Double Depth First Search Based Parametric Analysis for Parametric Time-Interval Automata*

Tadaaki Tanimoto Akio Nakata Hideaki Hashimoto Teruo Higashino

Graduate School of Information Science and Technology,
Osaka University

Abstract

In this paper, we propose a parametric model checking algorithm for a subclass of Timed Automata called Parametric Time-Interval Automata(PTIA). In a PTIA, we can specify upper- and lower-bounds of the execution time (time-interval) of each transition using *parameter variables*. The proposed algorithm takes two inputs, a model described in a PTIA and a property described in a PTIA accepting all *invalid* infinite/finite runs (called a *never claim*), or *valid finite* runs of the model. In the proposed algorithm, firstly we determinize and complement the given property PTIA if it accepts valid finite runs. Secondly, we *accelerate* the given model, that is, we regard all the actions that are not appeared in the given property PTIA as *invisible* actions and eliminate them from the model while preserving the set of visible traces and their timings. Thirdly, we construct a parallel composition of the model and the property PTIAs which is accepting all invalid runs that are accepted by the model. Finally, we perform the extension of *Double Depth First Search(DDFS)*, which is used in the automata-theoretic approach to Linear-time Temporal Logic(LTL) model checking, to derive the weakest parameter condition in order that the given model never executes the invalid runs specified by the given property.

Key words: parametric model checking, timed automata, Büchi automata, complementation, acceleration, emptiness checking, double depth first search

1 Introduction

In recent years, hardware/software systems have been widely used in the areas that high-reliability and real-timeliness are required, and the importance of system verification techniques have been increasing. One of the most important verification techniques is *model checking*[1]. As for real-time system verification, there exist some useful model checking tools[2, 3]. In such tools, the system models are described in Timed Automata[4]. A Timed Automaton is a state transition model with *clock variables*. However, in order to verify Timed Automata using such a tool, we have to fix various design parameters such as timeouts, delays, and so on, to specific values. If the desired property is not satisfied, we have to explore the appropriate values of the parameters by try and error using such a traditional model checking tools. In designing such a system containing design parameters, we would rather want to derive a condition of parameters (*parameter condition*) in order to satisfy the given property, and then fix the parameters to the values satisfying the derived parameter condition. Such a method to derive parameter condition is called *parametric model checking* [5].

There exist some parametric model checking tools HyTech[7], TReX[10], and so on. However, they have some problems — termination of the procedure is not guaranteed, time and space complexity of the procedure tend to be extremely large if the numbers of states and parameters grow larger, and so on. One of the considerable approaches to such problems is restricting the expressing power of Timed Automata. Comparison between existing parametric model checking methods are summarized in Table 1. For example, [6] proposed a parametric model checking algorithm in that both a model and a property are written in finite parametric timed automata (PTA) with one integer clock variable. Although the complexity issue is not discussed in [6], their algorithm apparently runs in exponential time or greater. In [8], a parametric model checking algorithm

*This work is partially supported by Grant-in-Aid for Young Scientists (B) (16700062, 2004–2005) from the Ministry of Education, Science, Sports and Culture(MEXT), Japan.

Table 1: Comparison of Parametric Model Checking Methods

proposed by	model	property	decidable?	time complexity
Alur et.al.[6]	finite PTA w/ one integer clock	safety/reachability	yes	unknown (exponential or greater)
		finite PTA w/ one integer clock (never claim)	yes	unknown (exponential or greater)
		finite PTA w/ one integer clock	no	—
Henzinger et.al.[7]	parametric hybrid automaton	safety/reachability	no	—
Wang[8]	timed automaton	safety/reachability	yes	doubly exponential
		parametric timed CTL	yes	doubly exponential
Wang[9]	statically parametric automaton	safety/reachability	yes	triply exponential
		parametric timed CTL	yes	triply exponential
Annichini et.al.[10]	PTA	safety/reachability	no	—
Nakata et.al.[11]	periodic PTA	safety/reachability	yes	doubly exponential
		real-time parametric CTL	yes	doubly exponential
Bruyere et.al.[12]	PTA w/ one integer clock	safety/reachability	yes	unknown (exponential or greater)
		parametric timed CTL w/o equality in time constraints	yes	unknown (exponential or greater)
Mori et.al.[13]	concurrent periodic EFSMs	safety/reachability	yes	doubly exponential
		real-time parametric CTL	yes	doubly exponential
this paper	finite/Büchi PTIA	safety/reachability	yes	exponential
		finite/Büchi PTIA (never claim)	yes	exponential
		finite PTIA	yes	doubly exponential

is proposed for non-parametric timed automata and timed and parametric extension of Computation Tree Logic(CTL). The algorithm of [8] is extended to *statically-parametric* automata in [9], that is, timed automata having only *static parameters* (i.e. parameters which may affect whether a transition is on or off, but may not affect its execution time). However, both of [8] and [9] do not handle parameters written in a model and may affect the execution times of transitions. In [11], a parametric CTL model checking method is proposed for periodic timed automata, which is forced to return to their initial states when some specified period has elapsed, and timing parameters can be written. Applicability of [11] is limited because of the restriction of the model. In [12], the algorithm of [6] is extended to the parametric CTL model checking of PTA having only one integer clock variable. However, the time complexity is still exponential or greater (although not evaluated in [12]).

On the other hand, considering verification problems of web services or business process specifications[14, 15], each system's behavior itself merely depends on timing of each transition. To guarantee the performance of such systems as well as its correctness, it is useful to derive a parameter condition of the execution time of each transition which ensures the entire performance of the system. In this case, it is sufficient to describe the time constraint of each transition in a form of a time interval containing parameter variables. According to [16], the expressive power of such a restricted form of timing constraints is strictly less than traditional timed automata.

Therefore, in this paper, we propose a subclass of Timed Automata, Parametric Time-Interval Automata (PTIA in short), and propose a parametric model checking algorithm for PTIAs. Each transition condition of a PTIA does not depend on the execution time of past transitions. Thus, we naturally expect that the traditional (untimed) model checking methods can be easily applied with an appropriate extension. One of such traditional model checking methods is the automata-theoretic approach to Linear-time Temporal Logic(LTL) model checking[17],[1, Chap. 9]. In the method of [17], firstly, a Büchi automaton[1] accepting all the infinite sequences that violate the given LTL property is constructed. Secondly, the product automaton of the given model and the constructed Büchi automaton is constructed. The constructed product automaton accepts all the infinite sequences which is executable by the given model and violates the given property. Finally the emptiness of the accepting language of the product automaton is checked. The emptiness check of Büchi automata can be efficiently done by *Double Depth First Search*[18],[1, Sect. 9.3] (DDFS for short). If it is empty, then we can conclude that the given model does not violate the given LTL property. Otherwise, the accepting run of the product automaton is a counter-example (an evidence that the given model does not satisfy the given property).

