DERIVING PARAMETER CONDITIONS FOR PERIODIC TIMED AUTOMATA SATISFYING REAL-TIME TEMPORAL LOGIC FORMULAS*

Akio Nakata and Teruo Higashino
Dept. of Informatics and Mathematical Science,Osaka University
Toyonaka, Osaka 560-8531, Japan
{nakata,higashino}@ics.es.osaka-u.ac.jp

Abstract A symbolic model checking method for parametric periodic timed automata is proposed. The method derives symbolically the weakest condition for parameters such that the specified control state of a periodic timed automaton satisfies some temporal properties. Unlike several existing parametric symbolic model checking methods, the proposed method is 'on-the-fly' — it does not unnecessarily check all the states. Instead, it traverses some necessary part of the computation tree to derive the weakest condition. We show that if we constrain a timed automaton to be *periodic*, i.e. if we force a timed automaton to return to its initial state periodically at the specified constant time, we have only to traverse at most the first 3 periods in the infinite computation tree. In the proposed method, we can avoid a costly (and generally undecidable) fixpoint-calculation for dense-time-domain state sets, and derive the weakest condition for parameters of a timed automaton to satisfy given temporal properties written in a real-time temporal logic formula.

Keywords: symbolic model checking, real-time periodic system, temporal logic

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 have been 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

*This work is partially supported by Semiconductor Technology Academic Research Center (STARC), Japan.

hard to modify in later (e.g. hardware chips or embedded systems). Model checking techniques may be very useful for developing such reliable real-time systems to ensure that the system's design written in some formal model satisfies the required properties such as safety, liveness, and 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. In recent years, several *symbolic* model checking methods are proposed [2, 3, 4, 5]. Symbolic model checking is a method to construct a set of states (state-sets) which satisfy given temporal properties by representing infinite or finitely huge state-sets symbolically and using a symbolic calculation. The crucial part of symbolic calculation is a fixpoint calculation. There are some symbolic representations of state-sets, such as BDDs[2], which enable us to compute fixpoints efficiently. Although BDDs themselves are aimed for compression of finitely-huge state space, if we adopt some symbolic representations such as Presburger Arithmetic[6], we can extend the method to possibly infinite state space. Such an approach is especially useful for *parametric analyses*, that is, we can obtain the representation of the set parameter variables in order that the behavioral specification satisfies the requirement specification, instead of repeatedly guessing concrete parameter values and checking satisfiability.

More recently, [3] proposed a semi-decision procedure to derive a symbolic representation of parametric states of a hybrid automaton (an extension of a timed automaton) in order to reach some given state-sets. They adopt first-order theory with addition on real-numbers[6] 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. [4] proposed some approximation techniques to cope with such undecidability of fixpoint calculations using Presburger Arithmetic as symbolic representation of state-sets, but the domain of the variables is restricted to integers. On the other hand, [5] 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*[7]) 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 as symbolic representation of sets of parameter values, as similar to [3]. In order to reduce the size of the intermediate symbolic representation, we decompose the given problem on-the-fly to several subproblems, and recursively solve the subproblems to construct the entire condition for parameters. Unlike [3], 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 state s of the given model should 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 either s or some *next* state of s, and each f_i is either a *proper* subformula of f or some *derived* formula of f (not necessarily a subformula 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 automata 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 repeats its behavior periodically, we have only to check some finite part of the infinite computation tree and can 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. Moreover, in our on-the-fly approach, intermediate results need to compute $WPC(s, f)$ are kept small compared to the state-space construction approaches. Using dynamic programming, we can also avoid duplicate computation of $WPC(s_i, f_i)$'s when the pair of s_i and f_i is the same. Moreover, each subcondition $WPC(s_i, f_i)$ can be computed in parallel, so we can easily parallelize our method using parallel processors to improve the efficiency.

The rest of this paper is organized as follows. In Section 2, we introduce our model, periodic timed automata. In Section 3, we give a definition of the logic, real-time CTL. In Section 4, we explain our method to obtain $WPC(s, f)$ in detail. Section 6 concludes this paper.

