 1. A Shared Buffer Example

When priority between processes is considered, a stack is adopted to store the current running status, and *a time lag function* to transform a timed automaton to wait a certain time when preempted by another timed automata. We show in this paper that the reachability problem on controller automata is in general undecidable. Thus the expressiveness of controller automata goes beyond that of timed automata. With a *strict partial order* over the state set of a controller automaton, an algorithm is proposed to translate an ordered controller automaton to a timed automaton. Hence the reachability problem of ordered controller automata is reducible to the reachability problem of timed automata, which is already implemented by several tools, e.g., UPPAAL [23]. By this approach, we are able to analyze various properties, such as schedulability analysis on complex real-time systems automatically.

Paper Organizations

The rest of the paper is organized as follows. Section 2 briefly introduces timed automata, as a preliminary of our research. Section 3 gives the formal definition and semantics of controller automata. Section 4 presents decidability discussion of controller automata and an algorithm to translate an ordered controller automaton to a timed automaton. 5 illustrates the usages of the controller automata by several examples. Section 6 gives the related work and section 7 concludes the paper.

2. Timed Automata

This section briefly reviews *timed automata* [3, 19].

Definition 1 (Time Constraints) *Let $\mathbb{R}^+$ be the set of non-negative real numbers and $X = \{x_1, \ldots, x_n\}$ be a finite set of* clocks*. The set of* clock constraints*, $\Phi(X)$, over X is defined by the grammar:*

$$\phi ::= \top \mid x \bowtie c \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$$

where $c \in \mathbb{R}^+$, $x \in X$, and $\bowtie \in \{<, >, \leq, \geq\}$.

For the set of clocks X, a *clock valuation* is a function $\nu : X \to \mathbb{R}^+$. It assigns a value to each clock $x \in X$. For a clock valuation ν and a clock constraint ϕ, we write $\nu \models \phi$ to denote that ν satisfies the constraint ϕ. Given a set of clocks $\lambda \subseteq X$ and a clock valuation ν, let a *clock reset function* $\nu[\lambda]$ be a clock valuation, defined as follows:

$$(\nu[\lambda])(x) = \begin{cases} 0 & \text{if } x \in \lambda \\ \nu(x) & \text{otherwise} \end{cases}$$

Given a clock valuation ν and a time $t \in \mathbb{R}^+$, we define $(\nu + t)(x) = \nu(x) + t$, for $x \in X$.

Timed automata were first proposed in [3]. The original timed automata use Büchi accepting conditions to enforce progress properties [3]. Later *timed safety automata*, introduced in [19], specify progress properties using local invariants. In this paper, we shall focus on timed safety automata, referring them as timed automata, when it is understood from the context. In order to define the parallel compositions, we distinguish actions of timed automata by internal actions and external actions [20].

Definition 2 (Timed Automata) *A timed (safety) automaton is a tuple $\mathscr{A} = (E, H, Q, q_0, X, I, \delta)$, where*

- *E is a finite set of external symbols, composed of two disjoint sets $E = E_o \cup E_i$, where E_o is the set of* triggering symbols*, and E_i is the set of* triggered symbols*.*
- *H is a finite set of internal symbols.*
- *Q is a finite set of control locations.*
- *$q_0 \in Q$ is the initial location.*
- *X is a finite set of clocks.*
- *$I : Q \to \Phi(X)$ is a function assigning each location with a clock constraint, called an* invariant*.*
- *$\delta \subseteq Q \times (E \cup H) \times \Phi(X) \times 2^X \times Q$.*

When $\langle q_1, a, \phi, \lambda, q_2\rangle \in \delta$, we write $q_1 \xrightarrow{a,\phi,\lambda} q_2$. If we let $\Sigma = E \cup H$, then the definition above is exactly the same as the definition in [19].

Given a timed automaton $\mathcal{A} \in \mathscr{A}$, we use $E(\mathcal{A})$, $H(\mathcal{A})$, $Q(\mathcal{A})$, $q_0(\mathcal{A})$ and $X(\mathcal{A})$ to represent its set of external symbols, internal symbols, control locations, initial location, and set of clocks, respectively. We will use similar notations for controller automata.

The semantics of timed automata includes progress transitions, for time elapsing within one control location, and discrete transitions, for transference between two control locations [3].

Definition 3 (Semantics of Timed Automata) *A* configuration of TA *is a pair* (q, ν) *of a control location* $q \in Q$*, and a clock valuation* ν *on* X*. The labeled transition system (LTS) of timed automata is represented as follows,*

- Progress transition*:* $(q, \nu) \xrightarrow{t}_{\mathscr{A}} (q, \nu + t)$*, where* $t \in \mathbb{R}^+$ *and* $(\nu + t) \models I(q)$.
- Discrete transition*:* $(q_1, \nu) \xrightarrow{a}_{\mathscr{A}} (q_2, \nu[\lambda])$*, if* $q_1 \xrightarrow{a,\phi,\lambda} q_2$*, and* $\nu \models \phi$*, and* $\nu[\lambda] \models I(q_2)$.

Here we use LTS to define semantics of timed automata, in order to provide facilities for later theorems. This semantics is consistent with the original one [19], if we write $(q, \nu) \rightarrow_{\mathscr{A}} (q', \nu')$ instead of $(q, \nu) \xrightarrow{t}_{\mathscr{A}} (q', \nu')$ or $(q, \nu) \xrightarrow{a}_{\mathscr{A}} (q', \nu')$.

Although general verification problems, such as the language inclusion problem, are undecidable on timed automata, the reachability problem for real-time systems [3, 19] is decidable.

Fact 1. *The reachability problem of timed automata is decidable* [3, 19].

Timed automata are often composed by a *parallel composition* [22] over a common set of clocks and external actions, borrowing the synchronization idea from CCS [28]. A parallel composition of a set of timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is denoted as $\mathscr{P} = \mathcal{A}_1 \| \ldots \| \mathcal{A}_n$.

Assuming $\mathcal{A}_i = (E, H_i, Q_i, q_i^0, X, I_i, \delta_i)$ for $i \in \{1, \ldots, n\}$, a *location vector* is a vector $\overline{q} = (q_1, \ldots, q_n)$. We compose the invariant functions into a common function over location vectors $I(\overline{q}) = \bigwedge_{i=1}^{n} I_i(q_i)$. Let $\overline{q}[q_i'/q_i]$ to denote the location vector where the i-th element q_i is replaced by q_i'. The semantics of a parallel composition of timed automata is defined as follows:

Definition 4 (Semantics of a Parallel Composition of TA) *A* configuration of parallel composition of TA *is a pair* $(\overline{q}, \nu)$*, where* $\overline{q}$ *is a location vector,* $\overline{q} \in Q_1 \times \ldots \times Q_n$*, and* ν *is a clock valuation on* X.

