Modeling and Analysis of Real -Time Systems with Mutex Components

Guoqiang Li
BASICS, School of Software
Shanghai Jiao Tong University
Shanghai, China
li.g@sjtu.edu.cn

Xiaojuan Cai
BASICS, Department of Computer Science
Shanghai Jiao Tong University
Shanghai, China
cxj@sjtu.edu.cn

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

***Abstract*—Timed automata are popular for formally analyzing real-time systems. However, it is difficult to depict real-time systems with compositional components that interact with each other in a synchronization way or a mutex way. Synchronized components are modeled using parallel composition of timed automata by Larsen et al. [1]. This paper proposes controller automata to represent real-time systems with mutex components. In a controller automaton each state corresponds to a timed automaton with a built-in mechanism of relations, e.g., preemptions, in which every such timed automaton models a component of the real-time system. It is shown that given a strict partial order over states, an ordered controller automaton can be translated into a timed automaton. Various analyses are thus performed by checking the reachability to an error state.**

***Keywords*-real-time systems; mutex; formal analysis; controller automata; timed automata**

I. Introduction

Real-time systems, due to the requirements to complete their works 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 [2], such as *timed automata* [3], [4], have been proposed and widely used.

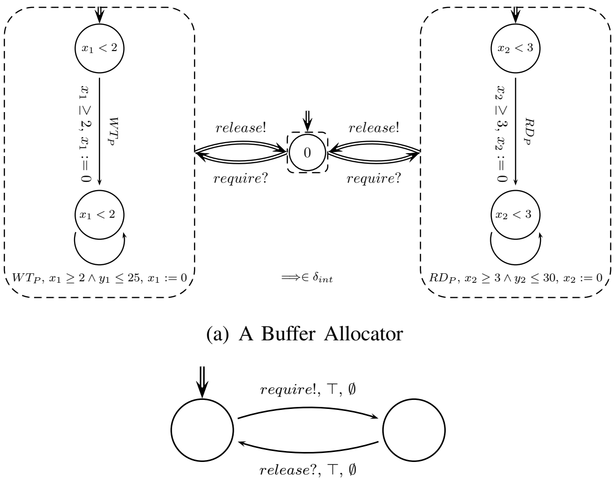
A real-time system usually consists of functionally independent components. In order to assure the response time sharing one processor, the control mechanism of these components is implemented either in a *synchronization* way or a *mutex* way. Each component behaves time-aware, which is naturally modeled as a timed automaton. In order to reason timed behavior of such a composite timed system, a special formal treatment for the control among components is useful where the composition is parallel conceptually but sequential in the actual execution.

As for synchronization, we refer to the pioneering work [1], 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 [5]. 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 triggered and triggering symbols synchronize on a channel and move simultaneously.

In general, mutex provides a form of sequential execution when a processor is regarded as a shared resource. Traditionally, mutex can be encoded by synchronization [5]. 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 to be executed. We represent the require/release operations of control directly by meta-level transitions. Although the synchronization gives a basic operation for mutex, the direct treatment of mutex has an advantage for reasoning timed behavior of the whole system in the sense that it leads to a simpler representation of time passage. In our model, mutex works with a stack that controls preemption/resumption of tasks. When preemption of tasks occurs, a task in execution releases the processor and a task taking over acquires the processor. As the preemptive task terminates, the waiting task in the stack will be resumed to acquire the processor.

This paper proposes *controller automata*, generalized from the model in our previous work [6], to describe behaviors and interactions of real-time systems with mutex components. In a controller automaton, each component is modeled as a timed automaton; A relation is adopted to show transitions 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 preemptions. 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 a simple example. 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 respectively. 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 that 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).


$x_1 < 2$
$x_1 \geq 2, x_1 := 0$
WT_P
$x_1 < 2$
WT_P, $x_1 \geq 2 \wedge y_1 \leq 25$, $x_1 := 0$
$release!$
$require?$
0
$release!$
$require?$
$\Longrightarrow \in \delta_{int}$
$x_2 < 3$
$x_2 \geq 3, x_2 := 0$
RD_P
$x_2 < 3$
RD_P, $x_2 \geq 3 \wedge y_2 \leq 30$, $x_2 := 0$
$require!$, $\top$, $\emptyset$
$release?$, $\top$, $\emptyset$

(a) A Buffer Allocator

(b) Semaphore

Figure 1. A Shared Buffer Example

When priority between processes are 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. 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 [7].