Related Works Several parametric model checking method have been proposed for real-time models[7, 8, 9] and value-passing I/O models[10]. For given two parametric state models, Refs.[7, 8, 10] have proposed the methods to derive the parameter conditions to make one model a correct implementation of another model. [7] has adopted language inclusion as an implementation relation, while [8, 10] has adopted bisimulation equivalence. [8] is an extension

of [10] to a timed model. The proposed method in [9] takes a non-parametric state model on discrete time domain and a temporal logic formula which may have some parameters bounded by quantifiers, and checks whether the model satisfies the logic formula. Although some very interesting properties may be expressed using parameters, it simply checks whether or not the given model satisfies the given property. In addition, they consider that the time domain is integers.

2. Parametric and Periodic Timed Automata

In this section, we formally define our model, periodic timed automata. In addition to the traditional theory of Timed Automata[11], we introduce parameters on any (discrete or dense) domain. Then, we define a (non-periodic) parametric timed automaton model. Our definition of timed automata is essentially the same but slightly different from the parametric timed automata in [7], since we generally allow timing constraints to be first-order formulas with addition on real-numbers.

Let Act, Var, $Pred(Var)$ denote the set of all actions, the set of all variables, and the set of all formulas of first order theory with addition on real-numbers over Var, respectively. We also denote the set of real-numbers by $\mathbf{R}$ and the set of non-negative 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 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 zero. In any state s, values of all clocks 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*[12] of the action. Also, any incoming transition of the state s' violating $Inv(s')$ is not allowed.

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*

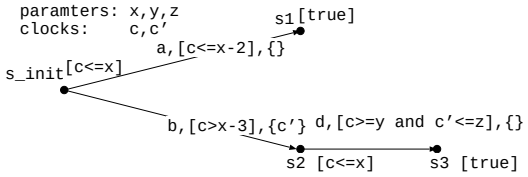

Figure 1. Example of Parametric Timed (DAG-)Automata

is s, and the transition $s[c \leq x] \overset{a,[c \leq x-2],\{\}}{\longrightarrow} s_1[true]$ *means that at 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* $[c \leq x]$*, and when* $c \leq x - 2$ *holds, action a can be executed, no clocks are reset to zero, and the state changes to* s_1 *(the invariant of* s_1 *is* $[true]$*, which means that any clock and parameter values are allowed at* s_1*). Similarly, the transition* $s[c \leq x] \overset{b,[c>x-3],\{c'\}}{\longrightarrow} s_2[c \leq x]$ *means that some time may be elapsed while c satisfies the invariant* $[c \leq x]$*, and when* $c > x - 3$ *holds, action b can be executed, clock* c' *is reset to zero, and the state changes to* s_2*. In the transition* $s_2[c \leq x] \overset{d,[c \geq y \wedge c' \leq z],\{\}}{\longrightarrow} s_3[true]$*, 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 $\sigma : (C \cup PVar) \mapsto \mathbf{R}$. We refer to such a function as a *value-assignment*. We represent a set of all value-assignments by Val. We write $\sigma \models P$ if a formula $P \in Pred(Var)$ is true under a value-assignment $\sigma \in Val$. 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, σ), where s is a control state and σ is a value-assignment. Let $CS \overset{\text{def}}{=} \{(s, \sigma) | s \in S, \sigma \in Val\}$ be a set of concrete states. The semantic transition system consists of *delay-transitions* and *action-transitions*. A delay transition represents a time passage within the same control state $s \in S$, whereas an action transition represents an execution of an action which changes the control state to the next one s'. Formally, the semantic transition system is defined as follows.

Definition 2.2 *For any value-assignment* σ*,* $t \in \mathbf{R}^+$*, and* $r \subseteq C$*, let* $\sigma + t$ *and* $\sigma[r \to 0]$ *be the value-assignments such that*

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

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

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:*

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

□

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 *parametric timed DAG-automaton*). Formally it is defined as follows.

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

The parametric timed automaton in Example 2.1 is a parametric timed DAG-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 from a parametric timed DAG-automaton by adding the special reset transition (called a* return transition*)* $s \xrightarrow{i,[c_p=T],C} s_{init}$ *for state* s*, where* s_{init} *is the initial state,* C *is a set of all clocks,* $c_p \in C$ *is 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 2.2 *Fig. 2 is an example of a periodic timed automaton. This example is a modified version of Example 2.1 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] including some operators in ACTL[13][1].

