Deductive Schedulability Verification Methodology of Real-Time Software using both Refinement Verification and Hybrid Automata

Deductive Schedulability Verification Methodology of Real-Time Software using both Refinement Verification and Hybrid Automata

Satoshi Yamane
Dept. of Information Engineering
Kanazawa University
Kanazawa City, Japan
Tel:+81.76.234.4856, Fax:+81.76.234.4900
Email:syamane@is.t.kanazawa-u.ac.jp

Abstract

Real-time software runs over real-time operating systems, and guaranteeing qualities is difficult. As timing constraints and resource allocations are strict, it is necessary to verify schedulability, safety and liveness properties. In this paper, we formally specify real-time software using hybrid automata and verify its schedulability using both deductive refinement theory and scheduling theory. In this case, the above real-time software consists of periodic processes and a fixed-priority preemptive scheduling policy on one CPU. Using our proposed methods, we can uniformally and easily specify real-time software and verify its schedulability based on hybrid automata. Moreover, we can verify its schedulability at design stage.

1 Introduction

Recently almost microprocessors are used in embedded systems. Real-time software runs in embedded systems. As real-time software is reactive and concurrent, and its timing conditions are strict, it is difficult to design real-time software.

In this paper, we formally specify real-time software using hybrid automata, and propose verification method of its schedulability using both deductive refinement theory and scheduling theory. Using our proposed methods, we can uniformally and easily specify real-time software and verify its schedulability based on hybrid automata. Moreover, we can verify its schedulability at design stage.

In this paper, we model real-time software consisting of periodic tasks and a preemptive scheduling with fixed priorities. In order to easily specify real-time software and verify its schedulability, we do not consider resource allocations and delays of dispatches. Moreover, we assume that tasks are independent. But we can easily extend our real-time software with these restrictions. By our proposed methods, we can combine specification of real-time software with its schedulability verification based on hybrid automata.

Recently many researches about formal specification and verification of real-time software have been studied. Existing main researches are as follows:

1. R. Alur and T.A. Henzinger have specified a preemptive scheduler by hybrid automata [1]. Moreover, they have verified the safety and liveness properties using model checking. By hybrid automata, we can easily specify a preemptive scheduler. But they have not proposed the schedulability verification.

2. S. Vestal has proposed restricted hybrid automata and showed the reachability problem is decidable [2]. He has showed the schedulability verification by verifying the reachability problem of a preemptively scheduled task. But he has not divided tasks and schedulers from preemptively scheduled tasks.

3. V. Braberman has proposed automatic schedulability verification of a preemptive scheduler based on timed automata [3]. As he has specified time constraints as maximun and minimum distances between events, his method is a conservative.

4. J. Sifakis has proposed timed automata with priorities. He has specified scheduling policies by priority rules [4]. He has not directly specified schedulers.

5. Z. Liu has specified and verified real-time software and schedulers by deductive verification based on TLA [5]. As he has specified preemptive schedulers by TLA, it is very difficult to specify and verify them.

As we proposed the schedulability verification method of real-time software by the integration of refinement and scheduling theory based on hybrid automata, our study is quite different from existing studies.

In general, in order to specify a preemptive scheduler, we must record accumulated computer time. If we specify recording accumulated computer time by hybrid automata, we can easily and simply specify it. (At design stage, we must assume computer time.) But if we specify recording accumulated computer time by timed automata, we must conservatively specify it by complex styles. In this paper, we model real-time software consisting of periodic tasks and a preemptive scheduling with fixed priorities. We specify this real-time software by hybrid automata, and propose the schedulability verification method based on hybrid automata.

The paper is organized as follows: In section 2, we present specification of real-time software. In section 3, we present

Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03)
0730-3157/03 $ 17.00 © 2003 IEEE

our schedulability verification method. Finally, in section 4, we present conclusions.

2 Specification of real-time software

In this section, we introduce hybrid automata, and specify real-time software.

2.1 Hybrid automata

We extend hybrid automaton [1] by distinguishing observable variables and unobservable variables.

First, we define hybrid automata.

Definition 1 *Hybrid automaton* $\mathbf{A}$ *is a 10-tuple* $(\vec{x}, V, local, inv, dif, E, act, Label, syn, E_\Theta)$ *as follows:*

1. **Data variables** *: A finite vector* $\vec{x} = (x_1, x_2, \ldots, x_n)$ *of real-valued data variables. The size of* n *of* $\vec{x}$ *is called the dimension of* $\mathbf{A}$*. A data state is a point* $\vec{s} = (s_1, \ldots, s_n)$*, equivalently, a function that assigns to each data variable* x_i *a real value* $s_i \in \mathbf{R}$*. A data predicate* p *defines the data region* $\ll p \gg \subseteq \mathbf{R}^n$*, where* $\vec{s} \in \ll p \gg$ *iff* $p[\vec{x} := \vec{s}]$ *is true. For each data variable* x_i*, we use the dotted variable* $\dot{x}_i$ *to denote the first derivative of* x_i*. A differential inclusion is a convex polyhedron in* $\mathbf{R^n}$*. A rate predicate is a convex linear formula over the vector* $\vec{\dot{x}} = (\dot{x}_1, \ldots, \dot{x}_n)$ *of dotted variables. The rate predicate* r *defines the differential inclusion* $\ll r \gg \subseteq \mathbf{R^n}$*, where* $\vec{\dot{s}} \in \ll r \gg$ *iff* $r[\vec{\dot{x}} := \vec{\dot{s}}]$ *is true.*