Paper Organizations: The rest of the paper is organized as follows. Section 2 briefly introduces timed automata, as a preliminary of our research. The formal definition and semantics of controller automata are given in Section 3. Decidability discussion of controller automata and an algorithm to translate an ordered controller automaton to a timed automaton are presented in Section 4. Section 5 briefly illustrates the usages of the controller automata by two examples. Section 6 concludes the paper.

II. Timed Automata

This section briefly reviews *timed automata* [3], [4].

Definition 1 (Time constraints)**.** Let $X = \{x_1, \ldots, x_n\}$ 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 $\nu : X \to \mathbb{R}^+$, where $\mathbb{R}^+$ denotes the set of non-negative real numbers. 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 have been first proposed in [3]. Later *timed safety automata*, introduced in [4], to 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 [8].

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 as same as the definition in [4].

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 sets of external symbols, internal symbols and 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 TA)**.** 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 TA 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 TA, in order to provide facilities for later theorems. This semantics is consistent with the original one [4], 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 language inclusion problem, are undecidable on timed automata, reachability problem for real-time systems [3], [4] is decidable.

Fact 1. *The reachability problem of timed automata is decidable [3], [4].*

Another important fact revealed in [1] is that timed automata composed by a *parallel composition* over a common set of external symbols does not enrich the expressiveness of timed automata.

Fact 2. *A parallel composition of two timed automata can be translated to a timed automaton.*

III. Controller Automata

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

A. Formal Definition

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

Definition 4 (Controller Automata)**.** A controller automaton 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.
- $M : S \rightarrow \mathscr{A}$ is a bijective map 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.
- $T : S \rightarrow \mathbb{R}^+$ 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.

B. 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 current timed automaton, store the current status, and begin to run the latter 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 $\mathtt{TimeLag} : \mathscr{A} \times Q(\mathscr{A}) \times \mathbb{R}^+ \rightarrow \mathscr{A}$ accepts a timed automaton, a control location on this timed automaton, a time interval, and returns a timed automaton, with the following definition, $\mathtt{TimeLag}(\mathcal{A}, q, t) = (E, H \cup \{r, l\}, Q \cup \{q^{id}\}, q_0, X \cup \{x_p\}, I', \delta')$, 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 translation. Assume a time lag t occurs on 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 to the control location q, and during the time t, the system can only stay within the fresh control location. This result is shown in (b) of Figure 2.

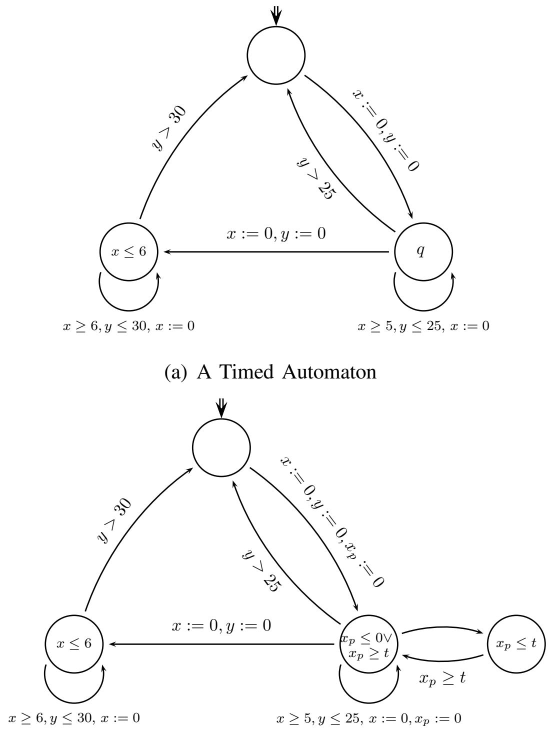
When the location q transits to the idle location q^{id}, it may not return back to q after t time units, due to the violation of $I'(q)$. It also cannot stay in q^{id}, since $I'(q^{id})$ is also violated after t time. Thus an empty timed automaton is produced, which explains that when some unexpected break happens, a system aborts the execution after resumed.