[1] Unlike [1] and many timed extensions of CTL such as TCTL[3], the next operator of RPCTL is attributed by an action name (similar to Hennessy-Milner Logic[14], or ACTL[13]) so that we can verify the properties

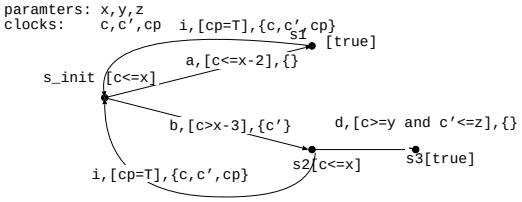

Figure 2. Example of Periodic Timed Automata

f	::=	$true$ (universally valid)		
	|	$false$ (universally invalid)	| $fEU_{\sim p}f$	(existential 'until' operator)
	|	$\neg f$ (negation)	| $fAU_{\sim p}f$	(universal 'until' operator)
	|	$f \wedge f$ (conjunction)	| $EF_{\sim p}f$	(existential 'eventually' operator)
	|	$f \vee f$ (disjunction)	| $AF_{\sim p}f$	(universal 'eventually' operator)
	|	$f \Rightarrow f$ (implication)	| $EG_{\sim p}f$	(existential 'always' operator)
	|	$\langle a \rangle_{\sim p} f$ (existential 'next' operator)	| $AG_{\sim p}f$	(universal 'always' operator)
	|	$[a]_{\sim p} f$ (universal 'next' operator)		

Figure 3. Syntax of RPCTL

In compared to TCTL[3], we do not adopt the *freeze quantifer* as a primitive operator of RPCTL.

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 parameter variables, and $\sim \in \{<, \leq, >, \geq, =\}$ is a comparison operator. We may omit '$\sim p$' specifier, and in that case '≥ 0' is assumed.* □

RPCTL is a logic to specify a temporal property at the state of of a parametric timed automaton for its succeeding behavior using temporal operator with timing constraints which may contain parameters. Intuitive meaning of basic constructs of RPCTL is as follows. '*true*' holds at any concrete state. '$\neg f$' holds at a concrete state (s, σ) if and only if f does not hold at (s, σ). '$false$' never holds at any concrete state, which is equivalent to $\neg true$. '$f_1 \wedge f_2$' holds if and only if both f_1 and f_2 hold. '$f_1 \vee f_2$' and '$f_1 \Rightarrow f_2$' are also defined similarly to classic propositional logic. '$\langle a \rangle_{\leq p} f$' holds at (s, σ) if and only if there exists *some* transition from (s, σ) performing a within p units of time, such that f holds at the next (control) state. Since we can define similarly if

when we view the model as a mealy-machine, or (timed-extension of) process algebra with observation semantics (i.e. processes are identified by observing actions, not states) such as CCS, CSP or LOTOS.

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

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

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

$(s,\sigma) \models \langle a \rangle_{\sim p} f \stackrel{\text{def}}{=}$

there exists some transition sequence $(s,\sigma) \xrightarrow{t} (s,\sigma+t) \xrightarrow{a} (s',\sigma')$
such that $t \sim p$ and $(s',\sigma') \models f.$

$(s,\sigma) \models f_1 E U_{\sim p} f_2 \stackrel{\text{def}}{=}$

there exists some transition sequence
$(s,\sigma) = (s_1,\sigma_1) \xrightarrow{t_1} (s_1,\sigma_1+t_1) \xrightarrow{a_1} \cdots \xrightarrow{t_{k-1}} (s_{k-1},\sigma_{k-1}+t_{k-1}) \xrightarrow{a_{k-1}} (s_k,\sigma_k)$
and some non-negative real-number t_k, s.t. $(s_k,\sigma_k+t_k) \models f_2$ and $t_1+\cdots+t_k \sim p$
and for any $i(1 \le i \le k)$and for any $t_i'(0 \le t_i' < t_i)$, $(s_i,\sigma_i+t_i') \models f_1$