In a non-parametric (traditional) model checking, it is sufficient to try to find one counter-example to check whether a given property is satisfied. However, it is not the case for parametric model checking. In parametric model checking, we must consider all possible cases to construct the parameter condition which makes the accepting language of the product automaton empty. Consider that there are two accepting runs for the

[1] A Büchi automaton is a kind of a finite automaton accepting infinite sequences which make it visit one of the accepting states infinitely often.

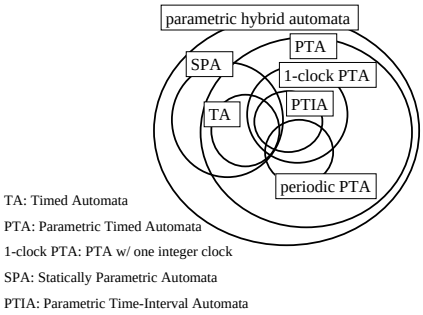

Figure 1: Class Hierarchy of Models in Table 1

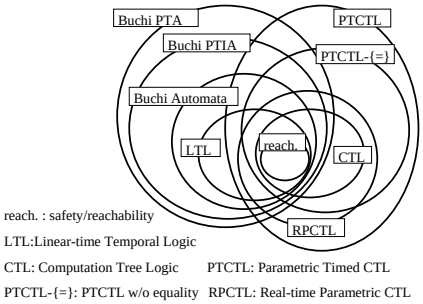

Figure 2: Class Hierarchy of Properties in Table 1

product automaton, and each run is executable at different parameter values. Then, the parameter condition in order not to satisfy the property is the disjunction of the parameter conditions each of which activates each accepting run. The parameter condition in order to satisfy the property is its negation. Therefore, in the parametric model checking, we must find all the accepting runs to construct the expected parameter condition.

The proposed algorithm takes two inputs, a model described in a PTIA and a property described in a PTIA accepting all *invalid* infinite/finite runs (called a *never claim*[19]), or *valid finite* runs of the model. The output of the algorithm is the weakest parameter condition, that is, the necessary and sufficient condition of parameters, in order that the given model satisfies the given property. In the proposed algorithm, firstly we determinize and complement the given property PTIA if it accepts valid finite runs. Secondly, we *accelerate* the given model, that is, we regard all the actions that are not appeared in the given property PTIA as *invisible* actions and eliminate them from the model while preserving the set of visible traces and their timings. Thirdly, we construct a parallel composition of the model and the property PTIAs which is accepting all invalid runs that are accepted by the model. Since the parallel composition of PTIAs does not always fall into the class of PTIAs, we show some sufficient conditions of the set of PTIAs whose parallel composition can be transformed into an equivalent single PTIA. Finally, we perform the extension of DDFS to derive the weakest parameter condition in order that the given model never executes invalid runs specified by the given property.

Class hierarchies for models and properties in Table 1 are shown in Figs. 1, 2, and 3. To the best of our knowledge, there are no parametric model checking algorithm that can handle the class of Büchi PTIAs as a property description language. The class of Büchi PTIAs is a proper superclass of Büchi automata, which is used by SPIN model checker[19] as a property description language. Therefore, our parametric model checking algorithm is a conservative extension of SPIN, that can handle a new class of models and properties. Although the time complexity of our method is exponential in the worst case even if the property is safety/reachability, it is not practically a serious problem in many cases since the size of the property PTIA is rather small compared to the model PTIA, and the number of states can be reduced by the proposed acceleration algorithm.

The rest of this paper is organized as follows. In Section 2, the proposed model is defined. In Section 3, the parametric model checking problem is formally defined and the overview of the proposed parametric model checking procedure is presented. In Section 4, we describe the method to determinize and complementing PTIAs

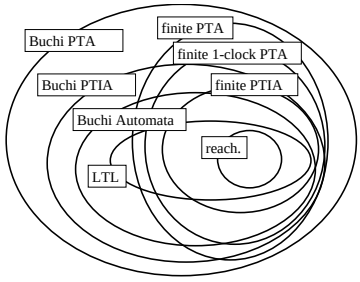

Figure 3: Detailed Class Hierarchy of Linear Properties in Table 1

accepting finite sequences. In Section 5, the acceleration algorithm for PTIAs is described. In Section 6, we define a parallel composition of PTIAs, and give a sufficient condition for a set of PTIAs in order that their parallel composition is transformed into a single PTIA. In Section 7, we describe the extended DDFS algorithm to derive the weakest parameter condition that the set of accepting sequences of a given PTIA is empty. Finally, we conclude this paper in Section 8.

2 Preliminaries

A parametric time-interval automaton(PTIA) is a subclass of a parametric timed automaton[6] (PTA for short). At first, we recall the formal definition of PTAs. Let Act and Var denote a set of actions and a set of variables, respectively. We denote the set of real-numbers by $\mathbf{R}$ and the set of non-negative real-numbers by $\mathbf{R}^+$. Let $Pred(Var)$ denote the minimum set of formulas satisfying $e1 \sim e2 \in Pred(Var)$ for $\sim \in \{<, \leq, >, \geq, =\}$, and if $P, Q \in Pred(Var)$ then $P \wedge Q \in Pred(Var)$ and $P \vee Q \in Pred(Var)$, where $e1$ and $e2$ are linear arithmetic expression (that is, only addition and subtraction are allowed) over variables in Var and constants in $\mathbf{R}$.

Definition 1 *A* parametric timed automaton *M is a tuple $\langle S, C, PVar, E, F, s_{init}\rangle$, where S is a finite set of* control states *(also referred to as* locations*), $C \subseteq Var$ is a set of* clock variables*, $PVar \subseteq Var$ is a finite set of* parameters*, $E \subseteq S \times (Act \cup \{\tau\}) \times Pred(C \cup PVar) \times 2^C \times S$ is a transition relation, $F \subseteq S$ is a set of accepting states, and s_{init} is the initial state. Note that τ represents an* internal action*. On the other hand, every other action in Act represents an* observable action*. 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 clock variables and parameters satisfy the formula P (called a *transition condition*), and after executed, the state moves into s_j and the clock variables in the set r are reset to zero. In any state s, the values of all the clock variables in C increase continuously at the same rate, representing the time passage.

Formal semantics of parametric timed automata is the same as that of parametric timed automata[6], which 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 $CS \stackrel{\text{def}}{=} \{(s, \sigma) | s \in S, \sigma \in Val\}$ be a set of concrete states.

