数理解析研究所講究録 1205 巻 2001 年 160-165

Deriving Parameter Conditions for Periodic Timed Automata Satisfying Real-Time Temporal Logic Formulas

大阪大学 大学院基礎工学研究科 情報数理系　中田 明夫 (Akio Nakata)
Department of Informatics and Mathematical Science, Osaka University

1 Introduction

Model checking[1] have been recognized as one of very useful and effective methods for designing reliable hardware/software systems. Especially, in recent years, real-time systems must be developed for the areas that high reliability is required, such as aircraft/train/car controlling, nuclear reactors, medical devices and other real-time systems which may be produced a lot and hard to modify in later (e.g. hardware chips or embedded syste ms). Model checking techniques for real-time systems may be very useful for developing such a reliable real-time system to ensure that the system's design written in a some formal model satisfies the required properties such as safety, liveness, or fairness.

The classical model checking method is not *parametric*, that is, to check whether a behavioral specification written in some state model, satisfies some requirement specification (property) written in temporal logic, all the parameters in the specification must be fixed to some concrete values. [2] proposed a semi-decision procedure to derive a symbolic representation of states of a (parametric) hybrid automaton (an extension of a timed automaton) which satisfy a given temporal property written in a (parametric) real-time temporal logic formula. They adopt first-order theory with addition on real-numbers[3] as symbolic representation of state-sets. Although the satisfiability of the first-order theory with addition on real-numbers is decidable, fixpoint calculation is very costly and generally undecidable. On the other hand, [4] proposed an algorithm to obtain the condition of parameters in order that the given *non-parametric* state model on dense time domain satisfies the given parametric temporal logic formula. However, they only allow to write parameters to temporal logic formulas, not in a timed automaton. In a realistic system design process, we usually want to choose parameter values of models (implementations) rather than in temporal logic (specifications).

Thus, we propose a decision algorithm to derive a set of parameters of a subclass of a timed automaton model which may contain parameters (*parametric timed automata*[5]) and satisfies a formula of a real-time extension of CTL[1]. In our method, parameters are allowed in both a model and a temporal logic formula. We adopt formulas of first-order theory with addition on real-numbers[3] as symbolic representation of sets of parameter values. It is known to be decidable to check satisfiability and mechanical quantifier-elimination is also possible. In order to reduce the size of the intermediate symbolic representation, we take an *on-the-fly* approach, that is, we decompose the given problem on-the-fly to several subproblems, and recursively solve the subproblems to construct the entire condition for parameters. In this approach, we need not encode symbolically the entire state space (it tends to be very long). and only the necessary part of the tree is traversed. Specifically, we compute the weakest condition $WPC(s, f)$ of parameters in order that the state s of the given model satisfy the given temporal property f. First, we define $WPC(s, f)$ as a recursive function such as

$$WPC(s, f) \stackrel{\text{def}}{=} F(WPC(s_1, f_1), \cdots, WPC(s_k, f_k)),$$

where $F()$ is a functional on first-order formulas, each s_i is some *next* state of s, and each f_i is some derived formula of f. Basically, we can compute the weakest condition $WPC(s, f)$ if the application of the recursive definition of $WPC(s, f)$ is ensured to terminate. However, that is not the case in general. If the model contains some loops, such a recursive application does not terminate. To cope with the problem, we find some subclass of the timed automaton such that we need not to explore the infinite computation tree. When the model is a *periodic* timed automaton, that is, after some fixed time period it returns to its initial state and continues its behavior, we have only to check some finite part of the infinite computation tree and output the result. In our method, $WPC(s, f)$ is ensured to be obtained after a fixed steps of recursive computations, thus we can avoid costly fixpoint calculations of first order theory on real-numbers.

2 Periodic Timed Automata

In this section, we formally define our model, periodic timed automata. To begin with, we define a (non-periodic) parametric timed automaton model. Our definition of timed automata is essentially the same but slightly different from the traditional definition in [6], since we generally allow timing constraints to be first-order formulas with addition on real-numbers.