$(s,\sigma) \models f_1 A U_{\sim p} f_2 \stackrel{\text{def}}{=}$

for any transition sequence such that
$(s,\sigma) = (s_1,\sigma_1) \xrightarrow{t_1} (s_1,\sigma_1+t_1) \xrightarrow{a_1} \cdots \xrightarrow{t_{k-1}} (s_{k-1},\sigma_{k-1}+t_{k-1}) \xrightarrow{a_{k-1}} (s_k,\sigma_k)$
and for any nonnegative real-number $t_k, (s_k,\sigma_k+t_k) \models f_2$ and $t_1+\cdots+t_k \sim p$
and for any $i(1 \le i \le k)$ and for any $t_i'(0 \le t_i' < t_i)$, $(s_i,\sigma_i+t_i') \models f_1.$

Figure 4. Semantics of RPCTL

$\sim$ is other than $\le$ (case of $\ge, <, >, =$), we only mention the case of $\le$ in the following explanation. '$[a]_{\le p} f$' holds if and only if for *any* transition from the state performing a within p units of time, f holds at the next state, which is the same as $\neg\langle a\rangle_{\le p}\neg f$. '$f_1 E U_{\le p} f_2$' holds 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 always holds. '$f_1 A U_{\le p} f_2$' holds if and only if for *any* transition sequence, f_2 eventually holds within p units of time and until then, f_1 always holds. We can also use '$EF_{\le p} f$', '$AG_{\le p} f$', '$AF_{\le p} f$' and '$EG_{\le p} f$' as abbreviations for $true\ EU_{\le p} f$, $\neg EF_{\le p}\neg f$, $true\ AU_{\le p} f$ and $\neg AF_{\le p}\neg f$, respectively.

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

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

Example 3.1 *The RPCTL formula*

$$[a_1]_{<q_1}((\langle a_2\rangle true) E U_{\ge q_2} (\langle a_3\rangle true))$$

means that for every state reachable after executed action a_1 within q_1 units of time, there exists an execution path such that 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. Derivation 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 state s of the periodic timed automaton satisfies the RPCTL property f. To begin with, we give a precise description of our problem.

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

$$\sigma \models WPC(s, f) \textit{ iff } (s, \sigma) \models f.$$ □

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

4.1. Case of DAG Models

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 function $WPC(s, f)$ for each construct of RPCTL formula f.

Definition 4.2 *Let s and f be a state of a parametric timed automaton and an RPCTL formula f, respectively. Then, function $WPC(s, f)$ is defined in Fig. 5. In Fig. 5, C is a set of all clocks, r,r_i,etc. denote subsets of clocks, $P[C + t/C]$ ($P[0/r]$) 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).* □

The meaning 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, $\sigma \not\models WPC(s, f)$ immediately implies $\sigma \models WPC(s, \neg f)$, 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 σ is a value-assignment such that $(s, \sigma) \models \langle a \rangle_{\sim p} f'$. From Definition 3.2, there must exist a concrete transition

$$
\begin{aligned}
WPC(s, true) &\stackrel{\text{def}}{=} true \\
WPC(s, \neg f) &\stackrel{\text{def}}{=} \neg WPC(s, f) \\
WPC(s, f_1 \wedge f_2) &\stackrel{\text{def}}{=} WPC(s, f_1) \wedge WPC(s, f_2) \\
WPC(s, \langle a \rangle_{\sim p} f) &\stackrel{\text{def}}{=} \exists t(0 \le t \wedge t \sim p \wedge (Inv(s) \wedge \\
&\qquad \bigvee_{i \in I(s,a)} \{P_i \wedge (Inv(s_i) \wedge WPC(s_i, f))[0/r_i]\})[C + t/C]) \\
&\qquad \text{where } I(s, a) = \{i | s \xrightarrow{a,[P_i],r_i} s_i\}, \\
WPC(s, f_1 EU_{\sim p} f_2) &\stackrel{\text{def}}{=} \exists t(0 \le t \wedge \\
&\qquad \forall t'((0 \le t' \wedge t' \le t) \Rightarrow WPC(s, f_1)[C + t'/C]) \wedge \\
&\qquad (Inv(s) \wedge (t \sim p \wedge WPC(s, f_2) \vee \bigvee_{i \in I(s)} \{P_i \wedge WPC(s_i, f_1 EU_{\sim (p-t)} f_2)\}))[C + t/C]) \\
&\qquad \text{where } I(s) = \{i | s \xrightarrow{a_i,[P_i],r_i} s_i\}, \\
WPC(s, f_1 AU_{\sim p} f_2) &\stackrel{\text{def}}{=} \forall t(0 \le t \Rightarrow \\
&\qquad \forall t'((0 \le t' \wedge t' \le t) \Rightarrow WPC(s, f_1)[C + t'/C]) \wedge \\
&\qquad (Inv(s) \Rightarrow (t \sim p \wedge WPC(s, f_2) \wedge \bigwedge_{i \in I(s)} \{P_i \Rightarrow WPC(s_i, f_1 AU_{\sim (p-t)} f_2)\}))[C + t/C])
\end{aligned}
$$