- *Progress transition:*
  - $(\overline{q}, \nu) \xrightarrow{t}_{\mathscr{P}} (\overline{q}, \nu + t)$*, where* $t \in \mathbb{R}^+$ *and* $(\nu + t) \models I(\overline{q})$.
- *Discrete transition:*
  - $(\overline{q}, \nu) \xrightarrow{a}_{\mathscr{P}} (\overline{q}[q_i'/q_i], \nu[\lambda])$*, if* $q_i \xrightarrow{a,\phi,\lambda} q_i'$*,* $\nu \models \phi$*,* $a \in H_i$*, and* $\nu[\lambda] \models I(\overline{q}[q_i'/q_i])$.
  - $(\overline{q}, \nu) \xrightarrow{\tau}_{\mathscr{P}} (\overline{q}[q_i'/q_i, q_j'/q_j], \nu[\lambda_1 \cup \lambda_2])$*, if* $q_i \xrightarrow{a?,\phi_1,\lambda_1} q_i'$*,* $q_j \xrightarrow{a!,\phi_2,\lambda_2} q_j'$*,* $a! \in E_o$ *,*$a? \in E_i$*,* $\nu \models \phi_1 \wedge \phi_2$*, and* $\nu[\lambda_1 \cup \lambda_2] \models I(\overline{q}[q_i'/q_i, q_j'/q_j])$.

The parallel composition of timed automata does not enrich the expressiveness of timed automata. There exists an algorithm to translate a parallel composition of timed automata to a timed automaton [22]. Thus,

Fact 2. *A parallel composition of two timed automata is a timed automaton.*

3. Controller Automata

This section presents *controller automata* to deal with real-time systems with mutex components. We equip timed automata with three relations, *internal*, *push*, and *pop*, to handle competition, preemption and resumption,respectively.

3.1. *Formal Definition*

To avoid conflicts with the terminology in timed automata, we name control locations of controller automata *states*.

Definition 5 (Controller Automata) *A controller automaton (CA) is a tuple* $\mathscr{C} = (E, H, S, s_0, M, x_{run}, T, \delta)$*, where*

- E *is a finite set of external symbols, and* H *is a finite set of internal symbols.*
- S *is a finite set of states, and* $s_0 \in S$ *is the initial state.*
- $M : S \to \mathscr{A}$ *is a function assigning each state with a timed automaton, where the assigned timed automata share external symbols with* E*, and their internal symbols (including* H*) are pairwise disjoint.*
- $T : S \to \mathbb{R}^+$ *is a function assigning each state with a time, named* expected running time*.*
- x_{run} *is a global clock to accumulate the running time of all states.*
- $\delta \subseteq S \times (E \cup H) \times S$*, which is partitioned into three disjoint subsets,* δ_{push}*,* δ_{pop}*,* δ_{int}*, for push, pop and internal relations, respectively.*

3.2. *Time Lag in Timed Automata*

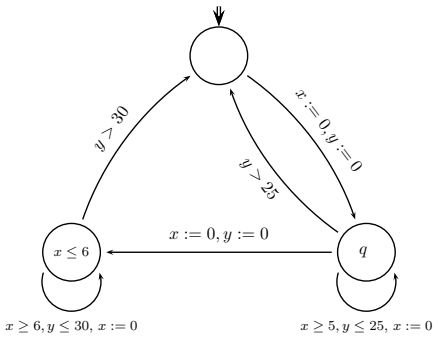
To formally define the operational semantics of controller automata, an important technique is to suspend and resume timed automata.

When a timed automaton is preempted by another one, the system will stop running the current timed automaton, store the current status, and begin to run the other timed automaton. A *time lag* transforms a timed automaton to wait a certain time when preempted by another timed automata.

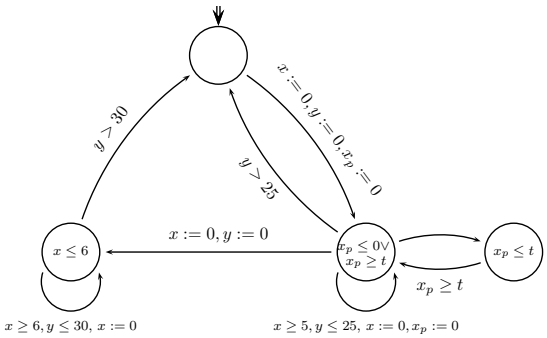
A time lag occurs at a given control location q in a timed automaton $\mathcal{A}$. We need an extra idle location q^{id} for q. The definition of $\texttt{TimeLag} : \mathscr{A} \times Q(\mathscr{A}) \times \mathbb{R}^+ \to \mathscr{A}$ accepts a timed automaton, a control location on this timed automaton, a time interval, and returns a timed automaton, with the following definition, $\texttt{TimeLag}(\mathcal{A}, q, t) = (E, H \cup \{r, l\}, Q(\mathcal{A}) \cup \{q^{id}\}, q_0, X \cup \{x_p\}, I', \delta')$, where

- $I'(q) = I(q) \wedge \{x_p \leq 0 \vee x_p \geq t\}$, $I'(q^{id}) = \{x_p \leq t\}$, and $I'(q') = I(q')$ for all $q' \neq q$.
- Define $\delta'' = \{q' \xrightarrow{a,\phi,\lambda \cup \{x_p\}} q \mid q' \in Q, q' \xrightarrow{a,\phi,\lambda} q \in \delta\} \cup \{q' \xrightarrow{a,\phi,\lambda} q'' \mid q', q'' \in Q, q' \xrightarrow{a,\phi,\lambda} q'' \in \delta \wedge q'' \neq q\}$, and $\delta' = \delta'' \cup \{q \xrightarrow{r,\top,\emptyset} q^{id}, q^{id} \xrightarrow{l,x_p \geq t,\emptyset} q\}$.

Let's take an example to illustrate the function. Assume a time lag t occurs in the control location q of the timed automaton in (a) of Figure 2. After performing the above translation, a fresh clock x_p, and a fresh control location are inserted into the timed automaton. x_p is reset to 0 on each edge of the control location q, and during the time t, the system can only stay within the fresh control location. The obtained automaton is shown in (b) of Figure 2.



(a) A Timed Automaton



(b) Time Lag t on the control location q

Fig. 2. Time Lag in Timed Automata

It may happen that the inserted fresh location affects the whole system negatively. That is, when the location q transits to the idle location q^{id}, it may not return to q after t time units, due to the violation of $I'(q)$. It also cannot stay in q^{id}, since $I'(q^{id})$ is also violated after t time. Thus an empty timed automaton is produced, which exactly explains that a system may abort its normal execution after resumption when some unexpected break happens.

An idle location can be regarded as a context of the preempting timed automaton, which will be replaced by the preempting timed automaton later.