Let Act denote a set of actions, Var denote a set of variables, and $Pred(Var)$ denote a set of formulas of

first order theory with addition on real-numbers over Var. We also denote a set of real-numbers by $\mathbf{R}$ and a set of nonnegative real-numbers by $\mathbf{R}^+$

Definition 2.1 *A parametric timed automaton is a tuple* $\langle S, C, PVar, E, Inv(), s_{init}\rangle$, *where* S *is a finite set of control states,* $C \subseteq Var$ *is a finite set of* clocks, $PVar \subseteq Var$ *is a finite set of* parameters, $E \subseteq S \times Act \times Pred(Var) \times 2^C \times S$ *is a transition relation,* $Inv() : S \mapsto Pred(Var)$ *is an* invariant condition *for each state,* s_{init} *is the initial state. We write* $s_i \xrightarrow{a,P,r} s_j$ *if* $(s_i, a, P, r, s_j) \in E$. □

Informally, a transition $s_i \xrightarrow{a,P,r} s_j$ means that the action a can be executed from s_i when the values of both clocks and parameters satisfy the formula P (called a *guard condition*), and after executed, the state moves into s_j and clocks in the set r are reset to 0. In any state s, values of all clocks always increase continuously at the same speed, representing the time passage. Note that the values of clocks (and parameters if any) can never violate the invariant condition $Inv(s)$. Intuitively, $Inv(s)$ represents the range of values (e.g. minimum and maximum values) allowed for clocks (and parameters). Thus, time passage at state s will stop when the value of some clock will exceed the maximum value specified by $Inv(s)$. When time passage stops, some executable action is forced to execute, representing *urgency* of the action. Also, if $Inv(s')$ will be violated, any transition into the state s' is not allowed to execute.

Example 2.1 *Fig. 1 is a simple example of a parametric timed automaton. In Fig. 1, a set of parameters is* $\{x, y, z\}$, *a set of clocks is* $\{c, c'\}$, *the initial state is* s, *and the transition* $s[c \le x] \xrightarrow{a,[c\le x-2],\{\}} s_1[true]$ *means that in state* s, *for a given value of the parameter* x, *some time may be elapsed (i.e. the clock* c *increases) while* c *satisfies the invariant* $[c \le x]$, *and when* $c \le x - 2$ *holds, the action* a *can be executed, no clocks are reset to* 0, *and the state changes to* s_1 *(the invariant of* s_1 *is* $[true]$*). Similarly, the transition* $s[c \le x] \xrightarrow{b,[c>x-3],\{c'\}} s_2[c \le x]$ *means that some time may be elapsed while* c *satisfies the invariant* $[c \le x]$, *and when* $c > x - 3$ *holds, the action* b *can be executed, the clock* c' *is reset to* 0, *and the state changes to* s_2. *In the transition* $s_2[c \le x] \xrightarrow{d,[c\ge y \wedge c'\le z],\{\}} s_3[true]$, *a guard condition for both clocks* c *and* c' *are specified using parameters* y *and* z. □

Formal semantics of timed automata is defined as follows. The values of clocks and parameters are given by a function $\rho : (C \cup PVar) \mapsto \mathbf{R}$. We refer to such a function as a *value-assignment*. We represent a set of all value-assignments by Val. We write $\rho \models P$ if a formula $P \in Pred(Var)$ is true under a value-assignment $\rho \in Val$. The semantic behavior of a parametric timed automaton is given as a semantic transition system on *concrete states*. A concrete state is represented by (s, ρ), where s is a control state and ρ is a value-assignment.

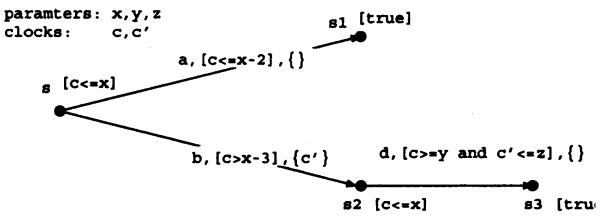

Fig. 1: Example of Parametric Timed Automata

Let $CS \stackrel{\text{def}}{=} \{(s, \rho) | s \in S, \rho \in Val\}$ be a set of concrete states. The semantic transition system consists of *delay-transitions* and *action-transitions*. A delay transition represents a time passage and an action transition represents an execution of an action. Formally, the semantic transition system is defined as follows.