Figure 5. Function $WPC(s, f)$

sequence $(s, \sigma) \xrightarrow{t} (s, \sigma + t) \xrightarrow{a} (s', \sigma')$ such that $t' \sim p$ and $(s', \sigma') \models f'$. Thus, the following conditions must also hold:

- some timed automaton transition $s \xrightarrow{a,P,r} s'$ must exists.
- $\sigma + t$ must satisfy both $Inv(s)$ and P (Definition 2.2).
- σ' is a value-assignment $\sigma + t$ whose values of the clocks in r are reset to zero, i.e. $\sigma' = (\sigma + t)[r \rightarrow 0]$ (recall that $\sigma + t$ is the same value-assignment as σ except all clock values are increased by t, and $\sigma[r \rightarrow 0]$ is the same as σ except all the clocks in r are reset to zero, as defined in Definition 2.2) and it satisfies $Inv(s')$.
- $(s', \sigma') \models f'$, i.e. $\sigma' \models WPC(s', f')$.

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 $\sigma \models (t \sim p)$, $\sigma + t \models Inv(s) \wedge P$ and $(\sigma + t)[r \rightarrow 0] \models Inv(s') \wedge WPC'(s', f')$"

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

$WPC(s', f'))[C + t/C, 0/r]$. Therefore, the following condition holds:

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

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:

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

where $I(s,a) \stackrel{\text{def}}{=} \{i | s \xrightarrow{a,P_i,r_i} s_i\}$ 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 σ such that $(s,\sigma) \models f$, and t is some fresh 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 σ is a value-assignment such that $(s,\sigma) \models f_1 EU_{\sim p} f_2$. From Definition 3.2, there must exist some transition sequence $(s,\sigma) = (s_1,\sigma_1) \xrightarrow{t_1} (s_1,\sigma_1 + t_1) \xrightarrow{a_1} \cdots \xrightarrow{t_{k-1}} (s_{k-1},\sigma_{k-1} + t_{k-1}) \xrightarrow{a_{k-1}} (s_k,\sigma_k) \xrightarrow{t_k} (s_k,\sigma_k + t_k)$, such that $(s_k,\sigma_k + t_k) \models f_2$, $t_1 + \cdots + t_k \sim p$, and for any j $(1 \le j \le k)$ and for any t'_j $(0 \le t'_j < t_j)$, $(s_j,\sigma_j + t'_j) \models f_1$ holds. To obtain a recursive definition of $WPC(s, f)$, we divide the premise of the above statement into two cases, $k = 1$ (f_2 holds at the current state s_1) and $k \ge 2$ (f_2 holds at some future state s_k).