2. **Control locations** *: A finite set* V *of vertices called control locations.*

3. **Local variables** *:* $local$ *is a finite set of local variables.* $local$ *is a part of* $\vec{x}$*.* $local$ *is not observable from other automata.*

4. **Location invariants** *: A labeling function* inv *that assigns to each control location* $v \in V$ *a convex data predicate* $inv(v)$*, the invariant of* v*.*

5. **Continuous activities** *: A labeling function* $dif(v)$ *that assigns to each control location* $v \in V$ *a rate predicate* $dif(v)$*, the activity of* v*. The activities constrain the rates at which the values of data variables change: while the automaton control resides in the location* v*, the first derivatives of all data variables stay within the differential inclusion* $\ll dif(v) \gg$*.*

6. **Transitions** *: A finite multiset* E *of edges called transitions. Each transition* $(v, v\prime)$ *identifies a source location* $v \in V$ *and a target location* $v\prime \in V$*.*

7. **Discrete actions** *: A labeling function* act *that assigns to each transition* $e \in E$ *an action predicate* $act(e)$*, the action of* e*. The automaton control can proceed from the location* v *to the location* $v\prime$ *via the transition* $e = (v, v\prime)$ *only when the action* $act(e)$ *is enabled. If* $act(e)$ *is enabled in the data state* $\vec{s}$*, then the values of all data variables change nondeterministically from* $\vec{s}$ *to some point in the data region* $\ll act(e) \gg (\vec{s})$*.*

8. **Synchronization labels** *: A finite set* $Label$ *of synchronization labels and a labeling function* syn *that assigns to each transition* $e \in E$ *a set of synchronization labels from* $Label$*. The set* $Label$ *is called the alphabet of* $\mathbf{A}$*. The synchronization labels are used to define the parallel composition of two automata. If both automata share a synchronization label* a*, then each* a*-transition of one automaton must be accompanied by an* a*-transition of the other automaton.*

9. **Entry edge** *: An entry edge,* E_Θ*, that has no originating location, but an entry location* $v_i \in V$*.* E_Θ *is labeled by the form* $x_1 = c_1 \wedge x_2 = c_2 \wedge \ldots \wedge x_n = c_n$*, where* $c_1, c_2, \ldots, c_n$ *are real values.*

Next, we define parallel composition of hybrid automata.

Definition 2 *Let* $A_1 = (\vec{x}_1, V_1, local_1, inv_1, dif_1, E_1, act_1, Label_1, syn_1, E_{\Theta 1})$ *and* $A_2 = (\vec{x}_2, V_2, local_2, inv_2, dif_2, E_2, act_2, Label_2, syn_2, E_{\Theta 2})$ *be two hybrid automata of dimensions* n_1 *and* n_2*, respectively. The parallel composition of* A_1 *and* A_2 *is* $A = (\vec{x}_1 \cup \vec{x}_2, V_1 \times V_2, local_1 \cup local_2, inv, dif, E, act, Label_1 \cup Label_2, syn, E_\Theta)$*.*

1. *Each location* $(v, v\prime)$ *in* $V_1 \times V_2$ *has the invariant* $inv(v, v\prime) = inv_1(v) \wedge inv_2(v\prime)$ *and the activities* $dif(v, v\prime) = dif_1(v) \wedge dif_2(v\prime)$*.*

2. $E_\Theta = E_{\Theta 1} \wedge E_{\Theta 2}$*.*

3. E *contains* $e = ((v_1, v_1\prime), (v_2, v_2\prime))$ *iff*

   (a) $v_1 = v_1\prime$ *and there is a transition* $e_2 = (v_2, v_2\prime) \in E_2$ *with* $Label_1 \cap syn_2(e_2) = \emptyset$ *,where* $act(e) = act_2(e_2)$ *and* $syn(e) = syn_2(e_2)$*; or*

   (b) *there is a transition* $e_2 = (v_2, v_2\prime) \in E_2$ *with* $Label_1 \cap syn_2(e_2) = \emptyset$*, and* $v_2 = v_2\prime$ *,where* $act(e) = act_1(e_1)$ *and* $syn(e) = syn_1(e_1)$*; or*

   (c) *there is a transition* $e_1 = (v_1, v_1\prime) \in E_1$ *and* $e_2 = (v_2, v_2\prime) \in E_2$ *such that* $syn_1(e_1) \cap Label_2 = syn_2(e_2) \cap Label_1$ *, where if* $act_1(e_1) = (\vec{y_1}, q_1\prime)$ *and* $act_2(e_2) = (\vec{y_2}, q_2\prime)$*, then* $act(e) = (\vec{y_1} \cup \vec{y_2}, q_1\prime \wedge q_2\prime)$ *and* $syn(e) = syn_1(e_1) \cup syn_2(e_2)$*.*

2.2 Specification

In this paper, we model real-time software consisting of periodic tasks and a preemptive scheduling with fixed priorities.

2.2.1 Specification of periodic tasks

First, we specify periodic tasks by hybrid automata. If we determine period, execution time and deadline of a periodic task, we can specify the periodic task. Two periodic tasks are shown in figure 1.

1. Figure 1 (1) represents as a hybrid automaton a periodic task1 of period T1, execution time E1, and deadline D1 ($0 < E1 \leq D1 \leq T1$). Figure 1 (2) represents as a hybrid automaton a periodic task2 of period T2, execution time E2, and deadline D2 ($0 < E2 \leq D2 \leq T2$).