Definition 2.2 *A* semantic transition system *for a parametric timed automaton* $\langle S, C, PVar, E, Inv(), s_{init}\rangle$ *is a labelled transition system on concrete states* CS, *where the transition relation is defined by the following rules:*

- $(s, \rho) \xrightarrow{t} (s, \rho + t)$ *if* $t \in \mathbf{R}^+$ *and* $(\rho + t) \models Inv(s)$,
- $(s, \rho) \xrightarrow{a} (s', \rho[r \to 0])$ *if* $s \xrightarrow{a,P,r} s'$, $\rho \models P$ *and* $\rho[r \to 0] \models Inv(s')$,

where $\rho + t$ *and* $\rho[r \to 0]$ *are the functions from variables to real-numbers, defined as follows:*

$$(\rho + t)(x) \stackrel{\text{def}}{=} \begin{cases} \rho(x) + t & \text{if } x \in C, \\ \rho(x) & \text{otherwise.} \end{cases}$$

$$(\rho[r \to 0])(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x \in r, \\ \rho(x) & \text{otherwise.} \end{cases}$$

□

The major difference of periodic timed automata from normal parametric timed automata is that it checks the elapsed time since it is started, and if it is equal to the specified period T, then it resets to its initial state. Moreover, it is assumed that only finitely bounded actions can be performed before returning to the initial state. To ensure the above properties, we define a periodic timed automaton as one obtained by adding reset transitions to a parametric timed automaton with no loops (we refer to such a parametric timed automaton as a *finite parametric timed automaton*). Formally it is defined as follows.

Definition 2.3 *A* finite parametric timed automaton *is a parametric timed automaton whose transition graph has no directed cycles, (i.e. it is a Directed Acyclic Graph(DAG)).* □

Example 2.2 *The parametric timed automaton in Example 2.1 is a finite parametric timed automaton since its transition graph is a tree (so it is also a DAG).* □

Definition 2.4 *A* periodic timed automaton *is a parametric timed automaton which is obtained by adding to*

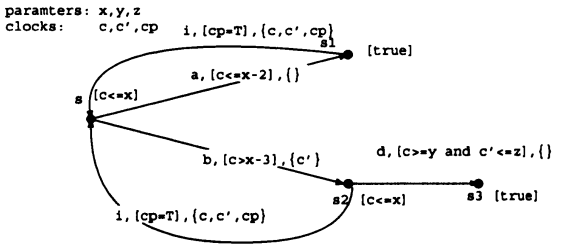

Fig. 2: Example of Periodic Timed Automata

$f ::= true$ (universally valid)
| $false$ (universally invalid)
| $\neg f$ (negation)
| $f \wedge f$ (conjunction)
| $f \vee f$ (disjunction)
| $f \Rightarrow f$ (implication)
| $\langle a \rangle_{\sim p} f$ (existential 'next' operator)
| $[a]_{\sim p} f$ (universal 'next' operator)
| $f E U_{\sim p} f$ (existential 'until' operator)
| $f A U_{\sim p} f$ (universal 'until' operator)
| $E G_{\sim p} f$ (existential 'always' operator)
| $A G_{\sim p} f$ (universal 'always' operator)
| $E F_{\sim p} f$ (existential 'eventually' operator)
| $A F_{\sim p} f$ (universal 'eventually' operator)

Fig. 3: Syntax of RPCTL

$(s,\rho) \models true.$

$(s,\rho) \models \neg f \stackrel{\text{def}}{=} (s,\rho) \not\models f.$

$(s,\rho) \models f_1 \wedge f_2 \stackrel{\text{def}}{=} (s,\rho) \models f_1$ and $(s,\rho) \models f_2$.

$(s,\rho) \models \langle a \rangle_{\sim p} f \stackrel{\text{def}}{=}$
there exists some transition sequence
$(s,\rho) \xrightarrow{t} (s,\rho+t) \xrightarrow{a} (s',\rho')$
such that $t \sim p$ and $(s',\rho') \models f$.

$(s,\rho) \models f_1 E U_{\sim p} f_2 \stackrel{\text{def}}{=}$
there exists some transition sequence
$(s,\rho) = (s_1,\rho_1) \xrightarrow{t_1} (s_1,\rho_1+t_1) \xrightarrow{a_1} \cdots$
$\xrightarrow{t_{k-1}} (s_{k-1},\rho_{k-1}+t_{k-1}) \xrightarrow{a_{k-1}} (s_k,\rho_k)$
and some non-negative real-number t_k,
s.t. $(s_k,\rho_k+t_k) \models f_2$ and $t_1 + \cdots + t_k \sim p$
and for any $i (1 \le i \le k)$ and for any $t_i' (0 \le t_i' < t_i)$,
$(s_i,\rho_i+t_i') \models f_1$

$(s,\rho) \models f_1 A U_{\sim p} f_2 \stackrel{\text{def}}{=}$
for any transition sequence such that
$(s,\rho) = (s_1,\rho_1) \xrightarrow{t_1} (s_1,\rho_1+t_1) \xrightarrow{a_1} \cdots$
$\xrightarrow{t_{k-1}} (s_{k-1},\rho_{k-1}+t_{k-1}) \xrightarrow{a_{k-1}} (s_k,\rho_k)$
and for any nonnegative real-number t_k,
$(s_k,\rho_k+t_k) \models f_2$ and $t_1 + \cdots + t_k \sim p$
and for any $i (1 \le i \le k)$ and for any $t_i' (0 \le t_i' < t_i)$,
$(s_i,\rho_i+t_i') \models f_1$.