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

$$\begin{aligned} \sigma \models\ & \exists t(0 \le t \wedge t \sim p \wedge (Inv(s) \wedge WPC(s, f_2))[C + t/C] \\ & \wedge \forall t'((0 \le t' \wedge t' \le t) \Rightarrow WPC(s, f_1)[C + t'/C])) \end{aligned} \quad (1)$$

[Case $k \ge 2$]: If we assume $k \ge 2$, then there must exist a transition sequence $(s,\sigma) = (s_1,\sigma_1) \xrightarrow{t_1} (s_1,\sigma_1 + t_1) \xrightarrow{a_1} (s_2,\sigma_2)$ such that $(s_2,\sigma_2) \models f_1 EU_{\sim(p-t_1)} f_2$ holds, and for any t'_1 $(0 \le t'_1 \le t_1)$, $(s_1,\sigma_1 + t'_1) \models f_1$ holds. Considering that there may exist multiple transitions $s \xrightarrow{a_i,P_i,r_i} s_i$, the weakest condition of σ is obtained as follows:

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

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

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

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

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. parametric timed DAG-automata).

Theorem 4.1 *For every state s of a parametric timed DAG-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.:*

$$\forall \sigma.[\sigma \models WPC(s, f) \textit{ iff } (s, \sigma) \models f].$$ □

4.2. Case of Periodic Models

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 the special terminating state. We obtain the corresponding parametric timed DAG-automata (Fig. 6-(b)). Then, attach the copy of the corresponding parametric timed DAG-automaton to the special terminating state of itself. Finally, we have the parametric timed DAG-automaton which represents the first 2 periodic behavior (Fig. 6-(c)). We refer to such a model as a 2-*unfolding* of the periodic timed automaton. Similarly, we can also define a k-unfolding of a periodic timed automaton as the parametric timed DAG-automaton which represents the first k periodic behavior.

For any RPCTL operators *except* 'until' operators EU and AU (and all the operators derived from EU and AU such as AG, AF, EF and EG), the weakest condition of parameters are obtained by just checking the current state (and the

next state if it is 'next' operators) and checking for the *proper* subformulas recursively. Since the size of each subformula is strictly decreasing, the recursion eventually terminates and we can obtain the result. However, it is not the case for the 'until' operators EU and AU. The definition of $WPC(s, f_1 EU_{\sim p} f_2)$ contains not only recursive calls for its proper subformulas f_1 and f_2, but also the recursive call for the formula $f_1 EU_{\sim(p-t)} f_2$, which is not a proper subformula. This means that it may execute the function call of the form $WPC(s_i, f_1 EU_{\sim(p-t_1-\ldots-t_k)} f_2)$ forever if the model contains some loops. 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).

Formally, it is proved by the following lemma:

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

$$(s, \sigma) \models f_1 EU_{\sim p} f_2 \text{ if and only if } (s, \sigma) \models f_1 EU_{\sim p'} f_2$$

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) Let α, α', etc. denote finite execution paths such as $(s_1, \sigma_1) \xrightarrow{t_1} (s_1, \sigma_1 + t_1) \xrightarrow{a_1} (s_2, \sigma_2) \xrightarrow{t_2} \ldots \xrightarrow{a_{k-1}} (s_k, \sigma_k)$. Let $ET(\alpha)$ denote the execution time of the path α, i.e., $ET(\alpha) = t_1 + \cdots + t_{k-1}$. Consider there exists a path α which begins with (s, σ) such that $(s, \sigma) \models f_1 EU_{\sim p} f_2$ (which implies $ET(\alpha) \sim p$). The case of $0 \leq ET(\alpha) < 3T$ is trivial, since if $(s, \sigma) \models f_1 EU_{\sim p} f_2$ holds for α, p must be less than $3T$ for any case of $\sim \in \{<, \leq, >, \geq, =\}$, which implies $p = p'$ from the definition of p', and thus obviously $(s, \sigma) \models f_1 EU_{\sim p'} f_2$ holds, and vice versa. Consider the case of $ET(\alpha) \geq 3T$. As illustrated in Fig. 7, there must also exist a path α' containing at least one cycle, such that $0 \leq ET(\alpha') < 3T$ and $(s, \sigma) \models f_1 EU_{\sim p} f_2$ holds. α' is obtained by removing cycles beginning with the initial state from α until $ET(\alpha) < 3T$ holds, while leaving at least one cycle.[2] Obviously, the relation between $ET(\alpha)$ and $ET(\alpha')$ is $ET(\alpha') = ET(\alpha) - m \times T$, where m is the number of the removed cycles. Therefore, $ET(\alpha) \sim p$ implies $ET(\alpha') \sim p'$ where $p' = p - m \times T$, and immediately we have $(s, \sigma) \models f_1 EU_{\sim p'} f_2$. Conversely, if there exists a path α' containing at least one cycle, such that $0 \leq ET(\alpha') < 3T$ and $(s, \sigma) \models f_1 EU_{\sim p'} f_2$,