3.3. *Semantics*

We prepare a stack for configurations of a controller automaton, in order to record the state and the running control location when a push action is performed, and to resume the running context when a pop action is performed. By introducing the time lag, the semantics of a controller automata can be formally defined as follows:

Definition 6 (Semantics of Controller Automata) *A configuration for a controller automaton $\mathcal{C}$ is a tuple $(s, q, \nu, \kappa, \mathtt{S})$,*

- *s is the running state of $\mathcal{C}$;*
- *q is the current running control location in $M(s)$;*
- *ν is the clock valuation for all clocks $\bigcup_{s\in S} X(M(s)) \cup \{x_{run}\}$ in $\mathcal{C}$;*
- *κ is the set of clocks keeping frozen when the time elapses, named* frozen clocks*;*
- *$\mathtt{S}$ is the stack.*

- *Progress transitions:*
  - *$(s, q, \nu, \kappa, \mathtt{S}) \xrightarrow{t}_{\mathscr{C}} (s, q, (\nu + t)[\kappa], \kappa, \mathtt{S})$, if $(\nu + t)[\kappa] \models x_{run} \leq T(s)$, and $(q, \mu) \xrightarrow{t}_{\mathscr{A}} (q, \mu + t)$, where μ $(\subseteq \nu)$ is a clock valuation on $X(M(s))$.*
- *Discrete transitions:*
  - *Intra-action: $(s, q, \nu, \kappa, \mathtt{S}) \xrightarrow{a}_{\mathscr{C}} (s, q', \nu[\lambda], \kappa, \mathtt{S})$, if $(q, \mu) \xrightarrow{a}_{\mathscr{A}} (q', \mu[\lambda])$ in $M(s)$, where μ $(\subseteq \nu)$ is a clock valuation on $X(M(s))$.*
  - *Push: $(s, q, \nu, \kappa, \mathtt{S}) \xrightarrow{a}_{\mathscr{C}} (s', q_0(M(s')), \nu[x_{run}], \kappa \backslash X(M(s')), (s, q) :: \mathtt{S})$, and $M(t) := \mathtt{TimeLag}(M(t), q, \nu(x_{run}))$ for all (t, q) in $\mathtt{S}$, if $s \xrightarrow{a} s' \in \delta_{push}$.*
  - *Pop: $(s, q, \nu, \kappa, (s', q') :: \mathtt{S}) \xrightarrow{a}_{\mathscr{C}} (s', q', \nu[x_{run}], \kappa \cup X(M(s)), \mathtt{S})$, and $M(t) := \mathtt{TimeLag}(M(t), q, T(s))$ for all (t, q) in $(s', q') :: \mathtt{S}$, if $s \xrightarrow{a} s' \in \delta_{pop}$ and $\nu(x_{run}) = T(s)$.*
  - *Inter-action: $(s, q, \nu, \kappa, \mathtt{S}) \xrightarrow{a}_{\mathscr{C}} (s', q_0(M(s')), \nu[x_{run}], \kappa \backslash X(M(s')) \cup X(M(s)), \mathtt{S})$, and $M(t) := \mathtt{TimeLag}(M(t), q, T(s))$ for all (t, q) in the stack $\mathtt{S}$, if $s \xrightarrow{a} s' \in \delta_{int}$, $\nu(x_{run}) = T(s)$.*

Each transition has the intuitive meanings as follows,

- Progress transitions say a system can stay within a state for t time units if it can stay within a control location of the timed automaton of this state.
- If the timed automaton of some state has any transitions, then the transitions are reserved by the controller automaton. This is illustrated by Intra-action transitions.
- Push transitions reflect that the system pushes the current running state and the control location into the stack, and transits to the initial location of the time automaton in the latter state. Simultaneously, all timed automata in the states pushed in the stack have a time lag, with the time recorded by x_{run}.
- In contrast, pops of stacks are handled with pop transitions. If the latter state and the corresponding control location are on the top of the stack, then after the execution of the expected running time, the system pops the previous state and control

location from the stack, and begins to run the timed automaton from the popped control location. Simultaneously, all timed automata in the states of the stack have a time lag, with the time recorded by $T(s)$.

- Inter-action transitions show that after the execution of the expected running time, the system transits to the initial location of the time automaton in the latter state. All timed automata in the states of the stack have a time lag, with the time recorded by $T(s)$.

A controller automaton differs from a *pushdown automaton* in that the pushed/popped content of the stack is decided statically in a pushdown automaton, but dynamically during run time in a controller automaton.

3.4. *Synchronization Representation*

As for timed automata [22], synchronization of a controller automaton and a timed automata is represented as parallel composition of two automata.

Consider a timed automaton $\mathcal{A}$ and a controller automaton $\mathcal{C}$, where for each $s_i \in S(\mathcal{C})$, $M(s_i) = (E_i, H_i, Q_i, q_i^0, X_i, I_i, \delta_i)$. Assume that $\mathcal{A}$ shares common sets of external symbols with $\mathcal{C}$ and all $M(s_i)$. A parallel composition of $\mathcal{A}$ and $\mathcal{C}$ is denoted as $\mathcal{A}\|\mathcal{C}$.

A location pair is a pair $\tilde{q} = (q_a, q_i)$, where $q_a \in Q(\mathcal{A})$ and $q_i \in \bigcup Q(M(s_i))$. The invariant function over location pairs is composed by $I((q_a, q_i)) = I(q) \wedge I_i(q_i)$ where $q_i \in Q(M(s_i))$. Let $\tilde{q}[q_i'/q_i]$ denote the location pair where the i-th element q_i is replaced by q_i'.

Definition 7. *A configuration of parallel composition between a timed automaton and a controller automaton is a tuple $(s, \tilde{q}, \nu, \kappa, S)$, where s is a state of the controller $\tilde{q}$ is a location pair, $\tilde{q} \in Q(\mathcal{A}) \times \bigcup Q_i(M(s_i))$, ν is a clock valuation, κ is a set of frozen clocks, and S is a stack.*

- *Progress transition:*
  - $(s, \tilde{q}, \nu, \kappa, S) \xrightarrow{t}_{\mathscr{P}} (s, \tilde{q}, (\nu + t)[\kappa], \kappa, S)$, *where* $t \in \mathbb{R}^+$ *and* $(\nu + t)[\kappa] \models I(\tilde{q})$.
- *Discrete transition:*
  - $(s, \tilde{q}, \nu, \kappa, S) \xrightarrow{a}_{\mathscr{P}} (s, \tilde{q}[q_i'/q_i], \nu[\lambda], \kappa, S)$, *if* $q_i \xrightarrow{a,\phi,\lambda} q_i'$, $\nu \models \phi$ *where* $a \in H_i$ *and* $\nu[\lambda] \models I(\tilde{q}[q_i'/q_i])$.
  - $(s, \tilde{q}, \nu, S) \xrightarrow{\tau}_{\mathscr{P}} (s, \tilde{q}[q_i'/q_i, q_j'/q_j], \nu[\lambda_1 \cup \lambda_2], S)$, *if* $q_i \xrightarrow{a?,\phi_1,\lambda_1} q_i'$, $q_j \xrightarrow{a!,\phi_2,\lambda_2} q_j'$ *where* $a! \in E_o$, $a? \in E_i$, $\nu \models \phi_1 \wedge \phi_2$, *and* $\nu[\lambda_1 \cup \lambda_2] \models I(\tilde{q}[q_i'/q_i, q_j'/q_j])$.
  - $(s, (q_a, q'), \nu, \kappa, \mathtt{S}) \xrightarrow{a}_{\mathscr{P}} (s', (q_a, q_0(M(s'))), \nu[\lambda \cup \{x_{run}\}], \kappa \backslash X(M(s')), (s, q) :: \mathtt{S})$, *and* $M(s) := \mathtt{TimeLag}(M(s), q, \nu(x_{run}))$ *for all* (s, q) *in* $\mathtt{S}$, *if* $s \xrightarrow{a} s' \in \delta_{push}$ *and* $a \in H(\mathcal{C})$.
  - $(s, (q_a, q'), \nu, \kappa, \mathtt{S}) \xrightarrow{\tau}_{\mathscr{P}} (s', (q_a', q_0(M(s'))), \nu[\lambda \cup \{x_{run}\}], \kappa \backslash X(M(s')), (s, q) :: \mathtt{S})$, *and* $M(s) := \mathtt{TimeLag}(M(s), q, \nu(x_{run}))$ *for all* (s, q) *in* $\mathtt{S}$, *if*