2. The periodic task1 has three states, sleep1, wait1, and execute1 where task1 is respectively, sleeping, waiting and executing. The periodic task2 has four states, sleep2, wait2, execute2, and preempt2 where task2 is respectively, sleeping, waiting, executing, and preemptive.

3. The periodic task1 has three events, arrive1, begin1, and end1 where arrive1 means the arrival of execution requirement, begin1 means the start of execution, and end1 means the end of execution. The periodic task2 has three events, arrive2, begin2, and end2 where arrive2 means the arrival of execution requirement, begin2 means the start of execution, and end2 means the end of execution.

4. The timer x1 and x2 are used to measure execution time while the timer t1 and t2 are used to measure the time elapsed since task arrivals.

In this case, task1 has higher priority than task2.

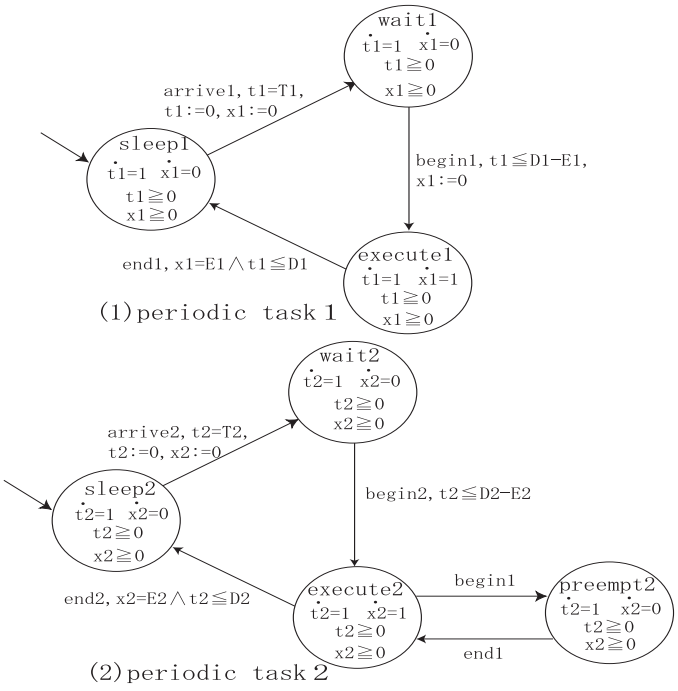

Figure 1. Specification of periodic tasks

2.2.2 Specification of preemptive scheduler

Next, we specify a preemptive scheduler, which controls both task1 and task2 as shown in figure 2. In this figure 2, rate monotonic scheduler [6] is specified by hybrid automaton. The preemptive scheduler has four states, idle, task1, task2, and preempt where tasks are idle, task1 or task2 is executing and task1 is preemptive.

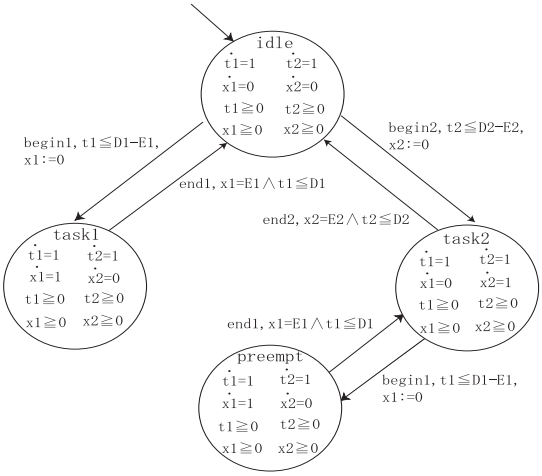

Figure 2. Specification of preemptive scheduler

2.2.3 Timing diagram

Next, we show the behaviors of task1 and task2 by timing diagram in figure 3. In this case, task1 has higher priority than task2. Execution of task2 is pre-empted by task1.

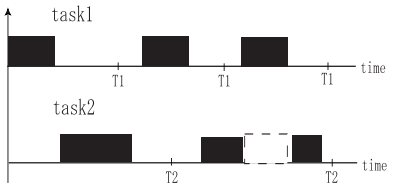

Figure 3. Timing diagram

3 Schedulability verification of real-time software

In this section, we introduce refinement verification method of hybrid automata and define schedulability verification method.

3.1 Refinement verification method of hybrid automata

We transform hybrid automaton into phase transition system, and verify refinement over phase transition systems [7].

3.1.1 Basic concept

Let Var be a set of typed variables. The allowed types include boolean, integer and real. We view the booleans and the integers as subsets of the reals. Let $\mathbf{R^n}$ be the set of real numbers. A state $\sigma : Var \rightarrow \mathbf{R^n}$ is a type-consistent interpretation of the variables in Var. We write Σ_V for the set of states. Time is modeled by the nonnegative real line $\mathbf{R}^+$. An interval $[a, b)$, where $a, b \in \mathbf{R}^+$ and $a < b$, is the set of points $t \in \mathbf{R}^+$ such that $a \leq t < b$. Let $I = [a, b)$ be an interval. A function $f : I \rightarrow \mathbf{R^n}$ is piecewise smooth on I if

1. At a, the right limit and all right derivatives of f exist;
2. At all points $t \in I$, the right and left limits and all right and left derivatives of f exist, and f is continuous either from the right or from the left;
3. At b, the left limit and all left derivatives of f exist.