[2] Note that in a periodic timed automaton, all clocks are initially zero and reset to zero when returned to the initial state, whereas any other variables (parameters) are never modified during execution. So, the possible behaviour from the initial state is always the same, no matter how it is reached. Thus, after removing cycles beginning with the initial state from the execution path α, it is still an executable path. Moreover, f_2 holds at the last state of the path and along with the path, f_1 always holds until f_2 holds. Therefore, if $ET(\alpha) \sim p$ holds, then $f_1 EU_{\sim p} f_2$ still holds on the shortened path.

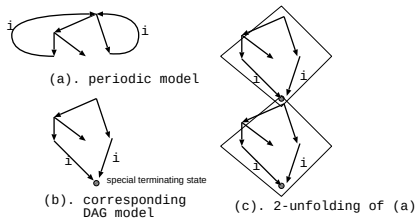

Figure 6. Unfolding of Periodic Timed Automata

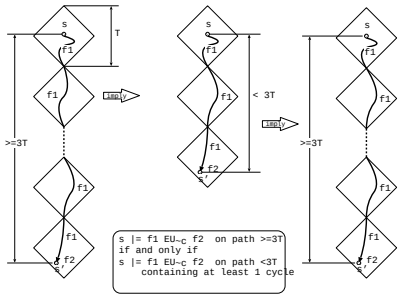

Figure 7. Illustration of Lemma 4.1

there must also exist a path α such that $ET(\alpha) \geq 3T$ and $(s, \sigma) \models f_1 EU_{\sim p} f_2$, as also illustrated in Fig. 7. In this case, α is obtained by duplicating intermediate cycles of α' (α' should have at least one cycle by the assumption) repeatedly until $ET(\alpha) \geq 3T$ and $ET(\alpha) \sim p$ hold. The case of $f_1 AU_{\sim p} f_2$ is similar (we consider a path which *violates* $f_1 AU_{\sim p} f_2$, instead), and we omit the detailed proof due to the space limitation. □

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 $WPC3(s, f)$ derives the weakest parameter condition such that s satisfies f within 3 periods.

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.:*

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

5. Computational Complexity

In this section, we evaluate the computational complexity of the functions $WPC(s, f)$ and $WPC3(s, f)$. Direct implementation of the recursive function $WPC(s, f)$ (and $WPC3(s, f)$) may perform very inefficiently because of repetitive function calls of the same tuple of arguments (s, f), which is redundant. Thus, we evaluate the complexity when we use a cache to avoid unnecessary computation. We assume that the function $WPC(s, f)$ has its own cache table, which has the entry (s, f, P) if $WPC(s, f)$ has already been computed and the result is P. If such a function is called, the cache table is checked first, and if its value has already been computed, the cached result is returned. Otherwise,

the computation is performed, the result is registered into the cache, and the result is returned.

The complexity of the above algorithm is evaluated as follows. For simplicity, we here ignore the length of the resulting formulas (i.e. we ignore the cost to simplify and/or check satisfiability of the formulas). Under the assumption, time and space complexity coincides, that is, the time necessary to compute $WPC(s, f)$ is equal to the size of the cache table, which is equal to the maximum number of the different tuples of arguments with which $WPC(s, f)$ will be called. From the definition of $WPC(s, f)$, the number of the different tuples of arguments is bounded by $n \times (m_1 + m_2)$, where n is the number of control states reachable from s, and m_1 (m_2) is the number of the different subformulas of f (the number of the different derived formulas of the form $f_1 op_{\sim(p-t_1-\cdots-t_k)} f_2$ ($op \in \{EU, AU\}$), respectively). m_1 is equal to the number of nodes of the syntax tree of f. Thus, m_1 is $O(m \log m)$, where m is the length of f. m_2 is equal to $m_1 \times l$, where l is the depth of the DAG, i.e. the length of the longest directed paths from the root to leaf nodes of the DAG (here we refer to a leaf node of the DAG as a node which has no outgoing edges). Obviously, l does not exceed the number n of nodes. Overall, the number of different tuples of arguments (= the size of cache table) is $O(n \times m \log m \times n) = O(n^2 m \log m)$. Since we have only to perform at least one computation for each tuple of arguments, the number of all computation is also $O(n^2 m \log m)$. Therefore, the time and space complexity of $WPC(s, f)$ is $O(n^2 m \log m)$.