$s \xrightarrow{a?} s' \in \delta_{push}$, $q_a \xrightarrow{a!,\phi,\lambda} q'_a$ *where* $a! \in E_o$, $a? \in E_i$, $\nu \models \phi$, *and* $\nu[\lambda \cup \{x_{run}\}] \models I(q'_a, q_0(M(s')))$.

- $(s, (q_a, q'), \nu, \kappa, \mathtt{S}) \xrightarrow{\tau}_{\mathscr{P}} (s', (q'_a, q_0(M(s'))), \nu[\lambda \cup \{x_{run}\}], \kappa \backslash X(M(s')), (s, q) :: \mathtt{S})$, *and* $M(s) := \mathtt{TimeLag}(M(s), q, \nu(x_{run}))$ *for all* (s, q) *in* $\mathtt{S}$, *if* $s \xrightarrow{a!} s' \in \delta_{push}$, $q_a \xrightarrow{a?,\phi,\lambda} q'_a$ *where* $a! \in E_o$, $a? \in E_i$, $\nu \models \phi$, *and* $\nu[\lambda \cup \{x_{run}\}] \models I(q'_a, q_0(M(s')))$.
- $(s, (q_a, q), \nu, \kappa, (s', q') :: \mathtt{S}) \rightarrow_{\mathscr{P}} (s', (q_a, q'), \nu[\lambda \cup \{x_{run}\}], \kappa \cup X(M(s)), \mathtt{S})$, *and* $M(s) := \mathtt{TimeLag}(M(s), q, T(s))$ *for all* (s, q) *in* $(s', q') :: \mathtt{S}$, *if* $s \xrightarrow{a} s' \in \delta_{pop}$ *and* $a \in H(\mathcal{C})$, $x_{run} = T(s)$.
- $(s, (q_a, q), \nu, \kappa, \mathtt{S}) \rightarrow_{\mathscr{C}} (s', (q_a, q_0(M(s'))), \nu[\lambda \cup \{x_{run}\}], \kappa \backslash X(M(s')) \cup X(M(s)), \mathtt{S})$, *and* $M(s) := \mathtt{TimeLag}(M(s), q, T(s))$ *for all* (s, q) *in the stack, if* $s \xrightarrow{a} s' \in \delta_{int}$ *and where* $a \in H(\mathcal{C})$, *and* $x_{run} = T(s)$.

A parallel composition between a timed automaton $\mathcal{A}$ and a controller automaton $\mathcal{C}$ can be translated to parallel compositions of $\mathcal{A}$ and the timed automaton assigned to each state of $\mathcal{C}$. Hence it does not enrich the expressiveness of controller automata.

Fact 3. *A parallel composition of a timed automaton and a controller automaton is a controller automaton.*

Definition 7 allows us to model compositional real-time systems, including both synchronized and mutex components.

4. A Decidable Subclass of Controller Automata

4.1. *Decidability Discussions*

Timed issues make decidability problems for automata more difficult [4]. For instance, the inclusion problem on timed automata is undecidable, and the reachability problem is decidable [3]. The reachability problem on *stopwatch automata* (also known as *integration graphs*), in which clocks can be frozen during transitions, and be resumed when given conditions are satisfied, is undecidable [21, 2].

Although there are frozen clocks in the controller automata, the expressiveness of time in controller automata does not go beyond timed automata. The occurrences of frozen clocks in controller automata seem similar to those in stopwatch automata. However, they are quite different. The reason that expressiveness of stopwatch automata goes beyond timed automata is that clocks in a stopwatch automaton can be stopped at any value. In comparison, all frozen clocks are kept zero in controller automata. It is easy to design an algorithm to handle frozen clocks by explicitly resetting these clocks on transitions, when translating a controller automaton to a timed automaton.

Unfortunately, the reachability problem of controller automata is still undecidable, due to potentially infinite insertions of fresh control locations and fresh clocks into timed automata when time lags occur. Such dynamic insertions, although easy to understand and adopt, lead to difficulties in analysis and proof. It is easy to show that these insertions can be avoided by the introduction of *timed automata with additions*, in which clocks may be

updated by addition under certain conditions. The existing work shows that the reachability problem on timed automata with subtraction, named *suspension automata* [27], is undecidable. For a similar reason, the reachability problem on timed automata with addition is also undecidable. Here we give another proof and witness of the undecidability result. A formal definition of timed automata with additions is as follows,

An *adjusting* is a function which assigns a clock valuation to another valuation. We define a *one-clock adjusting* α_x over the clock x has the following form:

$$\alpha_x ::= x := c \mid x := y + d \mid \alpha \circ \alpha$$

where $c, d \in \mathbb{Q}^+$, and $y \in X$. $\circ$ is named a *concatenation* of adjusting.

$$\alpha_x(\nu(x)) = \begin{cases} c & \text{if } \alpha \text{ is } x := c \\ \nu(y) + d & \text{if } \alpha \text{ is } x := y + d \\ \alpha_2(\alpha_1(\nu(x)) & \text{if } \alpha \text{ is } \alpha_1 \circ \alpha_2 \end{cases}$$

An *addition adjusting* $\Theta(X)$ over the clock X is a concatenation with the following form $\circ(\alpha_x)_{x \in X}$. It is easy to show that $\Theta(X)(\nu(x)) = \alpha_x(\nu(x))$ for each $x \in X$.

Definition 8 (Timed Automata with Additions) *A* timed automaton with additions *is a tuple* $\mathscr{A} = (Q, \Sigma, q_0, X, I, \delta)$*, where*

- Q *is a finite set of control locations.*
- Σ *is a finite set of input symbols.*
- $q_0 \in Q$ *is the initial location.*
- X *is a finite set of clocks.*
- $I : Q \to \Phi(X)$ *is a function assigning each location with a clock constraint, called an* invariant.
- $\delta \subseteq Q \times \Sigma \times \Phi(X) \times \Theta(X) \times Q$.