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

C. Semantics

We prepare a stack for configurations of a controller automaton, recording the state and the running control location when a push action performed, and resuming the running context when a pop action performed. By introduction of the time lag, the semantics of controller automata can be formally defined as follows:

Definition 5 (Semantics of Controller Automata)**.** A configuration for a controller automaton $\mathcal{C}$ is a tuple $(s, q, \nu, \kappa, \mathtt{S})$,



(a) A Timed Automaton

(b) Time Lag t on the control location q

Figure 2. Time Lag in Timed Automata

- 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*;
- S is the stack.
- Progress transitions:
  - $(s, q, \nu, \kappa, \texttt{S}) \xrightarrow{t}_{\mathscr{C}} (s, q, (\nu + t)[\kappa], \kappa, \texttt{S})$, if $(\nu + t)[\kappa] \models I(s)$, 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: $(s, q, \nu, \kappa, \texttt{S}) \xrightarrow{a}_{\mathscr{C}} (s, q', \nu[\lambda], \kappa, \texttt{S})$, 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: $(s, q, \nu, \kappa, \texttt{S}) \xrightarrow{a}_{\mathscr{C}} (s', q_0(M(s')), \nu[x_{run}], \kappa \backslash X(M(s')), (s, q) :: \texttt{S})$, and $M(t) := \texttt{TimeLag}(M(t), q, \nu(x_{run}))$ for all (t, q) in S, if $s \xrightarrow{a} s' \in \delta_{push}$.
  - Pop: $(s, q, \nu, \kappa, (s', q') :: \texttt{S}) \xrightarrow{a}_{\mathscr{C}} (s', q', \nu[x_{run}], \kappa \cup X(M(s)), \texttt{S})$, and $M(t) := \texttt{TimeLag}$ $(M(t), q, T(t))$ for all (t, q) in $(s', q') :: \texttt{S}$, if $s \xrightarrow{a} s' \in \delta_{pop}$ and $\nu(x_{run}) = T(s)$.
  - Inter-action: $(s, q, \nu, \kappa, \texttt{S}) \xrightarrow{a}_{\mathscr{C}} (s', q_0(M(s')), \nu[x_{run}], \kappa \backslash X(M(s')) \cup X(M(s)), \texttt{S})$, and $M(t) := \texttt{TimeLag}(M(t), q, T(t))$ for all (t, q) in the stack S, if $s \xrightarrow{a} s' \in \delta_{int}$, $\nu(x_{run}) = T(s)$.

Each transition can be intuitively explained as follows,

- A progress transition of a control automaton is that of the time automaton of the current running state.
- An intra-action transition is a discrete transition of the timed automaton of the current running state.
- A push transitions 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}.
- A pop transition pops the previous state and control location from the stack after the execution of the expected running time, 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 $T(s)$.
- An inter-action transitions transits to the initial location of the time automaton in the latter state after the execution of the expected running time. Simultaneously, 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/poped content of the stack is decided statically in a pushdown automaton, while dynamically during the running time in a controller automaton.

D. Synchronization Representation

Similar to timed automata [1], synchronization of a controller automaton and a timed automata is represented as parallel composition of two automata, borrowed the idea from CCS [5].

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 6. 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 $\nu[\lambda \cup \cup \{x_{run}\}] \models I(q_a', q_0(M(s')))$.
  - $(s, (q_a, q'), \nu, \kappa, \mathtt{S}) \xrightarrow{\tau}_{\mathscr{P}} (s', (q_a', q_0(M(s'))), \nu[\lambda \cup \{x_{run}\}], \kappa \backslash X(M(s')), (s, q) :: \mathtt{S})$, 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 $\nu[\lambda \cup \cup \{x_{run}\}] \models I(q_a', q_0(M(s')))$.

The transitions for pop and internal symbols in discrete transitions are elided.

A parallel composition between a timed automaton $\mathcal{A}$ and a controller automaton $\mathcal{C}$ can be translated to parallel compositions of the $\mathcal{A}$ 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.*

The definition of parallel composition between timed automata and controller automata allows us to model compositional real-time systems, including both synchronized and mutex components.

IV. A Subclass of Controller Automata

A. Decidability Discussions

Timed issues bring many difficulties for decidability problems on automata [9]. For instance, the inclusion problem on timed automata is undecidable; 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 [10], [11].

Although there are frozen clocks in the controller automata, the expressiveness of time in controller automata does not go beyond timed automata. 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 when frozen 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 infinite insertions of fresh control locations and fresh clocks into timed automata when time lags occur. Such dynamic insertions, although easy to be understood and be adopted, lead to difficulties for analysis and proof. It is easy to show that these insertions can be avoided by the introduction of *timed automata with addition*, 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* [12], is undecidable. By the similar reason, the reachability problem on timed automata with addition is also undecidable.