The function $f, g : I \rightarrow \mathbf{R^n}$ are indistinguishable on I if they agree on almost all points $t \in I$.

A phase $P =< I, f >$ over V is a pair consisting of

1. a nonempty left-closed right-open interval $I = [a, b)$
2. a type-consistent family $f = \{f_x | x \in Var\}$ of functions $f_x : I \rightarrow \mathbf{R^n}$ that are piecewise smooth in I and assign to each point $t \in I$ a value for the variable $x \in Var$

It follows that the phase P assigns to every real-valued time $t \in I$ a state $f(t) \in \Sigma_V$.

We write

$$\overrightarrow{P} = lim_{t \rightarrow a}\{f(t) | a < t < b\} \quad (1)$$

for the left-end limit state $\overrightarrow{P} \in \Sigma_V$ of the phase P, and

$$\overleftarrow{P} = lim_{t \rightarrow b}\{f(t) | a < t < b\} \quad (2)$$

for the right-end limit state $\overleftarrow{P} \in \Sigma_V$ of P.

3.1.2 Phase transition system

We extend Pnueli's phase transition system [8] by distinguishing observable variables and unobservable variables.

Definition 3 *Phase transition system* $\mathbf{M} = (Var, L, \Phi, \Theta, \mathcal{T})$ *consists of five components:*

1. *A finite set Var of state variables.*
2. *A finite set $L \subseteq Var$ of local variables, which are unobservable variables.*
3. *A finite set Φ of phase invariants over Var. Each phase invariant $\phi \in \Phi$ is presented by an assertion of the form $\rho_\phi(Var, \dot{V}ar)$, referring to the state variables and their derivatives.*
4. *An initial condition, Θ, which is a state formula over V that specifies the initial value of the variables at the left end of the first phase in computations.*
5. *A set $\mathcal{T}$ of transitions. Each transition $\tau \in \mathcal{T}$ is associated with an assertion $\rho_\tau(Var, Var\prime)$, relating values at the right-end limit state of a phase to the values at the left-end of a successor phase.*

A phase sequence $\overline{P} = P_0, P_1, P_2, \ldots$ is a finite or infinite sequence of adjacent phases, where $P_i =< [a_i, a_{i+1}), f_i >$ for all $i \geq 0$.

A phase sequence is a computation(run) of the $\mathbf{M}$ if it is equivalent to a phase sequence $\overline{P} = P_0, P_1, P_2, \ldots$ that satisfies the following conditions:

1. Initiality : If $P_0 = [a, b)$ then Θ holds at a.
2. Continuous activities : For all $0 \leq i < |\overline{P}|$, there is a phase invariant $\rho_\phi \in \Phi$ such that P_i is a ϕ-phase.
3. Discrete transitions : For all $0 \leq i < |\overline{P}| - 1$, there is a transition $\tau \in \mathcal{T}$ such that $\rho_\tau(\overrightarrow{P_i}[Var]\overleftarrow{P_{i+1}}[Var])$ holds.
4. Divergence : $\overline{P}$ is divergent.

Next, we define the transformation of hybrid automaton into phase transition system.

Definition 4 *Hybrid automaton* $\mathbf{A} = (\vec{x}, V, local, inv, dif, E, act, Label, syn, E_\Theta)$ *is transformed into phase transition system* $\mathbf{M} = (Var, L, \Phi, \Theta, \mathcal{T})$ *as follows:*

1. $Var = \pi \cup \vec{x}$*, where π is a variable, which denotes location V.*
2. $L = local$.
3. $\Phi = \{\phi_{l_i} | l_i \in V\}$*, where for each $l_i \in V$,*
   $\rho_{\phi_{l_i}} : inv(l_i) \wedge (\pi = l_i) \wedge dif(l_i)$
4. $\Theta = (x_1 = c_1 \wedge \ldots \wedge x_n = c_n) \wedge (\pi = l_i) \wedge inv(l_i)$*, where l_i is the entry location and $(x_1 = c_1 \wedge \ldots \wedge x_n = c_n)$ is a label of the entry edge.*
5. $\mathcal{T} = \{\tau_{(l_i, l_j)} | (l_i, l_j) \in E\}$*, where for $e = (l_i, l_j) \in E$ such that $syn(e) = label \in Label$,*
   $\rho_{\tau_{(l_i, l_j)}} : act(e) \wedge \pi = l_i \wedge \pi\prime = l_j$

3.1.3 Axioms of refinement verification

Consider two phase transition systems, M^C and M^A, to which we refer as the concrete and abstract specification, wishing to prove that M^C refines M^A. Phase transition system $\mathbf{M}$ is a 5-tuple $(Var, L, \Phi, \Theta, \mathcal{T})$. $L \subseteq Var$ is a finite set of local variables, which are unobservable from other systems. In this paper, we assume that only internal behaviors of abstract specification are refined into concrete specification. If both every observable phase of M^C is some phase of M^A and every observable transition of M^C is some transition of M^A, M^C refines M^A , and we denotes it by $M^C \sqsubseteq M^A$.