Fig. 4: Semantics of RPCTL

a finite parametric timed automaton the following special reset transitions for every state s:

$$s \xrightarrow{i,[c_p=T],C} s_{init},$$

where s_{init} is a initial state, C is a set of all clocks, $c_p \in C$ is a a special clock which keeps the elapsed time from the initial state s_{init} (no other transition can reset this clock), $T \in \mathbf{R}^+$ is a period, *$i \in Act$ is a special reset action.* □

Example 2.3 *Fig 2 is an example of a periodic timed automaton. This example is a modified version of Example 2.2 where a special clock c_p and some return transitions such as $s_1 \xrightarrow{i,[c_p=T],\{c,c',c_p\}} s$ are added. Note that we allow periodic timed automata to terminate instead of returning to the initial state, such as the state s_3 in Fig 2.* □

3 Real-time CTL

In this section, we define RPCTL, a Real-time and Parametric extension of Computation Tree Logic(CTL)[1].

Definition 3.1 *The syntax of* RPCTL *formula is defined by the BNF in Fig. 3. where $a \in Act$ is an action name, p is a linear expression which may contain constant real-numbers and/or a parameter variable, and $\sim \in \{<, \le, >, \ge, =\}$ is a comparison operator. We may omit '$\sim p$' specifier, and in that case '≥ 0' is assumed.* □

RPCTL is a logic to specify a property of a parametric timed automaton state and its succeeding behavior using temporal operator with timing constraints which may contain parameters. Intuitive meaning of basic constructs of RPCTL is as follows. '*true*' is satisfied by all concrete states. '$\neg f$' is satisfied by a concrete state (s,ρ) if and only if f is not satisfied by (s,ρ). '$false$' is never satisfied by any concrete states, which is equivalent to $\neg true$. '$f_1 \wedge f_2$' is satisfied if and only if both f_1 and f_2 is satisfied. '$f_1 \vee f_2$' and '$f_1 \Rightarrow f_2$' are defined similarly to classic propositional logic. '$\langle a \rangle_{\le p} f$' is satisfied by (s,ρ) if and only if there exists *some* transition from (s,ρ) performing a within c units of time, such that f is satisfied by the next (control) state. Since we can define similarly if $\sim$ is other than $<$ (case of $\ge, <, >, =$), we only mention the case of $\le$ in the following explanation. '$[a]_{\le p} f$' is satisfied if and only if for *any* transition from the state performing a within c units of time, f is satisfied by the next state, which is the same as $\neg\langle a \rangle_{\le p} \neg f$. '$f_1 E U_{\le p} f_2$' is satisfied if and only if there exists *some* transition sequence such that f_2 eventually holds within c units of time and until then, f_1 is always satisfied. '$f_1 A U_{\le p} f_2$' satisfied if and only if for *any* transition sequence, f_2 eventually holds within c units of time and until then, f_1 is always satisfied. '$E F_{\le p} f$',

'$AG_{\leq p}f$', '$AF_{\leq p}f$' and '$EG_{\leq p}f$' are abbreviations for $trueEU_{\leq p}f$, $\neg EF_{\leq p}\neg f$, $trueAU_{\leq p}f$ and $\neg AF_{\leq p}\neg f$, respectively.

In general, we write $M,(s,\rho) \models f$ to mean that an RPCTL formula f is satisfied by a concrete state (s,ρ) of a parametric timed automaton M. If there are no confusions, we omit M and just write $(s,\rho) \models f$. The formal definition of the relation $\models$ is as follows. We only give the definitions for six primitive constructs, *true*, $\neg f$, $f_1 \wedge f_2$, $\langle a\rangle_{\sim p}f$, $f_1 EU_{\sim p} f_2$ and, $f_1 AU_{\sim p} f_2$. Rest of the constructs can be rewritten by using the above constructs.