In [6], it is shown that the reachability of error states is practically solvable with the implementation of controller automata by Maude. 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 unbounded but finite number of fresh control locations and clocks inserted to timed automata within a controller automaton, the length of stack is also finite. Hence, an ordered controller automata can be translated to a timed automaton.

B. Ordered Controller Automata

Given a controller automaton $\mathcal{C} \in \mathscr{C}$, we construct a strict partial order (irreflexive, asymmetric and transitive) on the set of states, $(S(\mathcal{C}), \prec)$.

The transitions δ is restricted as follows:

- $s_1 \prec s_2$, 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 $s_1 \xrightarrow{a} s_2 \in \delta_{int}$.

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 1. *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 $\mathcal{N}$. ■

Remark 1. 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.

C. 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 $\hat{q} = q_1 \ldots q_i \ldots q_j \ldots q_n$, if $q_i \in Q(M(s_i))$ and $q_j \in Q(M(s_j))$, $s_i \prec s_j$ holds.
- $q_{\mathcal{A}}^0 = (q_0(M(s_0)), \varepsilon)$, 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 $q_i \in Q(M(s_i))$, $I_{\mathcal{A}}((q_i, \hat{q}_i)) = I_i(q_i) \wedge (x_{run} \le T(s_i))$.
- $\delta_{\mathcal{A}}$ 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 $X_j = X(M(s_j))$.
  - $(q, \hat{q}q') \xrightarrow{a,(x_{run} \ge T(s_i)),\{x_{run}\}} (q', \hat{q})$, if $q \in Q(M(s_i))$, $q' \in Q(M(s_j))$, and $s_i \xrightarrow{a} s_j \in \delta_{pop}$.
  - $(q, \hat{q}) \xrightarrow{a,(x_{run} \ge T(s_i)),X_j \cup \{x_{run}\}} (q', \hat{q})$, 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 [13] and used to capture the operational soundness and completeness between variants of π-calculus.

Definition 7 (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.

The first requirement implies the preservation of actions of a controller automaton, while the second ensures the reflection of these actions. We are now ready to show the correctness of our translation of an ordered controller automaton.

Theorem 1 (Operational Correspondence)**.** *Any ordered controller automaton $\mathcal{C}$ is subbisimilar to its corresponding timed automaton $\mathcal{A}^{\mathcal{C}}$.*

The proof is simple, 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 $\mathcal{C}$, and $((q, \hat{q}), \nu)$ is a configuration of $\mathcal{A}^{\mathcal{C}}$. It is proved that after their respective corresponding transitions, the pair of the two results also belongs to $\mathcal{S}$. Thus $\mathcal{S}$ is a subbisimilarity defined in Definition 7. 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).

V. Examples

This section illustrates how to adopt controller automata to represent mutex real-time systems by two examples. A real-time system with nested interrupts, as a more complicate and practical example, is shown in [6].

A. A Preemptive Buffer Allocator

Let's continue our buffer example in Section 1. Assume that the writing process has the higher priority than the reading process. When the reading process occupies 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 guarantees to receive the first datum within 20 time unit. 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 (omit here) and translating the ordered controller automaton to a timed automaton, we can show that the error control location ERR is reachable.

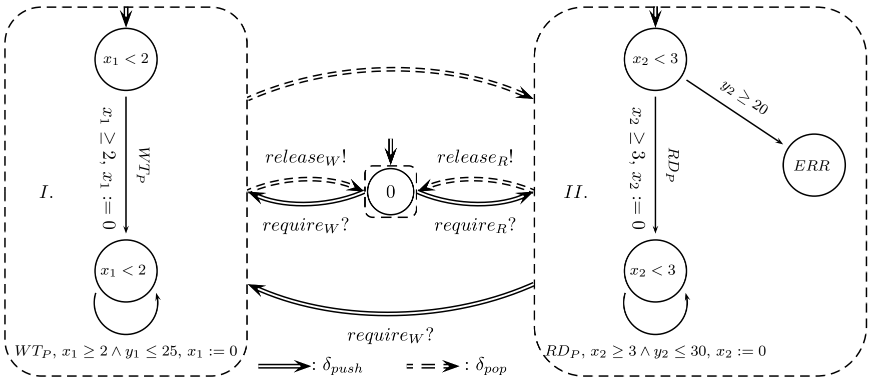


Figure 3. A Preemptive Buffer Allocator

B. 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 applied the 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 4, 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 [6], and analysis on *cyber-physical system*), 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.