Definition 5 *For $M^C = (Var^C, L^C, \Phi^C, \Theta^C, \mathcal{T}^C)$ and $M^A = (Var^A, L^A, \Phi^A, \Theta^A, \mathcal{T}^A)$, and a substitution α: $L^A \leftarrow \varepsilon(Var^C)$,*

1. $\Theta^C \rightarrow \Theta^A[\alpha]$
2. *For each ${\phi_i}^C \in \Phi^C$ and ${\tau_j}^C \in \mathcal{T}^C$, $(i = 1, \ldots, n, j = 1, \ldots, m)$!'*
   $\exists k.\ \rho_{{\phi_i}^C} = \rho_{{\phi_k}^A}[\alpha]$
   and
   $\exists l.\ \rho_{{\tau_j}^C} \rightarrow \rho_{{\tau_l}^A}[\alpha]$
3. $---------- \quad M^C \sqsubseteq M^A$

If some α exists , and 1. and 2. are satisfied, 4. such as $\sqsubseteq$ can be defined.

Premise 1 requires that Θ^C implies $\Theta^A[\alpha]$. Premise 2 ensures that every phase invariant of $\rho_{\phi_i C}$ is equal to some phase invariant of $\rho_{\phi_k A}[\alpha]$, and ensures that every transition of $\rho_{\tau C}$ can be simulated by some transition of $\rho_{\tau A}[\alpha]$. Conclusion 4 means $M^C \sqsubseteq M^A$.

Let α: $L^A \leftarrow \varepsilon(Var^C)$ be a substitution that replaces each local abstract variable $X \in L^A$ by an expression $\varepsilon(Var^C)$ over the concrete variables. We denote $X_\alpha(Var^C)$ when we assign an expression $\varepsilon(Var^C)$ over the concrete variables to $X \in L^A$.

α can give a mapping over phases as follows: Let P^C be a phase which appears in a computation of M^C. We refer to P^C as a concrete phase. The abstract $P^A = m_\alpha(P^C)$ corresponding to the concrete phase P^C is such that the value of each local abstract variable $X \in L^A$ in P^A is the value of the expression $X_\alpha(Var^C)$ when evaluated in P^C. We say that m_α is a refinement mapping from M^C to M^A. This refinement mapping is equal to Abadi's refinement mapping [9].

3.2 Schedulability verification

We will combine specification of real-time software with scheduling theory based on hybrid automata. In order to verify schedulability, we integrate refinement verification with scheduling theory.

First, we present existing scheduling theory [10]. We represents as a hybrid automaton a periodic task i of period Ti, execution time Ei, and deadline $Di(i = 1, \ldots, n)$. Mathai Joseph has proposed the worst-case response time Ri of task i as recursive equation. The $(n+1)th$ worst-case response time $Ri^{(n+1)}$ for task i is as follows:

$$Ri^{(n+1)} = Ei + \sum_{j=1}^{i-1} \lceil \frac{Ri^{(n)}}{Tj} \rceil \times Ej$$

,where $Ri^{(0)}$=Ei and $Ri = \lim_{n\to\infty} Ri^{(n)}$.

Definition 6 *Real-time software is schedulable if the following two conditions are satisfied:*

1. *For $\forall i$, $Ri \leq Di$(i=1,..,n) holds true.*
2. *$M^C \sqsubseteq M^A$, where M^A is the parallel composition of periodic tasks, M^C is the parallel composition of periodic tasks and scheduler.*

3.3 Example

We show example of schedulability verification of periodic tasks in section 2.

3.3.1 Phase transition system of periodic tasks

First, we construct parallel composition of hybrid automata of periodic tasks as shown in figure 4.

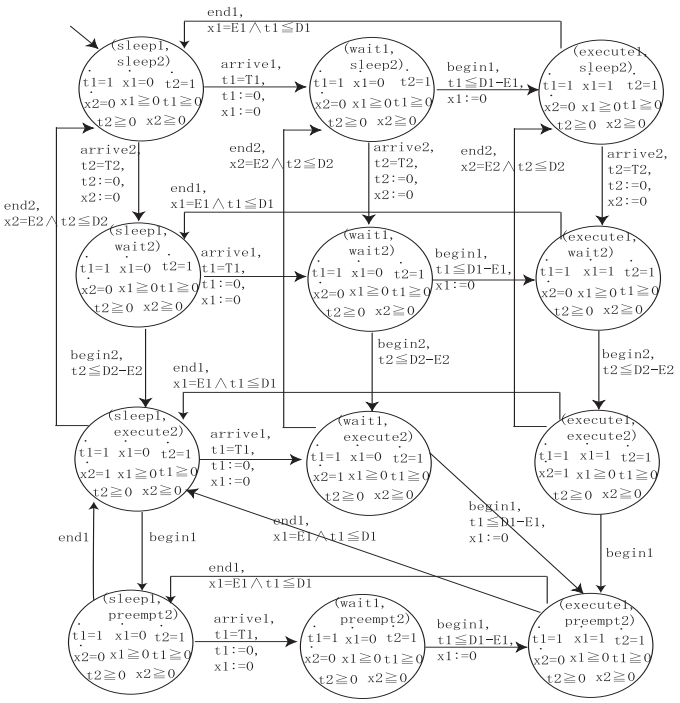

Figure 4. Hybrid automaton of periodic tasks

Next, we construct phase transition system from parallel composition of hybrid automata. The phase transition system of periodic tasks $M^A = (Var^A, L^A, \Phi^A, \Theta^A, \mathcal{T}^A)$ consists of five components as follows:

1. $Var^A = \{t_1, t_2, x_1, x_2, {\pi_1}^A, {\pi_2}^A\}$.
2. $L^A = \{{\pi_1}^A, {\pi_2}^A\}$.
3. $\Phi^A = \{\phi_{(sleep_1,sleep_2)}, \phi_{(sleep_1,wait_2)}, \phi_{(sleep_1,execute_2)}, \ldots\ldots\ldots, \phi_{(execute_1,execute_2)}, \phi_{(execute_1,preempt_2)}\}$.
   (a) ${\rho_{\phi_{(sleep_1,sleep_2)}}}^A : {\pi_1}^A = sleep_1 \wedge {\pi_2}^A = sleep_2 \wedge 0 \leq t_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0$
   (b) ${\rho_{\phi_{(sleep_1,wait_2)}}}^A : {\pi_1}^A = sleep_1 \wedge {\pi_2}^A = wait_2 \wedge 0 \leq t_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0$
   ……
   ……
4. $\Theta^A : {\pi_1}^A = sleep_1 \wedge {\pi_2}^A = sleep_2 \wedge t_1 = 0 \wedge t_2 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge \dot{t_1} = 0 \wedge \dot{t_2} = 0 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0$
5. $\mathcal{T}^A = \{{\tau_{(sleep_1,sleep_2)(sleep_1,wait_2)}}^A, {\tau_{(sleep_1,wait_2)(sleep_1,execute_2)}}^A, \ldots\ldots\ldots\ldots, {\tau_{(wait_1,preempt_2)(execute_1,preempt_2)}}^A\}$.
   (a) ${\rho_{\tau_{(sleep_1,sleep_2)(sleep_1,wait_2)}}}^A : {\pi_1}^A = sleep_1 \wedge {\pi_2}^A = sleep_2 \wedge {\pi_1}^A{\prime} = sleep_1 \wedge {\pi_2}^A{\prime} = wait_2 \wedge 0 \leq t_1 \wedge t_2 = T_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0 \wedge 0 \leq t_1{\prime} \wedge t_2{\prime} = 0 \wedge 0 \leq x_1{\prime} \wedge x_2{\prime} = 0 \wedge \dot{t_1}{\prime} = 1 \wedge \dot{t_2}{\prime} = 1 \wedge \dot{x_1}{\prime} = 0 \wedge \dot{x_2}{\prime} = 0$

(b) ${\rho_{\tau_{(sleep_1,wait_2)(sleep_1,execute_2)}}}^{A}$: ${\pi_1}^{A} = sleep_1 \wedge {\pi_2}^{A} = wait_2 \wedge {\pi_1}^{A\prime} = sleep_1 \wedge {\pi_2}^{A\prime} = execute_2 \wedge 0 \leq t_1 \wedge t_2 \leq D_2 - E_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0 \wedge 0 \leq t_1\prime \wedge t_2\prime \leq D_2 - E_2 \wedge 0 \leq x_1\prime \wedge x_2\prime = 0 \wedge \dot{t_1\prime} = 1 \wedge \dot{t_2\prime} = 1 \wedge \dot{x_1\prime} = 0 \wedge \dot{x_2\prime} = 1$

. . . . . .

. . . . . .

3.3.2 Phase transition system of real-time software

First, we construct parallel composition of hybrid automata of periodic tasks and scheduler as shown in figure 5.

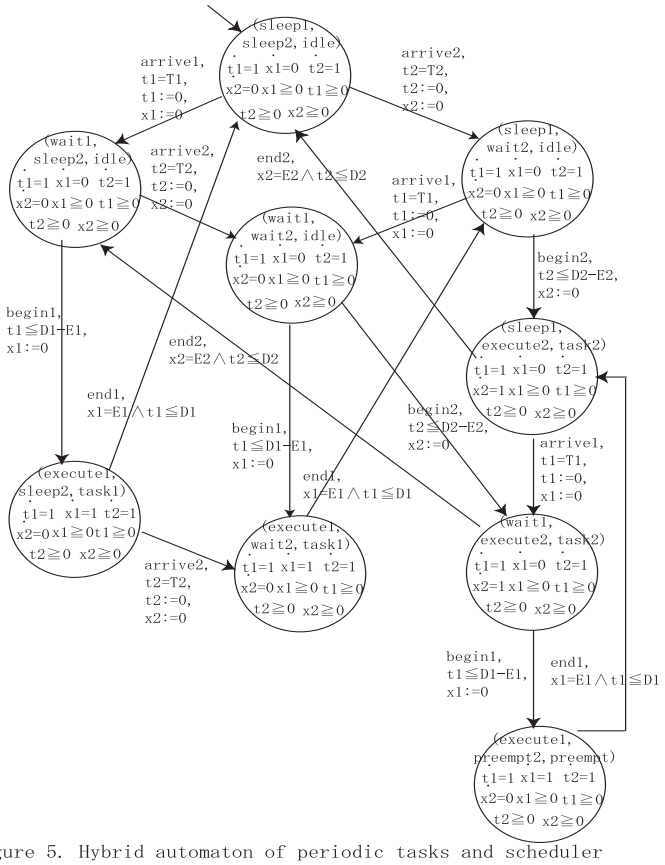

Figure 5. Hybrid automaton of periodic tasks and scheduler

Next, we construct phase transition system from parallel composition of hybrid automata. The phase transition system of periodic tasks and scheduler $M^C = (Var^C, L^C, \Phi^C, \Theta^C, \mathcal{T}^C)$ consists of five components as follows:

1. $Var^C = \{t_1, t_2, x_1, x_2, {\pi_1}^C, {\pi_2}^C, {\pi_3}^C\}$.

2. $L^C = \{{\pi_1}^C, {\pi_2}^C, {\pi_3}^C\}$.