This definition comes from the definition of *updateable timed automata (UTA)* [7, 8, 9], and the timed automata with additions are also a subclass of UTA. In [9], the authors prove decidability results of several kinds of subclass of UTA, which shows the reachability problem of an UTA is undecidable if its adjusting contains $x := y + c$. Hence, the reachability problem of timed automata with additions is undecidable.

Fact 4. *The reachability problem of timed automata with additions is undecidable.*

In [24], it is shown that the reachability of error states is practically solvable with the implementation of controller automata by Maude. The next subsection will introduce a subclass of controller automata, in which a strict partial order is imposed over the state set of a controller automaton. With this restriction, there are an unbounded but finite number of fresh control locations and clocks inserted to timed automata within a controller automaton, the length of the stack is also finite. Hence, an ordered controller automata can be translated to a timed automaton.

4.2. *Ordered Controller Automata*

Definition 9 (Ordered Controller Automata) *For a controller automaton $\mathcal{C} \in \mathscr{C}$, if there is a strict partial order (irreflexive, asymmetric and transitive) on the set of states, $(S(\mathcal{C}), \prec)$, that satisfies the followings:*

- $s_1 \prec s_2$, *if* $s_1 \xrightarrow{a} s_2 \in \delta_{push}$.
- $s_1 \succ s_2$, *if* $s_1 \xrightarrow{a} s_2 \in \delta_{pop}$.
- s_1 *and* s_2 *are incomparable, if* $s_1 \xrightarrow{a} s_2 \in \delta_{int}$.

then $\mathcal{C}$ is an ordered controller automaton.

Note that if $\prec$ is a *strict total order*, then $\delta_{int} = \emptyset$, since no two elements are incomparable in a strict total order.

Lemma 10. *Given an ordered controller automaton, there exists an upper bound on the length of its stack.*

Proof. Given a controller automaton $\mathcal{C} = (E, H, S, s_0, M, x_{run}, T, \delta)$, and a strict partial order on the set of states $(S, \prec)$, since S is finite, then there exists an upper bound set P for S, such that (1) $P \subseteq S$, for each $s \in S - P$, there exists a $p \in P$, $s \prec p$; (2) for each $p, p' \in P$, $p \not\prec p'$. From the initial state s_0, we construct a set of chains, $\mathcal{N}$, such that each chain $s_0 \ldots . s_n$ satisfies that $s_i \prec s_{i+1}$ for $i < n$, and $s_n \in P$. The bound of the stack is the length of the longest chain in $\mathcal{N}$. □

Actually, when applying controller automata to a real application. A controller automaton is usually adopted to model a structured system in which all components run sequentially with a given strategy. The usage of the controller automaton is to give a global view of the execution of these components. The given strategy is usually able to be modeled as a partial order, to avoid possible deadlocks and livelocks of the system.

Remark 11. *Ordered controller automata prohibit self-preempting loop transitions which is uncommon in real applications. Almost all real-time systems with mutex components can be modeled as ordered controller automata. That is, ordered controller automata are enough for our purpose of representations.*

4.3. *Translation to Timed Automata*

We present a translation from an ordered controller automaton to a timed automaton, satisfying soundness and completeness with respect to set of symbols and the time to release them.

Given an ordered controller automaton $\mathcal{C} = (E, H, S, s_0, M, x_{run}, T, \delta)$, where there is a strict partial order on the set of states, $(S, \prec)$, and for each $s_i \in S$, $M(s_i) = (E, H_i, Q_i, q_i^0, X_i, I_i, \delta_i)$, a timed automaton $\mathcal{A}^{\mathcal{C}} = (E_{\mathcal{A}}, H_{\mathcal{A}}, Q_{\mathcal{A}}, q_{\mathcal{A}}^0, X_{\mathcal{A}}, I_{\mathcal{A}}, \delta_{\mathcal{A}})$ is constructed by,

- $E_{\mathcal{A}} = E$.

- $H_{\mathcal{A}} = (\bigcup_{i=1}^{n} H_i) \cup H$.
- $Q_{\mathcal{A}} \subset (\bigcup_{i=1}^{n} Q_i) \times (\bigcup_{i=1}^{n} Q_i)^*$, where for $(q, \hat{q}) \in Q_{\mathcal{A}}$, $\hat{q}$ is an *ascending chain*. That is, for every two elements q_i, q_j in $\hat{q}$ such that $\hat{q} = q_1 \dots q_i \dots q_j \dots q_n$, if $q_i \in Q(M(s_i))$ and $q_j \in Q(M(s_j))$, $s_i \prec s_j$ holds.
- $q^0_{\mathcal{A}} = (q_0(M(s_0)), \varepsilon)$, where ε is the empty sequence.
- $X_{\mathcal{A}} = (\bigcup_{i=1}^{n} X_i) \cup \{x_{run}\}$.
- For each $(q_i, \hat{q}_i) \in Q_{\mathcal{A}}$, where $q_i \in Q(M(s_i))$, $I_{\mathcal{A}}((q_i, \hat{q}_i)) = I_i(q_i) \wedge (x_{run} \leq T(s_i))$.
- $\delta_{\mathcal{A}}$ is defined as follows,
  - $(q, \hat{q}) \xrightarrow{a,\phi,\lambda} (q', \hat{q})$ if $q \xrightarrow{a,\phi,\lambda} q' \in \delta_i$ for some $M(s_i)$.
  - $(q, \hat{q}) \xrightarrow{a,\top,X_j \cup \{x_{run}\}} (q', \hat{q}q)$, if $q \in Q(M(s_i))$, $q' = q_0(M(s_j))$, and $s_i \xrightarrow{a} s_j \in \delta_{push}$, where $X_j = X(M(s_j))$.
  - $(q, \hat{q}q') \xrightarrow{a,(x_{run} \geq T(s_i)),\{x_{run}\}} (q', \hat{q})$, if $q \in Q(M(s_i))$, $q' \in Q(M(s_j))$, and $s_i \xrightarrow{a} s_j \in \delta_{pop}$.
  - $(q, \hat{q}) \xrightarrow{a,(x_{run} \geq T(s_i)),X_j \cup \{x_{run}\}} (q', \hat{q})$, if $q \in Q(M(s_i))$, $q' = q_0(M(s_j))$, and $s_i \xrightarrow{a} s_j \in \delta_{int}$, where $X_j = X(M(s_j))$.

Now we justify the above translation from an ordered controller automaton to a timed automaton. It has become a consensus that such a justification should consist of two parts. Firstly the translation $\mathcal{A}^{\mathcal{C}}$ of an ordered controller automaton $\mathcal{C}$ should simulate the actions of $\mathcal{C}$, including the progress transitions and discrete transitions. Secondly $\mathcal{A}^{\mathcal{C}}$ should not introduce any additional actions other than those simulations. These two remarks lead to the following definition — *subbisimilarity*, which was first proposed in [17] and used to capture the operational soundness and completeness between variants of π-calculus.