To define the semantic model of a PTA, we need the following definition:

Definition 2 *Let $\sigma + v$ and $\sigma[r \to 0]$ be the value-assignments derived from σ, which are defined as follows:*

$$\text{For any } x \in PVar \cup C$$

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

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

□

Formally, the semantic model for a PTA is defined as follows.

Definition 3 *The* semantic model *for a PTA w.r.t. a value-assignment $\sigma_{init} \in Val$ is the* timed labelled transition system *(*timed LTS *for short) $\langle CS, Act \cup \mathbf{R}^+ \cup \{\tau\}, CE, (s_{init}, \sigma_{init}[C \to 0])\rangle$, where the set of states is CS, the set of labels is $Act \cup \mathbf{R}^+ \cup \{\tau\}$, the initial state is $(s_{init}, \sigma_{init}[C \to 0])$, and the transition relation $CE \subseteq CS \times (Act \cup \mathbf{R}^+ \cup \{\tau\}) \times CS$ is defined as the minimum set that satisfies the following conditions (in the following, we write $(s,\sigma) \xrightarrow{l} (s',\sigma')$ if $((s,\sigma), l, (s',\sigma')) \in CE$):*

- $(s,\sigma) \xrightarrow{v} (s, \sigma + v)$ *if* $v \in \mathbf{R}^+$,
- $(s,\sigma) \xrightarrow{a} (s', \sigma[r \to 0])$ *if* $a \in Act \cup \{\tau\}$, $s \overset{a[P],r}{\longrightarrow} s'$, *and* $\sigma \models P$. □

Now, we define PTIAs, a subclass of PTAs. Informally, the difference of a PTIA and a PTA is that, unlike a PTA, a PTIA has only one clock variable and the clock variable is always reset to zero at any transition. Since there is only one clock variable, the set of transition conditions $Pred(Var)$ in PTAs is restricted to $Intvl(Var)$, the set of time-intervals using one clock variable.

Formally, a PTIA is defined as follows. Let $Intvl(Var)$ denote the minimum set of formulas satisfying $e1 \le t \in Intvl(Var)$, $t \le e2 \in Intvl(Var)$, and if $P, Q \in Intvl(Var)$ then $P \wedge Q \in Intvl(Var)$ and $P \vee Q \in Intvl(Var)$, where $e1$ and $e2$ are linear arithmetic expression (that is, only addition and subtraction are allowed) over variables in $Var \setminus \{t\}$ and constants in $\mathbf{R}$, and $t \in Var$ is the clock variable representing the elapsed time since the latest visit of the current control state. Apparently, $Intvl(Var) \subseteq Pred(Var)$.

Definition 4 *A* parametric time-interval automaton *M is a tuple $\langle S, \{t\}, PVar, E, F, s_{init}\rangle$, where $E \subseteq S \times (Act \cup \{\tau\}) \times Intvl(Var) \times S$ and the semantics of M is the same as the corresponding parametric timed automaton $M' = \langle S, \{t\}, Intvl(Var), E', F, s_{init}\rangle$ where $(s, a, P, \{t\}, s') \in E'$ iff $(s, a, P, s') \in E$. We write $s \overset{a@?t[P]}{\longrightarrow} s'$ if $(s, a, P, s') \in E$.* □

If the timed LTS for a PTIA M is deterministic, that is, there are no two different concrete transitions $(s,\sigma) \xrightarrow{a} (s_1, \sigma_1')$ and $(s,\sigma) \xrightarrow{a} (s_2, \sigma_2')$, we call M a *deterministic PTIA*. Otherwise, we call it a *nondeterministic PTIA*.

In the following, we define a timed weak transition relation for timed LTSs, that is, a transition relation where time is observable but internal actions are not observable.

Definition 5 *A* timed weak transition relation *$\to_w$ on states of a timed LTS $\langle CS, Act \cup \mathbf{R}^+ \cup \{\tau\}, CE, (s_0, \sigma_0)\rangle$ is defined as follows:*

1. $\xrightarrow{\tau}_w \overset{\text{def}}{=} (\xrightarrow{0})^* (\xrightarrow{\tau})^*)^*$,
2. $(s,\sigma) \xrightarrow{v}_w (s',\sigma')$ $(v \in \mathbf{R}^+) \overset{\text{def}}{=} \exists v_1, v_2, \ldots, v_n \in \mathbf{R}^+ \; [\; v = \sum_{i=1}^n v_i \wedge \exists s_1, \sigma_1, \sigma_1', \ \ldots, \ s_n, \sigma_n, \sigma_n'$ *s.t.* $(s,\sigma) \xrightarrow{\tau}_w (s_1,\sigma_1) \xrightarrow{v_1} (s_1,\sigma_1') \cdots (s_n,\sigma_n) \xrightarrow{v_n} (s_n,\sigma_n') \xrightarrow{\tau}_w (s',\sigma') \;]$,
3. $\xrightarrow{a}_w$ $(a \in Act) \overset{\text{def}}{=} \xrightarrow{\tau}_w \xrightarrow{a} \xrightarrow{\tau}_w$. □

We define a *run* (execution sequence, or trace) of a PTIA M using timed weak transitions.

Definition 6 1. *A* symbolic finite run *π_f of PTIA M is a finite sequence of transitions on M such that $\pi_f = s \overset{a_1@?[P_1]}{\longrightarrow} s_1 \cdots s_{n-1} \overset{a_n@?t[P_n]}{\longrightarrow} s_n$. A symbolic finite run π_f is* finitely accepting *iff $s_n \in F$. We denote a set of all accepting symbolic finite run beginning with state s by ${\mathcal{L}_M}^f(s)$. We abbreviate $\mathcal{L}_M^f(s_{init})$ as $\mathcal{L}_M^f$.*

2. *A* concrete finite run *$\pi_f(\sigma)$ of PTIA M and a value-assignment $\sigma \in Val$ is a finite sequence of transitions on M such that $\pi_f(\sigma) = (s,\sigma) \xrightarrow{t_1}_w \xrightarrow{a_1}_w (s_1,\sigma) \cdots (s_{n-1},\sigma) \xrightarrow{t_n}_w \xrightarrow{a_n}_w (s_n,\sigma)$. A concrete finite run $\pi_f(\sigma)$ is* finitely accepting *iff $s_n \in F$. We denote a set of all accepting concrete finite run beginning with state s and a value-assignment $\sigma \in Val$ by ${\mathcal{L}_M}^f(s,\sigma)$. We abbreviate $\mathcal{L}_M^f(s_{init},\sigma)$ as $\mathcal{L}_M^f(\sigma)$.*

3. *A* symbolic infinite run π *of PTIA* M *is an infinite sequence of transitions on* M *such that* $\pi_\omega = s \overset{a_1@?[P_1]}{\longrightarrow} s_1 \cdots s_{n-1} \overset{a_n@?t[P_n]}{\longrightarrow} s_n \cdots$. *A symbolic infinite run* π *is* accepting *iff for an infinite number of indices* i, $s_i \in F$. *We denote a set of all accepting symbolic finite run beginning with state* s *by* $\mathcal{L}_M(s)$. *We abbreviate* $\mathcal{L}_M(s_{init})$ *as* $\mathcal{L}_M$.