Definition 3.2 *The relation $(s,\rho) \models f$ is formally defined in Fig. 4.* □

Example 3.1 *The RPCTL formula*

$$[a_1]_{<q_1}(((\langle a_2\rangle true)EU_{\geq q_2}(\langle a_3\rangle true))$$

means that for every state reachable after executed the action a_1 within q_1 units of time, there exists a execution path such that the action a_2 is always executable until a_3 becomes executable after q_2 units of time elapsed. Note that q_1 and q_2 are parameter variables. □

4 Deriving of the Weakest Condition of Parameters

Now we describe our method to derive symbolically the weakest condition of parameters $WPC(s,f)$ in order that the state s of the periodic timed automaton satisfies the RPCTL property f. Beforehand, we give the precise definition of our problem.

Definition 4.1 *Let M be an parametric timed automata, s be a state of M, and f be an RPCTL formula. The* parameter condition derivation problem *is a problem to derive a first-order formula $WPC(s,f)$ such that*

$$\rho \models WPC(s,f) \text{ iff } (s,\rho) \models f.$$ □

At first, we give an algorithm to solve the parameter condition problem for finite parametric timed automata, and then we extend it to periodic timed automata.

4.1 Finite Model Case

As mentioned in Section 1, we define $WPC(s,f)$ as a recursive function such that

$$WPC(s,f) \stackrel{\text{def}}{=} F(WPC(s_1,f_1),\ldots,WPC(s_k,f_k)).$$

Here we give a concrete definition of the function $WPC(s,f)$ for every construct of RPCTL formula f.

Definition 4.2 *The function $WPC(s,f)$, which takes as arguments a state s of a parametric timed automaton, and a RPCTL formula f, and returns a first order formula, is defined as Fig 5, where $P[C+t/C]$ ($P[0/r]$) represents a first order formula P whose every free occurrence of each variable $x \in C$ ($x \in r$) is replaced with $x+t$ (0, respectively).* □

$$WPC(s,true) \stackrel{\text{def}}{=} true$$

$$WPC(s,\neg f) \stackrel{\text{def}}{=} \neg WPC(s,f)$$

$$WPC(s,f_1 \wedge f_2) \stackrel{\text{def}}{=} WPC(s,f_1) \wedge WPC(s,f_2)$$

$$WPC(s,\langle a\rangle_{\sim p}f) \stackrel{\text{def}}{=} \exists t_s(0 \leq t_s \wedge t_s \sim p \wedge (Inv(s) \wedge \bigvee_{i\in I(s,a)} \{P_i \wedge (Inv(s_i) \wedge WPC(s_i,f'))[0/r]\})[C+t_s/C])$$

where $I(s,a) = \{i | s \xrightarrow{a,[P_i],r_i} s_i\}$,

$$WPC(s,f_1 EU_{\sim p} f_2) \stackrel{\text{def}}{=} \exists t_s(0 \leq t_s \wedge \forall t'_s((0 \leq t'_s \wedge t'_s \leq t_s) \Rightarrow WPC(s,f_1)[C+t'_s/C]) \wedge (Inv(s) \wedge (t_s \sim p \wedge WPC(s,f_2)) \vee \bigvee_{i\in I(s)} \{P_i \wedge WPC(s_i, f_1 EU_{\sim(c-t_s)} f_2)\}))[C+t_s/C])$$

where $I(s) = \{i | s \xrightarrow{a_i,[P_i],r_i} s_i\}$,

$$WPC(s,f_1 AU_{\sim p} f_2) \stackrel{\text{def}}{=} \forall t_s(0 \leq t_s \Rightarrow \forall t'_s((0 \leq t'_s \wedge t'_s \leq t_s) \Rightarrow WPC(s,f_1)[C+t'_s/C]) \wedge (Inv(s) \Rightarrow (t_s \sim p \wedge WPC(s,f_2)) \wedge \bigwedge_{i\in I(s)} \{P_i \Rightarrow WPC(s_i, f_1 AU_{\sim(c-t_s)} f_2)\}))[C+t_s/C])$$

Fig. 5: Function $WPC(s,f)$

Explanation of the definition of $WPC(s,f)$ is as follows. If f is one of *true* or $f_1 \wedge f_2$, the definition of $WPC(s,f)$ is straightforward. The case of $f = \neg f'$ is less obvious, but since we have defined $WPC(s,f)$ as the weakest condition, $\rho \not\models WPC(s,f)$ immediately implies $\rho \models WPC(s,\neg f)$, and vice versa. Hence we have $WPC(s,\neg f) = \neg WPC(s,f)$.