Definition 12 (Subbisimilarity) *Let $\mathcal{R}$ be a relation from the set of configurations of controller automata to that of timed automata. It is a* subbisimilarity *if the following properties hold:*

(1) If $d\mathcal{R}^{-1}c \xrightarrow{\sigma} c'$ for some $\sigma \in E \cup H \cup \mathbb{R}^+$, then there exists some d' such that $d \xrightarrow{\sigma} d'\mathcal{R}^{-1}c'$;
(2) If $c\mathcal{R}d \xrightarrow{\sigma} d'$ for some $\sigma \in E \cup H \cup \mathbb{R}^+$, then there exists some c' such that $c \xrightarrow{\sigma} c'\mathcal{R}d'$;

A controller automaton is subbisimilar to a timed automaton iff there is a subbisimilarity containing the pair of their initial configurations.

Subbisimilarity is a pre-order between the model of controller automata and the model of timed automata. The first requirement implies the preservation of actions of a controller automaton, while the second ensures the reflection of these actions.

Theorem 13 (Operational Correspondence) *Any ordered controller automaton $\mathcal{C}$ is subbisimilar to its corresponding timed automaton $\mathcal{A}^{\mathcal{C}}$.*

Proof. The proof is direct by constructing a relation $\mathcal{S} = \{\langle(s, q, \nu, \kappa, \mathtt{S}), ((q, \hat{q}), \nu)\rangle\}$, where $(s, q, \nu, \kappa, \mathtt{S})$ is a configuration of some ordered controller automaton $\mathcal{C}$, and $((q, \hat{q}), \nu)$ is a configuration of its corresponding timed automaton $\mathcal{A}^{\mathcal{C}}$. Considering all possible transitions of $(s, q, \nu, \kappa, \mathtt{S})$ and the definition of $\mathcal{A}^{\mathcal{C}}$, it is easy to see that $(s, q, \nu, \kappa, \mathtt{S})$ and $((q, \hat{q}), \nu)$ have same possible transitions, and after the same transitions the pair of two resulted configurations also belongs to $\mathcal{S}$. Thus $\mathcal{S}$ is a subbisimilarity. Note that all time recorded by fresh inserted clock in time lag are current captured by x_{run}; time lags are captured by the fresh inserted transitions (which satisfy the definition of the time lag). □

Theorem 13 points out that there is a subbisimilarity between any ordered controller automaton C and its translation A^C. This means that ordered controller automata can be embedded into timed automata by using our translation. More precisely, subbisimilarity ensures the soundness and completeness of our translation:

(1) Soundness — for any ordered controller automaton $\mathcal{C}$ there is a timed automaton $\mathcal{A}^{\mathcal{C}}$ which can simulate all its actions.
(2) Completeness — $\mathcal{A}^{\mathcal{C}}$ can also be simulated faithfully by $\mathcal{C}$.
(3) Both simulations are step by step and alternative.

Therefore, all the model-checking problems of any ordered controller automaton $\mathcal{C}$ are equivalent to those of timed automaton $\mathcal{A}^{\mathcal{C}}$. This is an elegant property which means that we do not need extra model checking tools for ordered controller automata.

5. Examples

This section illustrates how to adopt controller automata to represent mutex real-time systems by several examples.

5.1. *A Preemptive Buffer Allocator*

Let's continue our buffer example in Section 1. Assume that the writing process has a higher priority than the reading process. When the reading process has the shared buffer P, and the writing process is coming, the timed automaton of the reading process will be suspended, until the writing process finishes. Furthermore, the reading process is guaranteed to receive the first datum within 20 time units. The ordered controller automaton for this preemptive buffer allocator is shown in Figure 3, in which we assign $T(I) = 25$ and $T(II) = 30$ for their respective expected running time. Two more relations are inserted into the previous automaton, representing preemption and resumption relations between two processes.

By composing with a timed automaton for semaphores (omitted here) and translating the ordered controller automaton to a timed automaton, we can show that the error control location ERR is reachable.

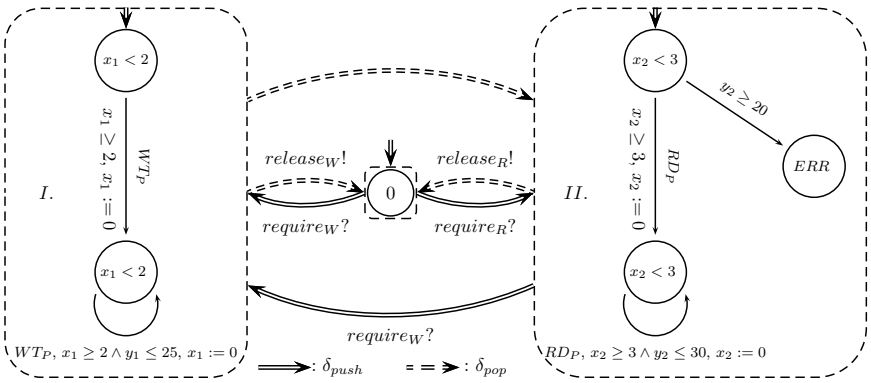


Fig. 3. A Preemptive Buffer Allocator

5.2. *System Representations for Nested Interrupts*

Nested interrupts of real-time systems easily fail when they are not properly designed. Especially in automotive systems, interrupt handling is critical.

A controller automaton can be used to represent nested interrupts, allocating different interrupt handlers. An interrupt handler is represented as a timed automaton in a state of the controller automaton, with external actions as awaited requirements of signals, or as events to trigger tasks.

A controller automaton to represent a system of a robot puppy is shown in Figure 4. The robot puppy has two functions, turning around and moving forward. If one pats the puppy's body, it will turn around; after 25 time units, if no one touches the puppy, it will stop. When the puppy is turning around, and is patted again, the puppy will move forward; after 30 time units, it will stop. At any time, when it is doubly patted, the puppy will stop. An interrupt handler is used to handle the interrupt signal from its skin sensor. Furthermore, another interrupt handler with a higher priority is implemented to handle the interrupt triggered by low battery power. The controller automaton has three states. The top right state is the initial state, representing the situation when no interrupts are invoked. The timed automaton in the left one represents the interrupt handler to handle interrupt signals from the skin sensor. The right-bottom state is for the interrupt handler triggered by low battery power.

Analysis techniques on real-time systems with nested interrupts, based on controller automata and timed automata are shown in our previous work [24]. In [24], implement a simulation tool for controller automata, which, due to the undecidability results, is unable to guarantee termination. However, nested interrupts can actually be modeled as ordered controller automata, as shown in this subsection. The schedulability analysis of real-time systems with nested interrupts is thus decidable.