3. $\Phi^C = \{\phi_{(sleep_1,sleep_2,idle)}, \phi_{(wait_1,sleep_2,idle)}, \ldots\ldots, \phi_{(sleep_1,execute_2,task_2)}\}$.

   (a) ${\rho_{\phi_{(sleep_1,sleep_2,idle)}}}^{C}$: ${\pi_1}^{C} = sleep_1 \wedge {\pi_2}^{C} = sleep_2 \wedge {\pi_3}^{C} = idle \wedge 0 \leq t_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0$

   (b) ${\rho_{\phi_{(wait_1,sleep_2,idle)}}}^{C}$: ${\pi_1}^{C} = wait_1 \wedge {\pi_2}^{C} = sleep_2 \wedge {\pi_3}^{C} = idle \wedge 0 \leq t_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0$

   . . . . . .

   . . . . . .

4. Θ^C : ${\pi_1}^C = sleep_1 \wedge {\pi_2}^C = sleep_2 \wedge {\pi_3}^C = idle \wedge t_1 = 0 \wedge t_2 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge \dot{t_1} = 0 \wedge \dot{t_2} = 0 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0$

5. $\mathcal{T}^C = \{{\tau_{(sleep_1,sleep_2,idle)(wait_1,sleep_2,idle)}}^{C}, {\tau_{(sleep_1,sleep_2,idle)(sleep_1,wait_2,idle)}}^{C}, \ldots\ldots\ldots\ldots, {\tau_{(execute_1,sleep_2,task_1)(execute_1,wait_2,task_1)}}^{C}\}$.

   (a) ${\rho_{\tau_{(sleep_1,sleep_2,idle)(wait_1,sleep_2,idle)}}}^{C}$: ${\pi_1}^{C} = sleep_1 \wedge {\pi_2}^{C} = sleep_2 \wedge {\pi_3}^{C} = idle \wedge {\pi_1}^{C\prime} = wait_1 \wedge {\pi_2}^{C\prime} = sleep_2 \wedge {\pi_3}^{C\prime} = idle \wedge t_1 = T_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0 \wedge \dot{t_1\prime} = 0 \wedge 0 \leq t_2\prime \wedge x_1\prime = 0 \wedge 0 \leq x_2\prime \wedge \dot{t_1\prime} = 1 \wedge \dot{t_2\prime} = 1 \wedge \dot{x_1\prime} = 0 \wedge \dot{x_2\prime} = 0$

   (b) ${\rho_{\tau_{(sleep_1,sleep_2,idle)(sleep_1,wait_2,idle)}}}^{C}$: ${\pi_1}^{C} = sleep_1 \wedge {\pi_2}^{C} = sleep_2 \wedge {\pi_3}^{C} = idle \wedge {\pi_1}^{C\prime} = sleep_1 \wedge {\pi_2}^{C\prime} = wait_2 \wedge {\pi_3}^{C\prime} = idle \wedge 0 \leq t_1 \wedge t_2 = T_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0 \wedge 0 \leq t_1\prime \wedge t_2\prime = 0 \wedge 0 \leq x_1\prime \wedge x_2\prime = 0 \wedge \dot{t_1\prime} = 1 \wedge \dot{t_2\prime} = 1 \wedge \dot{x_1\prime} = 0 \wedge \dot{x_2\prime} = 0$

   . . . . . .

   . . . . . .

3.3.3 Schedulability verification

Next, we verify whether $M^C = (Var^C, L^C, \Phi^C, \Theta^C, \mathcal{T}^C)$ refines $M^A = (Var^A, L^A, \Phi^A, \Theta^A, \mathcal{T}^A)$ or not.

Here we define α as follows:

$\alpha : ({\pi_1}^A, {\pi_2}^A) \rightarrow$

$$\begin{cases} if & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (sleep_1, sleep_2, idle) \\ then & (sleep_1, sleep_2) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (wait_1, sleep_2, idle) \\ then & (wait_1, sleep_2) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (sleep_1, wait_2, idle) \\ then & (sleep_1, wait_2, idle) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (wait_1, wait_2, idle) \\ then & (wait_1, wait_2) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (wait_1, execute_2, task_2) \\ then & (wait_1, execute_2) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (sleep_1, execute_2, task_2) \\ then & (sleep_1, execute_2) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (execute_1, sleep_2, task_1) \\ then & (execute_1, sleep_2) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (execute_1, wait_2, task_1) \\ then & (execute_1, wait_2) \\ elseif & ({\pi_1}^C, {\pi_2}^C, {\pi_3}^C) = (execute_1, preempt_2, preempt) \\ then & (execute_1, preempt_2) \end{cases}$$

1. $\Theta^C \rightarrow \Theta^A[\alpha]$:
   We can define Θ^C and Θ^A as follows:

   $\Theta^C : {\pi_1}^C = sleep_1 \wedge {\pi_2}^C = sleep_2 \wedge {\pi_3}^C = idle \wedge t_1 = 0 \wedge t_2 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge \dot{t_1} = 0 \wedge \dot{t_2} = 0 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0.$

   $\Theta^A : {\pi_1}^A = sleep_1 \wedge {\pi_2}^A = sleep_2 \wedge t_1 = 0 \wedge t_2 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge \dot{t_1} = 0 \wedge \dot{t_2} = 0 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0.$

   Moreover, we can define $\Theta^A[\alpha]$ as follows:

   $\Theta^A[\alpha] : {\pi_1}^C = sleep_1 \wedge {\pi_2}^C = sleep_2 \wedge {\pi_3}^C = idle \wedge t_1 = 0 \wedge t_2 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge \dot{t_1} = 0 \wedge \dot{t_2} = 0 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0.$

   We can conclude that $\Theta^C \rightarrow \Theta^A[\alpha]$ holds true.