Consider the case of $f = \langle a\rangle_{\sim p}f'$. Suppose ρ is a value-assignment such that $(s,\rho) \models \langle a\rangle_{\sim p}f'$. From the Definition 3.2, there must exist a concrete transition sequence $(s,\rho) \xrightarrow{t} (s,\rho+t) \xrightarrow{a} (s',\rho')$ such that $t' \sim p$ and $(s',\rho') \models f'$. Thus, the following conditions must also hold:

- some timed automaton transition $s \xrightarrow{a,P,r} s'$ must exists.
- $\rho+t$ must satisfy both $Inv(s)$ and P (Definition 2.2).
- ρ' is a value-assignment $\rho+t$ whose values of the clocks in r are reset to zero, i.e. $\rho' = (\rho+t)[r \rightarrow 0]$, and it satisfies $Inv(s')$.
- $(s',\rho') \models f'$, i.e. $\rho' \models WPC(s',f')$.

Hence, we obtain a necessary condition

> "there exists some non-negative real-number t and some transition $s \xrightarrow{a,P,r} s'$, such that $\rho \models (t \sim p)$, $\rho+t \models \wedge Inv(s) \wedge P$ and $(\rho+t)[r \rightarrow 0] \models Inv(s') \wedge WPC'(s',f')$"

for ρ to make the state s satisfy f. We can rewrite '$\rho+t \models Inv(s) \wedge P$' to the condition of ρ, such as $\rho \models (Inv(s) \wedge P)[C+t/C]$. By the same way, we can also rewrite $(\rho+t)[r \rightarrow 0] \models Inv(s') \wedge WPC'(s',f')$ as $\rho \models (Inv(s') \wedge WPC(s',f'))[C+t/C, 0/r]$. Therefore,

the following condition holds:

$$\rho \models (0 \le t \wedge t \sim p \wedge Inv(s) \wedge P \wedge (Inv(s') \wedge WPC(s', f'))[0/r])[C + t/C].$$

Since it is sufficient that some non-negative real-number t and some transition $s \xrightarrow{a,P,r} s'$ exist, we can weaken the above condition as:

$$\rho \models \exists t(0 \le t \wedge t \sim p \wedge Inv(s) \wedge \bigvee_{i \in I(s,a)} \{P_i \wedge (Inv(s_i) \wedge WPC(s_i, f'))[0/r]\})[C + t/C].$$

where $I(s,a) \stackrel{def}{=} \{i | s \xrightarrow{a,P_i,r_i} s_i\}$ is a set of indices of transitions whose source node is s and action name is a. We can easily prove that this is the weakest condition of ρ such that $(s,\rho) \models f$, and t is some flesh variable which does not appear in either $Inv(s)$, P_i, $Inv(s_i)$ or $WPC(s_i, f')$.

Consider the case of $f = f_1 EU_{\sim p} f_2$. Similar to above, suppose ρ is a value-assignment such that $(s,\rho) \models f_1 EU_{\sim p} f_2$. From the Definition 3.2, there must exist some transition sequence $(s,\rho) = (s_1,\rho_1) \xrightarrow{t_1} (s_1,\rho_1+t_1) \xrightarrow{a_1} \cdots \xrightarrow{t_{k-1}} (s_{k-1},\rho_{k-1}+t_{k-1}) \xrightarrow{a_{k-1}} (s_k,\rho_k) \xrightarrow{t_k} (s_k,\rho_k+t_k)$, such that $(s_k,\rho_k+t_k) \models f_2$, $t_1 + \cdots + t_k \sim p$, and for any j $(1 \le j \le k)$ and for any t'_j $(0 \le t'_j < t_j)$, $(s_j,\rho_j + t'_j) \models f_1$ holds. To obtain a recursive definition of $WPC(s,f)$, we divide the premise of the above statement into two cases, $k = 1$ and $k \ge 2$.