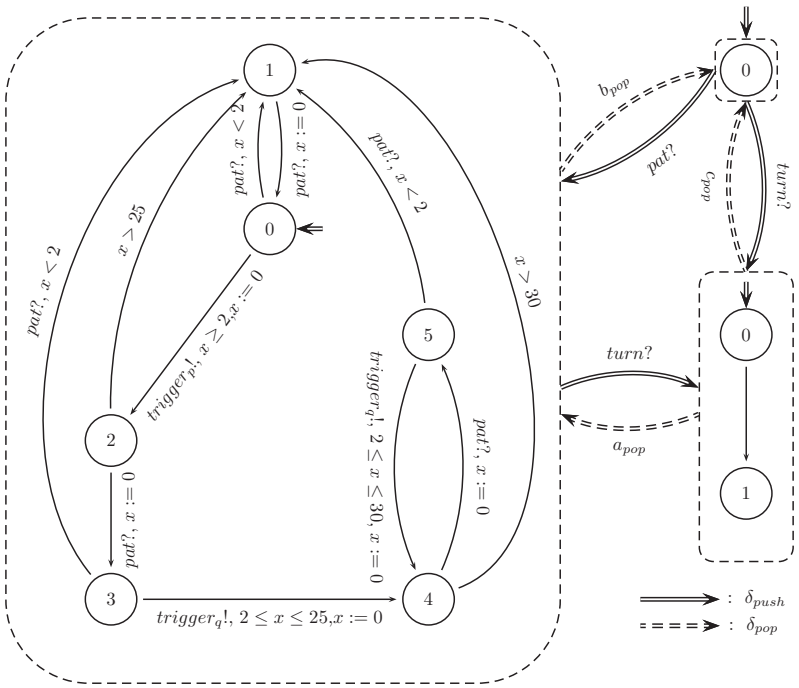


Fig. 4. An Interrupt Controller Represented by a Controller Automaton

5.3. *Scheduling Problem with Resource Sharing*

Scheduling is one of the most important issues in developing real-time systems. *Task automata* [14, 15], an extended version of timed automata with asynchronous processes are developed to model and analyze schedulability of tasks. These tasks can be periodic, sporadic, preemptive or non-preemptive. A task automaton is later translated to a parallel composition of two timed automata, one is to represent a *task system*, showing that time and frequency to release task instances, the other is to represent a *scheduler* that schedules the released task instances by some policies, e.g. FPS (fixed priority scheduling), EDF (earliest deadline first), SJF (shortest job first) [15, 14].

When these tasks share a mutex resource, scheduling problem becomes more difficult. For instance, a task with lower priority can block a task with higher priority when it acquires a shared resource earlier (by competition). In this subsection, we just simply illustrate that how to use controller automata to model such a task system. With the parallel composition with a scheduler by a timed automaton [15], the schedulability of the system can be analyzed.

Let $P(B, W, D) \in \mathscr{P}$ denote the type of a task, where B is the *best-case execution time*, W is the *worst-case execution time* and D is the *relative deadline* of P. A system should guarantee that each released task instances $p \in P(B, W, D)$ are executed at least

B time, at most W time within D time. In Figure 5, The (ordered) controller automaton depicts three tasks $A(4, 7, 15)$, $B(3, 5, 10)$, and $C(3, 7, 8)$ with priority that $A \succ B \succ C$, which compete with a mutex resource. $coming_A?$, $coming_B?$, and $coming_C?$ are used to describe the triggered conditions when three tasks instance are released, respectively. Their corresponding triggering actions will be provided by the scheduler. Controller automata can solve more complex scheduling problems rather that defined traditionally following the task type $P(B, W, D)$, e.g., schedulability analysis on nested interrupts [24], and analysis on *cyber-physical systems*, provided a task can be described as a timed automaton. In comparison, all tasks in task automata are considered atomically, which cannot analyze such complex systems directly.

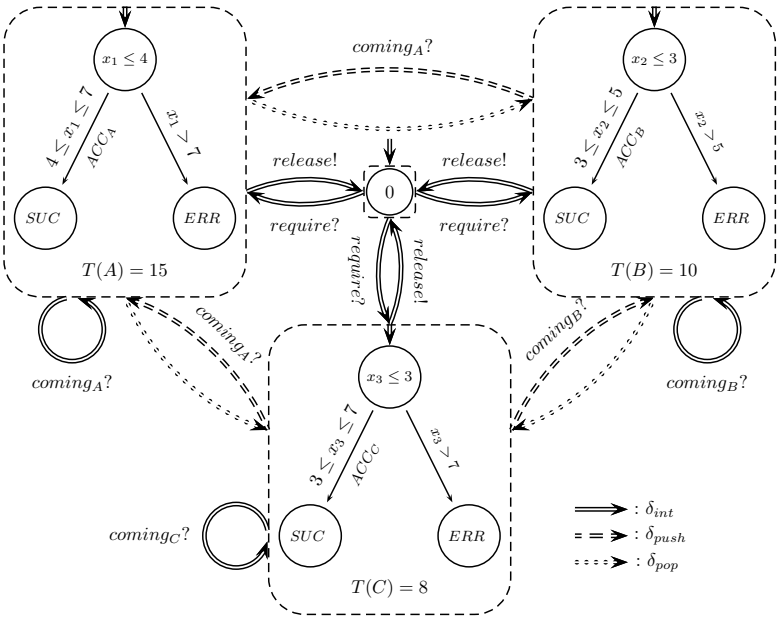


Fig. 5. Tasks with a Shared Resource

6. Related Work

Controller automata were first proposed in our previous work [24], as a formal model to represent and analyze real-time systems with nested interrupts. An analysis technique named *environmental simulation* in [24] adopted controller automata to treat interrupts. Together with an interrupt environment modeled as a timed automaton, and a scheduler as a timed automaton with semaphores [22], the system behaviors with nested interrupts were realized by a sequence of transitions with time. It was shown that the reachability to error states is practically solvable with the implementation of the environmental simulation by

Maude [11], a language and system supporting both equational and rewriting logic computation. Hence, various analyses, such as schedulability analysis could by solved by our implementation. We should point out that the environmental simulation in [24] is not a verification technique, which cannot guarantee termination generally. Actually, the ordered controller automata are expressive enough to represent interrupt controllers, so that we can adopt the algorithm given in this paper to guarantee termination of reachability analysis on real-time systems with nested interrupts.

Task automata [14, 13, 15], extended from timed automata, were a specific model for schedulability analysis. It assumed that time was dense, and tasks could come at any time, periodically or sporadically. There were many papers based on task automata. For example, when considering the fixed priority scheduling, two extra clocks were enough to represent a scheduler timed automaton for checking schedulability [16]. A tool, TIMES [5] was developed for schedulability analysis of tasks. In this approach, although dependence of tasks was considered, all tasks were represented atomically. It cannot describe complex interactive models, such that an object triggers another one during the running time, or an object may have multiple functions due to different environments, which can be represented by our controller automata.

BIP (Behavior, Interaction, Priority) system [18, 29] provided a language and a theory for incremental composition of heterogeneous components, ensuring correctness-by-construction for essential system properties, such as mutex, deadlock-freedom and progress. The construction of BIP system was composed of three layers. The lower layer described behavior. The intermediate layer included a set of connectors describing the interactions between transitions of the behavior. The upper layer was a set of priority rules describing scheduling policies for interactions. It also enabled formal verifications, providing formal tools such as D-Finder [6]. Controller automata have a similar viewpoint to model real-time systems in a component-based way. Behaviors are described by built-in timed automata, interactions are represented in synchronization and priorities are shown as mutexes in controller automata.