2. For each ${\phi_i}^C \in \Phi^C$ and ${\tau_j}^C \in \mathcal{T}^C$, $(i = 1, \ldots, n, j = 1, \ldots, m)$! ′
   $\exists k.\ \rho_{{\phi_i}^C} = \rho_{{\phi_k}^A}[\alpha]$ and $\exists l.\ \rho_{{\tau_j}^C} \rightarrow \rho_{{\tau_l}^A}[\alpha]$

   (a)We can define $\rho_{\phi_{(sleep_1, sleep_2, idle)}}{}^C$ and $\rho_{\phi_{(sleep_1, sleep_2)}}{}^A$ as follows:

   $\rho_{\phi_{(sleep_1, sleep_2, idle)}}{}^C : {\pi_1}^C = sleep_1 \wedge {\pi_2}^C = sleep_2 \wedge {\pi_3}^C = idle \wedge 0 \leq t_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0.$

   $\rho_{\phi_{(sleep_1, sleep_2)}}{}^A : {\pi_1}^A = sleep_1 \wedge {\pi_2}^A = sleep_2 \wedge 0 \leq t_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0.$

   Moreover, we can define $\rho_{\phi_{(sleep_1, sleep_2)}}{}^A[\alpha]$ as follows:

   $\rho_{\phi_{(sleep_1, sleep_2)}}{}^A[\alpha] : {\pi_1}^C = sleep_1 \wedge {\pi_2}^C = sleep_2 \wedge {\pi_3}^C = idle \wedge \dot{x_2} = 0 \wedge 0 \leq t_1 \wedge 0 \leq t_2 \wedge 0 \leq x_1 \wedge 0 \leq x_2 \wedge \dot{t_1} = 1 \wedge \dot{t_2} = 1 \wedge \dot{x_1} = 0 \wedge \dot{x_2} = 0.$

   We can conclude that $\rho_{\phi_{(idle_1, idle_2, idle)}}{}^C = \rho_{\phi_{(idle_1, idle_2)}}{}^A[\alpha]$ holds true.

   . . . . . .

   . . . . . .

From (1) and (2), we can conclude that $M^C \sqsubseteq M^A$ holds true.

Next, we will verify whether, for $\forall i$, $Ri \leq Di (i = 1, .., n)$ holds true or not. We represents a periodic task i of period Ti, execution time Ei, and deadline $Di (i = 1, \ldots, n)$. The $(n+1)th$ worst-case response time $Ri^{(n+1)}$ for task i is as follows:

$$Ri^{(n+1)} = Ei + \sum_{j=1}^{i-1} \lceil \frac{Ri^{(n)}}{Tj} \rceil \times Ej$$

, where $Ri^{(0)}$=Ei and $Ri = \lim_{n \rightarrow \infty} Ri^{(n)}$.

By Joseph' method [10], we will calculate Ri, where $T1 = 300$, $E1 = 30$, $D1 = 250$, priority of task1=1, $T2 = 500$, $E1 = 100$, $D1 = 450$, priority of task2=2. Task1 has higher priority than task2.

We can verify whether $R1 \leq D1$ holds true or not. We could conclude that $R1 \leq D1$ holds true. We can conclude that $R2 \leq D2$ holds true,too. Because of lack of space, we omit the details.

We can conclude that M^C is schedulable.

4 Conclusion

In this paper, we proposed the schedulability verification method of real-time software by the integration of refinement and scheduling theory based on hybrid automata. We may model real-time software consisting of periodic tasks and a preemptive scheduling with fixed priorities. Using our proposed methods, we can uniformally and easily specify real-time software and verify its schedulability based on hybrid automata. Moreover, we can verify its schedulability at design stage. To the best our knowledge, ours is the first proposal to verify the schedulability of real-time software by the integration of refinement and scheduling theory based on hybrid automata.

References

[1] R. Alur, T.A. Henzinger, P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on SE*, 22(3), pp.181-201, 1996.

[2] S. Vestal. Modeling and verification of real-time software using extended linear hybrid automata. *5th NASA Workshop*, pp.95-106, 2000.

[3] V. Braberman, M. Felder. Verification of real-time designs. *LNCS 1687*, pp.494-510, 1999.

[4] K. Altisen, G. Goessler, J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Journal of RTS*, No.23, pp.55-84, 2002.

[5] Z. Liu, M. Joseph. Specification and verification of fault-tolerance, timing and scheduling, *ACM TOPLAS*, Vol.21, No.1, pp.46-89, 1999.

[6] J. Lehoczky, et-al. The rate monotonic scheduling algorithm: Exact characterization and average case behavior, *RTS*, pp.166-171, IEEE, 1989.

[7] S. Yamane. Refinement Theory of Embedded systems based on Hybrid models. *The 2002 IKE*, pp.455-461, CSREA Press, 2002

[8] O. Maler, Z. Manna, A. Pnueli. From timed to hybrid systems. *LNCS 600*, pp.447-484, 1992.

[9] M. Abadi, L. Lamport. The existence of refinement mappings. *TCS*, 82(2):253-284, 1991.

[10] M. Joseph. Real-Time Systems: Specification, Verification and Analysis. *Prentice Hall Intl.*, 1996.