4. *A* concrete infinite run $\pi(\sigma)$ *of PTIA* M *and a value-assignment* $\sigma \in Val$ *is an infinite sequence of transitions on* M *such that* $\pi(\sigma) = (s,\sigma) \xrightarrow{t_1}_w \xrightarrow{a_1}_w (s_1,\sigma) \cdots (s_{n-1},\sigma) \xrightarrow{t_n}_w \xrightarrow{a_n}_w (s_n,\sigma) \cdots$. *A concrete infinite run* $\pi(\sigma)$ *is* accepting *iff for an infinite number of indices* i, $s_i \in F$. *We denote a set of all accepting concrete finite run beginning with state* s *and a value-assignment* $\sigma \in Val$ *by* $\mathcal{L}_M(s,\sigma)$. *We abbreviate* $\mathcal{L}_M(s_{init},\sigma)$ *as* $\mathcal{L}_M(\sigma)$. □

If a PTIA is intended to accept finite runs, we call it a *finite PTIA*. Otherwise, we call it a *Büchi PTIA*. Note that only the difference of a finite PTIA and a Büchi PTIA is the interpretation of the acceptance conditions.

Equivalence of PTIAs is defined as follows:

Definition 7 *Two given Büchi PTIAs (finite PTIAs)* M_1 *and* M_2 *are* timed weak trace equivalent *iff* $\forall\sigma \in Val$ $[\mathcal{L}_{M_1}(\sigma) = \mathcal{L}_{M_2}(\sigma)]$ *(*$\forall\sigma \in Val$ $[\mathcal{L}^f_{M_1}(\sigma) = \mathcal{L}^f_{M_2}(\sigma)]$*, respectively).* □

To relate symbolic runs and concrete runs w.r.t. a given value-assignment σ, we define the following notations:

Definition 8 *We say that* σ *satisfies a symbolic run* π *on* M *iff* $\sigma \models \exists t[P_i]$ *holds for any transition condition* P_i *appeared in* π*, denoted by* σ *sat.* π. □

Then, we have the following proposition:

Proposition 1 *For any* $\sigma \in Val$, $\pi(\sigma) \in \mathcal{L}_M(\sigma)$ *iff [*$\pi \in \mathcal{L}_M$ *and* σ *sat.* π*] (*$\pi^f(\sigma) \in \mathcal{L}^f_M(\sigma)$ *iff [*$\pi^f \in \mathcal{L}^f_M$ *and* σ *sat.* π^f*], respectively).* □

We define the following operations on PTIAs:

Definition 9 *We denote a complement automaton of PTIA* M *by* M^c*, that is,* M^c *accepts a run* π *iff* M *does not accept* π. *We denote a parallel composition (a product automaton) of two PTIAs* M_1 *and* M_2 *by* $M_1||M_2$*, that is,* $M_1||M_2$ *accepts a run* π *iff the projection of* π *to the executable actions of* M_i *is an accepting run of* M_i *for each* $i \in \{1,2\}$[2]. □

3 Parametric Model Checking of PTIA

In this section, we first formalize the parametric model checking problem of PTIA, then give an overview of the procedure to solve the problem.

3.1 Problem Formulation

The formal definition of the parametric model checking problem of PTIA is as follows:

Definition 10 *Let* M *be a model written in a (finite or Büchi) PTIA. Let* M^f_p *(or* M^ω_p*) be a property written in a finite PTIA (or a Büchi PTIA, respectively).* Parametric model checking problem *is to derive the weakest parameter condition* P *satisfying*

$$\forall\sigma[\sigma \models P \text{ iff } \mathcal{L}^f_M(\sigma) \cap \mathcal{L}^f_{M^f_p}(\sigma) = \emptyset]$$

$$(\text{ or } \forall\sigma[\sigma \models P \text{ iff } \mathcal{L}_M(\sigma) \cap \mathcal{L}_{M^\omega_p}(\sigma) = \emptyset], \text{ respectively}),$$

where $\sigma \in Val$. *Note that we call* P *is the* weakest *parameter condition iff* $\forall\sigma[\sigma \models Q \Rightarrow \sigma \models P]$ *for any parameter condition* Q. □

[2]For the concrete definition of $M_1||M_2$, see Definition 14 in Sect. 6.

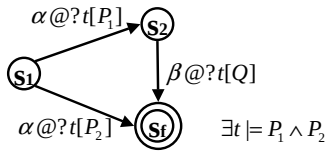

Figure 4: Example of Nondeterministic PTIA

3.2 Overview of Parametric Model Checking Procedure

Let M_p be an input property described by either a Büchi PTIA M_p^ω accepting invalid infinite runs (called a *never claim*[19]), or a finite PTIA M_p^f accepting invalid finite runs. Parametric model checking procedure is as follows:

Step 1. If M_p is a finite PTIA M_p^f accepting valid finite runs, construct a complement PTIA (co-PTIA) M_p^c from M_p (see Section 4). Otherwise let $M_p^c = M_p$.

Step 2. Replace all the actions of M which do not appear in M_p^c with the internal action. Then, apply *acceleration* (see Section 5) for M and M_p^c to eliminate all internal actions and reduce the number of states while preserving timed weak trace equivalence. Let M' and $M_p^{c\prime}$ be the resultant PTIAs for M and M_p^c, respectively.

Step 3. Construct *parallel composition* of M' and $M_p^{c\prime}$: $M'||M_p^{c\prime}$[3] (see Section 6).

Step 4. Perform *extended depth first search* (see Section 7) on $M'||M_p^{c\prime}$ to derive the weakest parameter condition P.

4 Determinizing and Complementing PTIAs

Complementing a nondeterministic PTIA can be performed by first determinizing it using a small extension of the traditional subset construction method for finite automata[21, Chap. 1], and then complementing the set of accepting state F. Not every Büchi automaton, however, has an equivalent deterministic Büchi automaton. In this section, we only discuss complementing a finite PTIA.