The *Updatable Timed Automata (UTA)* [7, 8, 9] extended timed automata, based on the possibility of updating the clocks in a more elaborate way than just resetting them to zero. They could provide more flexibility in representation and analysis of real applications. Authors gave undecidability and decidability results for several specific cases. The expressiveness of the UTA was also investigated [8]. The controller automata cannot be translated to any solved cases given in the paper. However, UTA provided us hints for the formal proof of (Un)decidability results for controller automata.

There were also several approaches to represent complex real-time systems, e.g., interrupts, with several considerations and restrictions. For example, In [12], interrupts, as well as other events, such as error recovery block, were represented as *additional interferences*. Each interference had bounded length of the time interval. Thus it restricted that there was only bounded number of interrupts that may happen. In [30], an interrupt handler was treated similar to a task. An interrupt controller was thus regarded as a scheduler. Hence a system was represented as a hierarchy tree, and interrupt handlers were assumed without

multiple functions and abilities to trigger tasks and other interrupts.

Abdeddaïm developed a methodology [1] for solving the *scheduling problem* based on timed automata. In her work, schedulers corresponded to paths in a timed automaton. The aim was to find an optimal scheduler, which corresponded to the shortest path. Furthermore, for the scheduling problem, there are many well-studied methods [10, 12], such as *rate monotonic scheduling* [25]. These researches arranged a scheduling policy under discrete time, and all tasks run at the same time periodically.

7. Conclusion

We presented a formal model, named *controller automata*, to model and analyze real systems with mutex components. A controller automaton is mainly designed to model a small scale real-time system in the low level, where control passing such as task switching and interrupt handling is explicitly handled. Although such behavior can be modeled by synchronization of parallel composition, our model can treat the primitive timed behavior in a more direct and simpler manner. In general, a controller automata properly extends a timed automaton. In the analysis of behavior, It is an advantage that a controller automaton can be composed with timed automata as an environment by parallel composition[24].

We have investigated a particular case of controller automata where an order of preemption over states is defined. An algorithm is presented to translate an ordered controller automaton to a timed automaton, and its correctness is shown by subbisimilarity. For this case, reachability problem on ordered controller automata is reduced to the reachability analysis of the timed automata, which was already implemented by several tools, e.g., UPPAAL [23].

In practice, an order of preemption is common such as task priority and interrupt levels. Although the class of ordered controller automata does not extend timed automata, it bridges the gap between the control mechanisms and the formal models. For instance, schedulability analysis on real-time embedded systems with nested interrupts in [24] by composing an interrupt environment that consists of an interrupt generator, interrupt handlers, and schedulers.

In this presentation, we incorporated a stack for control passing. This is primarily intended for using a controller automaton for a system with interrupts. We may consider more variations where other forms of control passing are in focus, such as a queue, seen in the task automata [14].

A toolkit to translate an ordered controller automaton to a timed automaton recognized by UPPAAL is currently under implementation. In addition, the future work also includes the theoretical investigation on the languages category recognized by controller automata, and the practical applications based on the (ordered) controller automata, such as schedulability analysis on cyber-physical systems.

Acknowledgement

The authors thank Prof. Ogawa Mizuhito and Prof. Jean-Francois Monin for fruitful discussions on the research. This research is supported by the National Nature Science

Foundation of China (60873034, 61011140074, 61003013, 61033002).

References

[1] Yasmina Abdeddaïm. *Scheduling with Timed Automata*. PhD thesis, Grenoble Institute of Technology, 2000.

[2] Yasmina Abdeddaïm and Oded Maler. Preemptive Job-Shop Scheduling Using Stopwatch Automata. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems(TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, 2002.

[3] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] Rajeev Alur and P. Madhusudan. Decision Problems for Timed Automata: A Survey. In *Formal Methods for the Design of Real-Time Systems*, Lecture Notes in Computer Science, pages 1–24. Springer-Verlag, 2004.

[5] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In *Proceedings of the 1st International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer-Verlag, 2003.

[6] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. D-Finder: A Tool for Compositional Deadlock Detection and Verification. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 614–619. Springer-Verlag, 2009.

[7] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are Timed Automata Updatable? In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer-Verlag, 2000.

[8] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Expressiveness of Updatable Timed Automata. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'00)*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer-Verlag, 2000.

[9] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable Timed Automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[10] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer-Verlag, 2004.

[11] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincolnand, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *Maude Manual (Version 2.4)*. http://maude.cs.uiuc.edu/maude2-manual/, 2008.

[12] R. I. Davis and A. Burns. Robust Priority Assignment for Fixed Priority Real-Time Systems. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 3–14. IEEE Computer Society, 2007.

[13] Christer Ericsson, Anders Wall, and Wang Yi. Timed Automata as Task Models for Event-Driven Systems. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, pages 182–189. IEEE Computer Society, 1999.

[14] Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task Automata: Schedulability, Decidability and Undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

[15] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability Analysis of Fixed-Priority Systems Using Timed Automata. *Theoretical Computer Science*, 354(2):301–317, 2006.

[16] Elena Fersman, Paul Pettersson, and Wang Yi. Timed Automata with Asynchronous Processes: Schedulability and Decidability. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer-Verlag, 2002.

[17] Yuxi Fu and Hao Lv. On the Expressiveness of Interaction. *Theoretical Computer Science*, 411:1387–1451, 2010.

[18] Gregor Gössler and Joseph Sifakis. Composition for Component-Based Modeling. *Science of Computer Programming*, 55(1-3):161–183, 2005.

[19] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[20] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata*. Morgan&Claypool Publishers, 2006.

[21] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Decidable Integration Graphs. *Information and Computation*, 150(2):209–243, 1999.

[22] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, pages 76–87. IEEE Computer Society, 1995.

[23] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[24] Guoqiang Li, Shoji Yuen, and Masakazu Adachi. Environmental Simulation of Real -Time Systems with Nested Interrupts. In *Proceedings of the 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering(TASE'09)*, pages 21–28. IEEE Computer Society, 2009.

[25] Chung Laung Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[26] Joseph Mattai. *Real-Time Systems: Specification, Verification, and Analysis*. Prentice Hall, 1995.

[27] Jennifer McManis and Pravin Varaiya. Suspension Automata: A Decidable Class of Hybrid Automata. In *Proceedings of the 6th International Conference on Computer Aided Verification(CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 105–117. Springer-Verlag, 1994.

[28] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[29] Marc Poulhiès, Jacques Pulou, Christophe Rippert, and Joseph Sifakis. A Methodology and Supporting Tools for the Development of Component-Based Embedded Systems. In *Proceedings of the 13th Monterey Workshop on Composition of Embedded Systems (Monterey'06)*, Lecture Notes in Computer Science, pages 75–96. Springer-Verlag, 2006.

[30] John Regehr, Alastair Reid, Kirk Webb, Michael Parker, and Jay Lepreau. Evolving Real-Time Systems Using Hierarchical Scheduling and Concurrency Analysis. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 25–36. IEEE Computer Society, 2003.