VI. Conclusion

We presented a formal model, named *controller automata*, to model and analyze real systems with mutex components. The analysis technique of controller automata was also investigated, showing that with a strict partial order over the state set, the reachability problem of this subclass of controller automata, named *ordered controller automata*, became decidable. An algorithm was proposed to translate an ordered controller automaton to a timed automaton, and its correctness was proved. Thus reachability problem on ordered controller automata was reduced to the reachability analysis of the timed automaton, which was already implemented by several tools, e.g., UPPAAL [7].

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 partly supported by the NSFC-JSPS joint project "*Foundation of Program Analyses for Multi-thread Programming*".

References

[1] K. G. Larsen, P. Pettersson, and W. Yi, "Compositional and Symbolic Model-Checking of Real-Time Systems," in *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*. IEEE Computer Society, 1995, pp. 76–87.

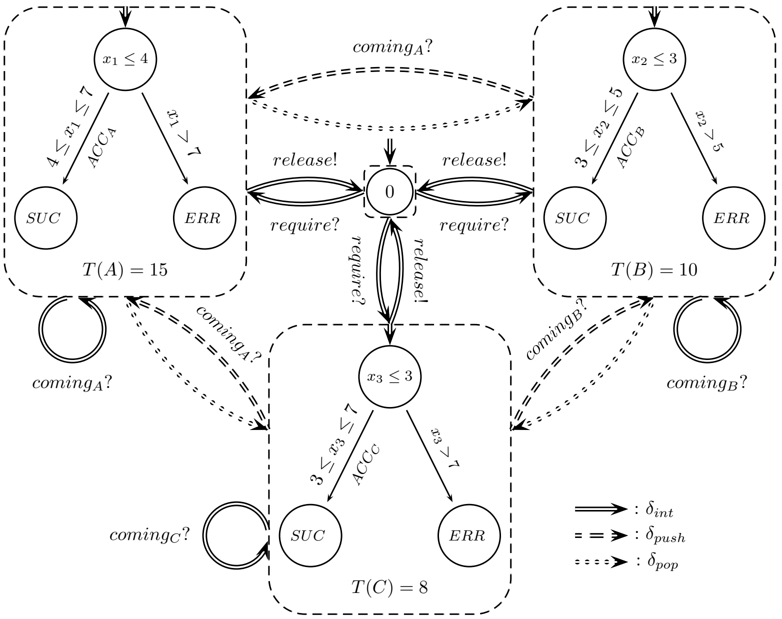


Figure 4. Tasks with a Shared Resource

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

[3] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[4] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," *Information and Computation*, vol. 111, no. 2, pp. 193–244, 1994.

[5] R. Milner, *Communication and Concurrency*, ser. International Series in Computer Science. Prentice Hall, 1989.

[6] G. Li, S. Yuen, and M. 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)*. IEEE Computer Society, 2009, pp. 21–28.

[7] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a Nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[8] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, *The Theory of Timed I/O Automata*. Morgan&Claypool Publishers, 2006.

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

[10] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine, "Decidable Integration Graphs," *Information and Computation*, vol. 150, no. 2, pp. 209–243, 1999.

[11] Y. Abdeddaïm and O. 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)*, ser. Lecture Notes in Computer Science, vol. 2280. Springer-Verlag, 2002, pp. 113–126.

[12] J. McManis and P. Varaiya, "Suspension Automata: A Decidable Class of Hybrid Automata," in *Proceedings of the 6th International Conference on Computer Aided Verification(CAV'94)*, ser. Lecture Notes in Computer Science, vol. 818. Springer-Verlag, 1994, pp. 105–117.

[13] Y. Fu and H. Lv, "On the Expressiveness of Interaction," *Theoretical Computer Science*, 2009, to appear.

[14] E. Fersman, P. Krcal, P. Pettersson, and W. Yi, "Task Automata: Schedulability, Decidability and Undecidability," *Information and Computation*, vol. 205, no. 8, pp. 1149–1172, 2007.

[15] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Schedulability Analysis of Fixed-Priority Systems Using Timed Automata," *Theoretical Computer Science*, vol. 354, no. 2, pp. 301–317, 2006.