Since a finite PTIA is different from a finite automaton in that PTIA has timed transition additionally, we need to take care of it in the determinization of PTIA. Indeed, if we only apply subset construction to PTIA for determinization, we will fail to get a deterministic PTIA.

Example 1 *Fig. 4 shows a nondeterministic finite PTIA M_{ex}, where we assume $\exists t[P_1(t) \wedge P_2(t)]$. In Fig. 4 and the following figures, a doubled circle represents an accepting state. Direct application of subset construction to M_{ex} gives a deterministic PTIA M'_{ex} as shown in Fig. 5. The concrete model for M'_{ex}, shown in Fig. 5, has a false path executing the action β after some time passage satisfying P_2 and the action α.* □

To avoid this problem, we modify the given PTIA $M = \langle S, \{t\}, PVar, E, F, s_{init}\rangle$ to the PTIA M' having only mutually exclusive or exactly the same transition conditions for multiple outgoing transitions of each state with the same action name by using the following transformation $M' = dj(M)$:

Definition 11

$$dj(M) \stackrel{\text{def}}{=} \langle S, \{t\}, PVar, \bigcup_{s\in S, a\in Act} \{djout(out(s,a))\}, F, s_{init}\rangle,$$

[3]Note that if M is deterministic, then M' is also deterministic. Since trace equivalence coincides bisimulation equivalence when automata are deterministic[20, Chap. 11], M and M' are bisimilar. Since bisimulation equivalence is congruent w.r.t. parallel composition[20, Chap. 7], $M||M_p$ and $M'||M_p$ are (timed) weakly bisimilar and thus they are timed weak trace equivalent. If M is nondeterministic, then we first convert it into deterministic one by the proposed algorithm in Section 4 and then apply acceleration algorithm in Section 5.

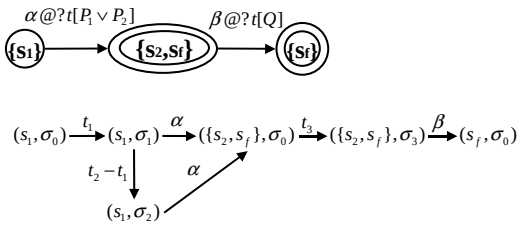

Figure 5: Direct Subset Construction of Fig. 4 and its Concrete Model

$$
\begin{array}{l}
\textit{where,} \\
Nxt(s,a) \stackrel{\text{def}}{=} \{i | s \stackrel{a@?t[P_i]}{\longrightarrow} s_i, a \in Act, \\
\qquad\qquad P_i \in Intvl(Var), s_i \in S\} \\
out(s,a) \stackrel{\text{def}}{=} \{s \stackrel{a@?t[P_i]}{\longrightarrow} s_i | a \in Act, \\
\qquad\qquad P_i \in Intvl(Var), s_i \in S\} \\
djout(E') \stackrel{\text{def}}{=} \\
\quad \begin{cases}
E' & \textit{if } |E'| = 1, \\
\{s \stackrel{a@?t[\bigwedge_i \{\neg P_i\} \wedge Q]}{\longrightarrow} s'\} \cup \bigcup_i \{s \stackrel{a@?t[P_i \wedge Q]}{\longrightarrow} s'\} & \\
\cup \bigcup_i \{s \stackrel{a@?t[P_i \wedge Q]}{\longrightarrow} s_i\} \cup \bigcup_i \{s \stackrel{a@?t[P_i \wedge \neg Q]}{\longrightarrow} s_i\} & \\
\qquad \textit{if } |E'| \geq 2 \textit{ and} & \\
\qquad djout(E' \setminus \{s \stackrel{a@?t[Q]}{\longrightarrow} s'\}) = & \\
\qquad\quad \bigcup_{i \in \{1,\ldots,l\} \subseteq Nxt(s,a)} \{s \stackrel{a@?t[P_i]}{\longrightarrow} s_i\}. &
\end{cases}
\end{array}
$$

□

Intuitively, $out(s,a) \subseteq E$ is the set of all outgoing transitions from the state s with the action a in M, and $djout(out(s,a))$ is a recursive function that transforms the transition relations of $out(s,a)$ into ones that any two transition conditions P_i and P_j in $djout(out(s,a))$ is either disjoint or exactly the same. Note that in Definition 11, if $|E'| \geq 2$, then the function $djout(E')$ computes the set of transition relations using the recursively computed value of $djout(E' \setminus \{s \stackrel{a@?t[Q]}{\longrightarrow} s'\})$, denoted by $\bigcup_{i \in \{1,\ldots,l\} \subseteq Nxt(s,a)} \{s \stackrel{a@?t[P_i]}{\longrightarrow} s_i\}$.

Formally, the following proposition states that the transformation $djout(out(s,a))$ is correct.

Proposition 2 *The return value of the function* $djout(E') = \bigcup_{k \in \{1,\ldots,l\}} \{s \stackrel{a@?t[P_k]}{\longrightarrow} s_k\}$ *satisfies the following property:*

$$
\begin{array}{c}
\forall i, j \in \{1, \ldots, l\}, \forall t \in \mathbf{R}^+, \\
[\neg(P_i(t) \wedge P_j(t)) \vee (P_i(t) \Leftrightarrow P_j(t))]
\end{array}
$$

(Proof) Routine by induction w.r.t. the size of E'. □

Example 2 *The resultant nondeterministic PTIA after applying dj() to* M_{ex} *is depicted in Fig. 6. We get a deterministic PTIA* M''_{ex} *shown in Fig. 7 by applying traditional subset construction. As in Fig. 8, the concrete model corresponds to* M''_{ex} *has no false paths, where* $t_1 \models P_1 \wedge \neg P_2$, $t_2 \models \neg P_1 \wedge P_2$, $t_3 \models P_1 \wedge P_2$, $t_4 \models Q$, *and* $t_5 \models Q$. □

Formally, the following proposition holds for the transformation $dj(M)$.

Proposition 3 *For any PTIA* M *and any value-assignment* $\sigma \in Val$, M *and* $dj(M)$ *are semantically equivalent under* σ, *that is, the corresponding timed LTSs of* M *and* $dj(M)$ *w.r.t.* σ *are exactly the same.* □

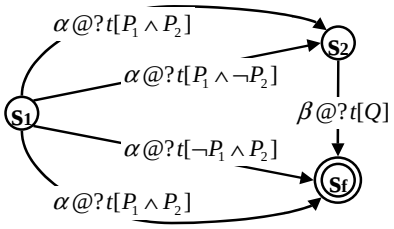

Figure 6: Applying Transformation $dj()$ for Fig. 4

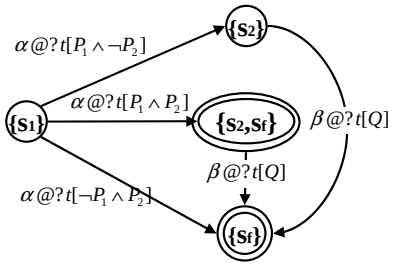

Figure 7: Applying Subset Construction for Fig. 6

The following proposition states that there is direct correspondence between nondeterministic branches in the concrete transition system of M and those in the symbolic transition system of $dj(M)$.