[Case $k = 1$]: If $k = 1$, then there must exist a transition sequence $(s,\rho) \xrightarrow{t} (s,\rho+t)$ such that $(s,\rho+t) \models f_2$, $t \sim p$ and for any t' $(0 \le t' < t)$, $(s,\rho + t) \models f_1$ holds. Similar to the case of $f = \langle a \rangle_{\sim p} f'$, the weakest condition of ρ is obtained as follows:

$$\rho \models \exists t(0 \le t \wedge t \sim p \wedge (Inv(s) \wedge WPC(s, f_2))[C + t/C] \wedge \forall t'((0 \le t' \wedge t' \le t) \Rightarrow WPC(s, f_1)[C + t'/C])) \quad (1)$$

[Case $k \ge 2$]: If we assume $k \le 2$, then there must exist a transition sequence $(s,\rho) = (s_1,\rho_1) \xrightarrow{t_1} (s_1,\rho_1 + t_1) \xrightarrow{a_1} (s_2,\rho_2)$ such that $(s_2,\rho_2) \models f_1 EU_{\sim(c-t_1)} f_2$ holds, and for any t'_1 $(0 \le t'_1 \le t_1)$, $(s_1,\rho_1 + t'_1) \models f_1$ holds. Considering that there may exist multiple transitions $s \xrightarrow{a_i,P_i,r_i} s_i$, the weakest condition of ρ is obtained as follows:

$$\rho \models \exists t[0 \le t \wedge (Inv(s) \wedge \bigvee_{i \in I(s)} \{P_i \wedge WPC(s_i, f_1 EU_{\sim(c-t)} f_2)\})[C + t/C] \wedge \forall t'[(0 \le t' \wedge t' \le t) \Rightarrow WPC(s, f_1)[C + t'/C]]] \quad (2)$$

where $I(s) \stackrel{def}{=} \{i | s \xrightarrow{a_i,P_i,r_i} s_i\}$ is a set of indices of transitions whose source node is s.

Therefore, the general case is (1) or (2), that is,

$$\rho \models \exists t[0 \le t \wedge \forall t'[(0 \le t' \wedge t' \le t) \Rightarrow WPC(s, f_1)[C + t'/C]]] \wedge (Inv(s) \wedge (t \sim p \wedge WPC(s, f_2)) \vee \bigvee_{i \in I(s)} \{P_i \wedge WPC(s_i, f_1 EU_{\sim(c-t)} f_2)\}))[C + t/C]$$

The case of $f = f_1 AU_{\sim p} f_2$ is similar, and we omit the detailed description due to space limitation.

If the transition graph contains no loops, there are no cases that $WPC(s,f)$ is recursively called during the computation of $WPC(s,f)$ itself. Thus, the function call $WPC(s,f)$ is ensured to terminate. Hence, a recursive function $WPC(s,f)$ is an algorithm to obtain the parameter condition for DAG-formed models (i.e. finite parametric timed automata).

Theorem 4.1 *For every state s of a finite parametric timed automaton M and every RPCTL formula f, a recursive function $WPC(s,f)$ always terminates and returns a correct solution of a parameter condition derivation problem, i.e.:*

$$\forall \rho.[\rho \models WPC(s,f) \text{ iff } (s,\rho) \models f]. \quad \square$$

4.2 Periodic Model Case

If parametric timed automata have some loops, the algorithm $WPC(s,f)$ in Theorem 4.1 may not terminate. In this section, we prove that if models are periodic timed automata, we have only to check a finite fragment of the computation tree to derive the weakest condition of parameters.

At first, we introduce the notion of unfolding. Replace all returning transitions of a periodic timed automaton (Fig. 6-(a)) with transitions to special terminating states. We obtain the corresponding finite parametric timed automata (Fig. 6-(b)). Then, attach the copies of the corresponding finite parametric timed automaton to each special terminating state of itself, once for all. Finally, we have the timed automaton which represents the first 2 period behavior (Fig. 6-(c)). We refer to such a model as an *unfolding* of the periodic timed automaton. Similarly, we can also define a k-unfolding of a periodic timed automaton as the finite parametric timed automaton which represents the first k period behavior.

Our result is that without loss of generality, we have only to check 3-unfolding of periodic timed automata for each subformula including 'until' operators EU and AU, to derive the paths whose execution times is within 3 periods,

Formally, it is proved by the following lemma:

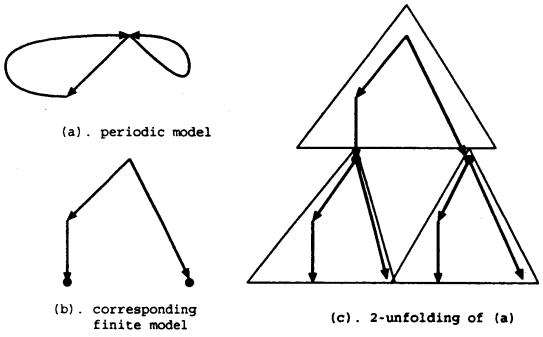

Fig. 6: Unfolding of Periodic Timed Automata

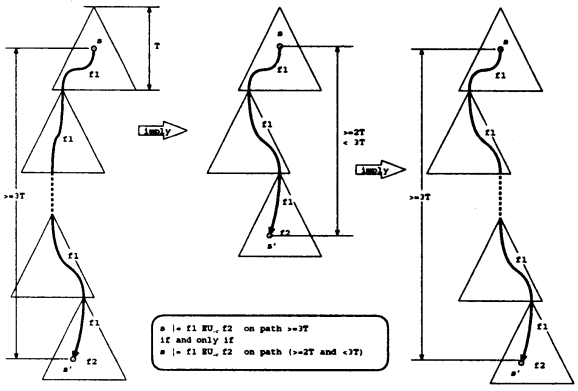

Fig. 7: Illustration of Lemma 4.1

Lemma 4.1 *For any concrete state (s,ρ) of periodic timed automata, the following condition holds:*

$$(s,\rho) \models f_1 EU_{\sim p} f_2 \Longleftrightarrow (s,\rho) \models f_1 EU_{\sim p'} f_2$$

where T is the period of the periodic timed automata, $p' \stackrel{\text{def}}{=} p - m \times T$ and m is the minimum nonnegative integer s.t. $p - m \times T < 3T$. The same condition also holds for $f_1 AU_{\sim p} f_2$.

(proof) Detailed proof is omitted due to space limitation, but we can prove that if there exists a execution path α which satisfies the 'until' property and whose execution time $ET(\alpha)$ is greater than 3 periods, there also exists a path α' whose execution time is less than 3 periods, and satisfy the same property, and vice versa (illustrated in Fig. 7). □

We define another algorithm $WPC3(s,f)$ instead of $WPC(s,f)$ for periodic timed automata. $WPC3(s,f)$ is almost the same as $WPC(s,f)$, except that $WPC3(s, f_1 op_{\sim p} f_2)$ (op is one of 'until' operators) is applied to 3-*unfolding* of the given periodic timed automaton. Specifically, for each subformula $f_1 op_{\sim p} f_2$, $WPC3(s, f_1 op_{\sim p} f_2)$ traverses the 3-unfolding where s is assumed to be the state of the first period. Unlike $WPC(s,f)$, $WPC3(s,f)$ is ensured to terminate. By Lemma 4.1, it is sufficient to consider finite paths whose execution time is at most $3T$ in order to derive the weakest condition of parameters. Therefore, we obtain the following main theorem:

Theorem 4.2 *For every state s of a periodic timed automaton M and every RPCTL formula f, a recursive function $WPC3(s,f)$ always terminates and returns a correct solution of a parameter condition derivation problem, i.e.:*

$$\forall \rho.[\rho \models WPC3(s,f) \textit{ iff } (s,\rho) \models f] \qquad \square$$

5 Concluding Remarks

In this paper, we propose a method to derive a parameter condition for a periodic timed automaton which satisfies a property written in a temporal logic RPCTL formula.

Although our method applies to only restricted class of timed automata, many real-time applications such as audio/video streaming, time sharing task schedulers, etc. can be specified as a periodic timed automaton. Therefore, our method may be useful for system designers choose correct parameters to guarantee the specified correctness properties of such periodic systems.

The future works are to extend our method to handle some internal variables, and to apply the method to a practical application to evaluate the efficiency.

References

[1] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite state concurrent systems using temporal logic specifications," *ACM Trans. on Program Languages and Semantics*, vol. 8, no. 2, pp. 244–263, 1986.

[2] R. Alur, T. A. Henzinger, and P. Ho, "Automatic symbolic verification of embedded systems," *IEEE Transactions on Software Engineering*, vol. 22, Mar. 1996.

[3] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[4] F. Wang, "Parametric timing analysis for real-time systems," *Information and Computation*, vol. 130, pp. 131–150, 1 Nov. 1996.

[5] R. Alur and T. A. Henzinger, "Parametric real-time reasoning," in *Proc. 25th ACM Annual Symp. on the Theory of Computing (STOC'93)*, pp. 592–601, 1993.

[6] R. Alur and D. Dill, "Automata for modelling real-time systems," in *Proc. of ICALP'90* (M. S. Paterson, ed.), vol. 443 of *Lecture Notes in Computer Science*, pp. 322–335, Springer-Verlag, 1990.