The complexity of $WPC3(s, f)$ is equal to the $WPC(s, f)$ for 3-unfolding DAG model. Let n denote the number of nodes of a periodic model. Then, from the definition of unfolding, the depth of the corresponding 3-unfolding is $3n$. Therefore, by replacing n with $3n$, we also conclude that the time and space complexity of $WPC3(s, f)$ is $O(n^2 m \log m)$.

Theorem 5.1 *The time and space complexity of $WPC(s, f)$ and $WPC3(s, f)$ for a DAG/periodic model M is $O(n^2 m \log m)$, where n is the number of control states in M, and m is the length of RPCTL formula f.* □

6. 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 the restricted class of timed automata, many real-time applications such as audio/video streaming, time sharing task schedulers can be specified as periodic timed automata.

The future works are to develop and improve the efficiency of the implementation of the algorithm, to extend our algorithm to simplify the parameter condition and to handle multiple periodic timed automata which run concur-

rently. We are also considering to extend our method to handle some internal state variables of finite domain to improve expressiveness of the model.

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] J. R. Burch, E. M. Clarke, K. L. McMilan, D. L. Dill, and L. J. Hwang, "Symbolic model-checking: 10^{20} states and beyond," *Information and Computation*, vol. 98, no. 2, pp. 142–170, 1992.

[3] R. Alur, T. A. Henzinger, and P. Ho, "Automatic symbolic verification of embedded systems," *IEEE Trans. on Software Engineering*, vol. 22, no. 3, pp. 181–201, 1996.

[4] T. Bultan, R. Gerber, and W. Pugh, "Symbolic model checking of infinite state systems using presburger arithmetic," in *Proc. of Int. Conf. on Computer Aided Verification (CAV'97)*, vol. 1254 of *Lecture Notes in Computer Science*, pp. 400–411, Springer-Verlag, 1997.

[5] F. Wang, "Parametric timing analysis for real-time systems," *Information and Computation*, vol. 130, no. 2, pp. 131–150, 1996.

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

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

[8] A. Nakata, T. Higashino, and K. Taniguchi, "Time-action alternating model for timed LOTOS and its symbolic verification of bisimulation equivalence," in *Proc. of Joint Int'l Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV'96)* (R. Gotzhein and J. Bredereke, eds.), pp. 279–294, IFIP, Chapman & Hall, 1996.

[9] E. A. Emerson and R. J. Trefler, "Parametric quantitative temporal reasoning," in *Proc. IEEE Int'l Conf. on Logic in Computer Science (LICS'99)*, pp. 336–343, 1999.

[10] M. Hennessy and H. Lin, "Symbolic bisimulations," *Theoretical Computer Science*, vol. 138, pp. 353–389, 1995.

[11] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[12] T. Bolognesi and F. Lucidi, "LOTOS-like process algebras with urgent or timed interactions," in *Formal Description Techniques, IV* (K. R. Parker and G. A. Rose, eds.), pp. 249–264, IFIP, Elsevier Science Publishers B.V.(North-Holland), 1992.

[13] R. De Nicola and F. W. Vaandrager, "Action versus state based logics for transition systems," in *Proc. of Ecole de Printemps on Semantics of Concurrency* (I. Guessarian, ed.), vol. 469 of *Lecture Notes in Computer Science*, pp. 407–419, Springer-Verlag, 1990.

[14] M. Hennessy and R. Milner, "Algebraic laws for nondeterminism and concurrency," *J. ACM*, vol. 32, pp. 137–161, 1985.