Proposition 4 *For any PTIA M, $\sigma \in Val$, $a \in Act$, $t \in \mathbf{R}^+$, and $s, s_1, s_2 \in S$, there are concrete transitions $(s,\sigma) \xrightarrow{t} \xrightarrow{a} (s_1,\sigma)$ and $(s,\sigma) \xrightarrow{t} \xrightarrow{a} (s_2,\sigma)$ iff there exists some $P \in Intvl(Var)$ such that $dj(M)$ has symbolic transitions $s \xrightarrow{a@?t[P]} s_1$ and $s \xrightarrow{a@?t[P]} s_2$.* □

Then, we have the following theorem.

Theorem 1 *For any finite PTIA M, let $dj(M) = \langle S, \{t\}, PVar, E_{dj}, F, s_{init}\rangle$ and $M_{NFA} = \langle S, \Sigma, E_{NFA}, F, s_{init}\rangle$ be the corresponding nondeterministic finite automaton such that $\Sigma = Act \times Intvl(Var)$ and $(s,(a,P),s') \in E_{NFA}$ iff $(s,a,P,s') \in E_{dj}$. Let $M_{DFA} = \langle S', \Sigma, E_{DFA}, F', s'_{init}\rangle$ be the corresponding deterministic finite automaton converted from M_{NFA} using the traditional subset construction method[21, Chap. 1]. Let $M' = \langle S', \{t\}, PVar, E_{DPTIA}, F', s'_{init}\rangle$ be the corresponding finite PTIA such that $(s,a,P,s') \in E_{DPTIA}$ iff $(s,(a,P),s') \in E_{DFA}$. Then M' is a deterministic finite PTIA such that for any $\sigma \in Val$, $\mathcal{L}^f_{M'}(\sigma) = \mathcal{L}^f_M(\sigma)$.* □

By Theorem 1, we can determinize any finite PTIA M by applying transformation $dj(M)$ and then performing the subset construction. Complementing accepting states F' of the resultant deterministic finite PTIA yields to M^c, a complement PTIA of M. This discussion yields the following corollary:

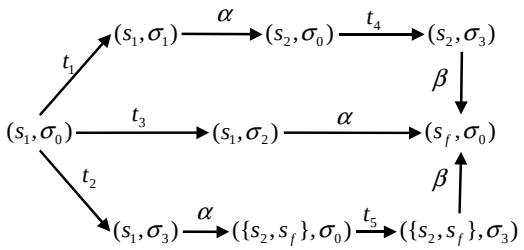

Figure 8: Concrete Model for Fig. 7

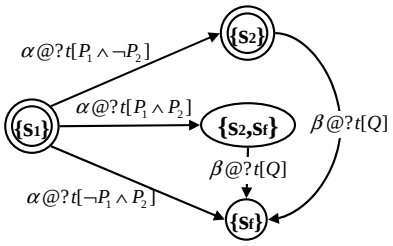

Figure 9: Complement PTIA for Fig. 7

Corollary 1 *For any finite PTIA M, there exists its complement PTIA M^c such that $\forall \sigma \in Val\ [\mathcal{L}^f_M(\sigma)^c = \mathcal{L}^f_{M^c}(\sigma)]$* □

Example 3 *The complement PTIA for Example 2 is shown in Fig. 9.* □

Determinization may cause exponential blowup of the state space. However, in the procedure shown in Sect.3.2, only a property PTIA may be determinized and then complemented. From the observation that the size of a property PTIA may be rather small compared to a model PTIA in many cases, we think that this exponential blowup does not practically cause any serious problems in verification.

5 Accelerating PTIAs

In this section, we discuss an acceleration algorithm for parametric model checking on a single PTIA.

As in the partial order reduction[1, Chap. 10] to accelerate model checking, we introduce the notion of *invisibility* w.r.t. a given property. For example, if a property is described as "The execution time from action a to action b takes at most v time units", we regard $a, b \in Act$ as visible actions and other actions as invisible (internal) action τ.

Intuitively, in order to preserve a property described by PTIA, it is sufficient to eliminate invisible actions in the model described by PTIA while preserving traces of visible actions and their execution time. To eliminate invisible actions while preserving the execution times of visible actions on M, we first map time constraints on M to actions on automata, and then apply the similar operations as those on regular expressions to sum up the time constraints of a consecutive sequence of invisible (internal) actions and a directly successing visible action, by regarding invisible actions on M as ϵ-transitions on automata. The formal definition of the transformation is as follows.

Definition 12 *For any $P, Q \in Intvl(Var)$,*

1. *Θ is a binary operator on $Intvl(Var)$ satisfying $(P \Theta Q)(t) \stackrel{\text{def}}{=} \exists t_1, t_2[P(t_1) \wedge Q(t_2) \wedge t = t_1 + t_2]$*
2. *ξ_k is a unary operator on $Intvl(Var)$ satisfying $\xi_k(P)(t) \stackrel{\text{def}}{=} \exists t'[t = k \times t' \wedge P(t')]$, where subscript k stands for a loop parameter.* □

For any PTIA $M = \langle S, \{t\}, PVar, E, F, s_{init} \rangle$, without loss of generality, we assume that for any states $s, s' \in S$, for any action $\alpha \in Act \cup \{\tau\}$, and for any transition conditions $P, Q \in Intvl(Var)$, if $s \stackrel{a@?t[P]}{\longrightarrow} s'$ and $s \stackrel{a@?t[Q]}{\longrightarrow} s'$, then $P = Q$, because if there are such transitions that $P \neq Q$, then we can merge them into $s \stackrel{a@?t[P \cup Q]}{\longrightarrow} s'$ while preserving the semantics.

In the sequel, we formally define a transformation $Accel(M, s_{rip})$ of a PTIA M to reduce the number of states of M by using the similar algorithm to derive a regular expression from a finite automaton described in Ref. [21, Chap. 1].

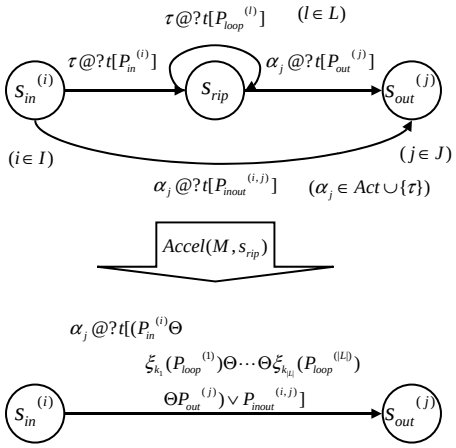

Figure 10: Illustration of $Accel(M, s_{rip})$

Definition 13 *For any PTIA $M = \langle S, \{t\}, PVar, E, F, s_{init}\rangle$ and any state $s_{rip} \in S$, we define the following subsets of transitions of M w.r.t. s_{rip}:*

$$
\begin{aligned}
IN_{s_{rip}} &\stackrel{\text{def}}{=} \{s_{in}^{(i)} \stackrel{\tau@?t[P_{in}^{(i)}]}{\longrightarrow} s_{rip} | i \in I, s_{in}^{(i)} \neq s_{rip}\},\\
OUT_{s_{rip}} &\stackrel{\text{def}}{=} \{s_{rip} \stackrel{\alpha_j@?t[P_{out}^{(j)}]}{\longrightarrow} s_{out}^{(j)} | j \in J, s_{out}^{(j)} \neq s_{rip},\\
&\qquad\qquad \alpha_j \in Act \cup \{\tau\}\},\\
LOOP_{s_{rip}} &\stackrel{\text{def}}{=} \{s_{rip} \stackrel{\tau@?[P_{loop}^{(l)}]}{\longrightarrow} s_{rip} | l \in L\},\\
INOUT_{s_{rip}} &\stackrel{\text{def}}{=} \{s_{in}^{(i)} \stackrel{\alpha_j@?t[P_{inout}^{(i,j)}]}{\longrightarrow} s_{out^{(j)}} | i \in I, j \in J\}.
\end{aligned}
$$

Then, we define an acceleration function $Accel(M, s_{rip}) \stackrel{\text{def}}{=} \langle S_{accel}, \{t\}, PVar, E_{accel}, F, s_{init}\rangle$, *where*

$$
\begin{aligned}
S_{accel} &\stackrel{\text{def}}{=} S \setminus \{s_{rip}\},\\
E_{accel} &\stackrel{\text{def}}{=}\\
&(E \setminus (IN_{s_{rip}} \cup OUT_{s_{rip}} \cup LOOP_{s_{rip}} \cup INOUT_{s_{rip}}))\\
&\cup \{s_{in}^{(i)} \stackrel{\alpha_j@?t[P_{i,j}]}{\longrightarrow} s_{out}^{(j)} | i \in I, j \in J,\\
&\quad P_{i,j} = (P_{in}^{(i)}\Theta\xi_{k_1}(P_{loop}^{(1)})\Theta\cdots\Theta\xi_{k_{|L|}}(P_{loop}^{(|L|)})\Theta P_{out}^{(j)})\\
&\qquad\qquad \vee P_{inout}^{(i,j)} \ \}.
\end{aligned}
$$

□

The transformation made by $Accel(M, s_{rip})$ is illustrated in Fig. 10. Since $Accel(M, s_{rip})$ introduces new integer parameters $k_1, \ldots, k_{|L|}$ ($|L|$ is the number of self-loops for the state s_{rip}), we refer to the resultant PTIA $M'(k_1, \ldots, k_{|L|})$ as a *PTIA with loop parameters*[4].

The following theorem is a direct consequence from the definition of operators Θ, ξ in Definition 12, and $\vee$ (a logical "or" operator on $Intvl(Var)$), and their correspondences to operations on regular expressions defined in Definition 13.

[4]Since loop parameters can be treated almost the same as other (real-valued) parameters except that their domain is restricted to non-negative integers, we omit the formal definition of a PTIA with loop parameters due to the lack of space.

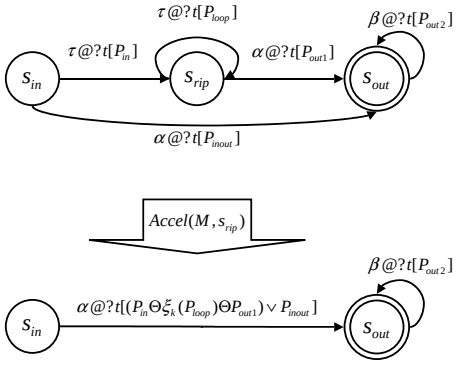

Figure 11: Example of Acceleration of PTIA

Theorem 2 *Let M be a PTIA and let $Accel(M, s_{rip}) = M'(k_1, ..., k_n)$ be a resultant PTIA with loop parameters derived by the acceleration function from M and a state s_{rip} in M. Then,*

$$\forall \sigma \in Val[\mathcal{L}_M(\sigma) = \bigcup_{i=1}^{n} \bigcup_{k_i \in \mathbf{N_0}} \mathcal{L}_{M'(k_1,...,k_n)}(\sigma)],$$

where $\mathbf{N_0}$ is the set of all non-negative integers. □

From Theorem 2, repeated application of $Accel(M, s_{rip})$ for a PTIA M yields to an equivalent (in the sense of Theorem 2) PTIA with loop parameters $M'(k_1, \ldots, k_m)$ which does not contain any internal actions.

Example 4 *An example of acceleration of a finite PTIA is shown in Fig. 11.* □

6 Parallel Composition of PTIAs and Transformation to Single PTIA

In this section, we define a parallel composition of PTIAs and give a sufficient syntactic restriction in order that a parallel composition of PTIAs is converted into a single PTIA.

The definition of a parallel composition is similar to the traditional timed automata[4]. The communication of PTIAs in the parallel composition is defined as follows: if one PTIA is ready to execute some (observable) action a and some other PTIAs can also execute a, it waits until all the other PTIAs are ready and then execute a simultaneously. Otherwise, it executes a independently. Formally, the parallel composition of PTIAs is defined as follows:

Definition 14 *A* parallel composition *of PTIAs $M_1||M_2$, where $M_1 = \langle S_1, \{t_1\}, PVar_1, E_1, F_1, s^{(1)}_{init}\rangle$ and $M_2 = \langle S_2, \{t_2\}, PVar_2, E_2, F_2, s^{(2)}_{init}\rangle$, is defined by the following parametric timed automaton $M = \langle S, C, PVar, E, F, s_{init}\rangle$ such that $S \stackrel{\text{def}}{=} S_1 \times S_2$, $C \stackrel{\text{def}}{=} \{t_1, t_2\}$, $PVar \stackrel{\text{def}}{=} PVar_1 \cup PVar_2$, $F \stackrel{\text{def}}{=} F_1 \times F_2$, $s_{init} \stackrel{\text{def}}{=} (s^{(1)}_{init}, s^{(2)}_{init})$, and*

$$\begin{aligned}
E \stackrel{\text{def}}{=} & \{((s_1, s_2), a, P_1 \wedge P_2, \{t_1, t_2\}, (s'_1, s'_2)) | \\
& \quad a \in Act_1 \cap Act_2, \\
& \quad (s_1, a, P_1, s'_1) \in E_1, \\
& \quad (s_2, a, P_2, s'_2) \in E_2\} \\
& \cup \{((s_1, s_2), a_1, P_1, \{t_1\}, (s'_1, s_2)) | \\
& \quad a_1 \in (Act_1 \setminus Act_2) \cup \{\tau\}, \\
& \quad (s_1, a_1, P_1, s'_1) \in E_1\} \\
& \cup \{((s_1, s_2), a_2, P_2, \{t_2\}, (s_1, s'_2)) | \\
& \quad a_2 \in (Act_2 \setminus Act_1) \cup \{\tau\}, \\
& \quad (s_2, a_2, P_2, s'_2) \in E_2\},
\end{aligned}$$

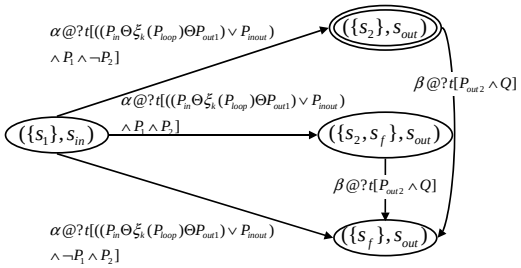

Figure 12: Parallel Composition of Figs. 9 and 11

.

where Act_1 and Act_2 denote the set of all executable actions by M_1 and M_2, respectively. □

If given two PTIAs always execute actions simultaneously, we can always convert a parallel composition of them into a single PTIA. Formally, it is defined as follows:

Definition 15 *PTIAs M_1 and M_2 are* completely synchronous *if M_1 and M_2 have no internal transitions and $Act_1 = Act_2$.* □

Theorem 3 *For any two M_1 and M_2 that are completely synchronous, there exists a single PTIA M that is semantically equivalent to the parallel composition of M_1 and M_2.*

(Proof Sketch) Let $M' = \langle S', C', PVar, E', F', s'_{init}\rangle$ be the parallel composition of M_1 and M_2. If M_1 and M_2 are completely synchronous, from Definitions 14 and 15, E' must be the following:

$$\begin{aligned} E' \;=\; \{&((s_1,s_2),a,P_1\wedge P_2,\{t_1,t_2\},(s'_1,s'_2))\mid \\ &a\in Act_1\cap Act_2 = Act, \\ &(s_1,a,P_1,\{t_1\},s'_1)\in E_1, \\ &(s_2,a,P_2,\{t_2\},s'_2)\in E_2\}. \end{aligned}$$

Then, construct a PTIA $M = \langle S,\{t\},PVar,E,F,s_{init}\rangle$ such that $S = S'$, $F = F'$, $s_{init} = s'_{init}$, and

$$\begin{aligned} E \;\stackrel{\text{def}}{=}\; \{&((s_1,s_2),a,P_1(t)\wedge P_2(t),(s'_1,s'_2))\mid \\ &((s_1,s_2),a,P_1\wedge P_2,\{t_1,t_2\},(s'_1,s'_2))\in E'\}. \end{aligned}$$

We can easily prove that M is semantically equivalent to M' since in M', the clock variables t_1 and t_2 are initially zeros and always reset to zeros simultaneously whenever some transition is executed. □

The following corollary is a direct consequence of Theorem 3 and the known results for finite automata[21, Chap. 1] and Büchi automata[1, Chap. 9]:

Corollary 2 *If M_1 and M_2 are completely synchronous finite PTIAs, then for any $\sigma\in Val$, $\mathcal{L}^f_{M_1||M_2}(\sigma) = \mathcal{L}^f_{M_1}(\sigma)\cap\mathcal{L}^f_{M_2}(\sigma)$. If M_1 and M_2 are completely synchronous Büchi PTIAs, and all the state of M_1 are accepting states, then for any $\sigma\in Val$, $\mathcal{L}_{M_1||M_2}(\sigma) = \mathcal{L}_{M_1}(\sigma)\cap\mathcal{L}_{M_2}(\sigma)$.* □

Example 5 *The parallel composition of Examples 3 and 4 are shown in Fig. 12* □

7 Extended Double Depth First Search

As described in Section 3.2, we need to extend DDFS to derive the weakest parameter condition in order that the set of accepting runs of the product Büchi PTIA is empty. We refer to the extended DDFS as EDDFS.

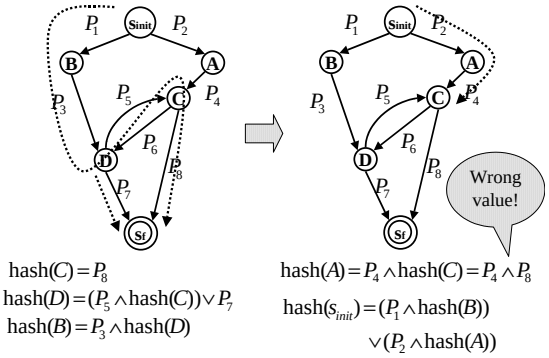

Figure 13: Parametric Double Depth First Search: Searching for Accepting States (Visiting state B first)

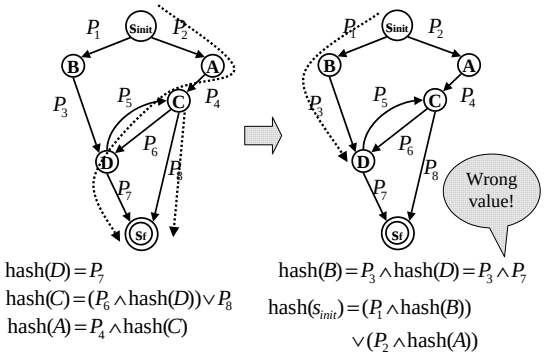

Figure 14: Parametric Double Depth First Search: Searching for Accepting States (Visiting state A first)

7.1 EDDFS Algorithm

In the original DDFS, a hash table is used to reduce the search space and a stack is used to store the path from initial state to the reached accepting states in order to find a loop including the accepting state. This is based on the idea of reuse of the visited state information. On the other hand, we cannot take advantage of these techniques to reduce search space on the product Büchi PTIAs, even if we extend the hash and stack to store the parameter condition to obtain the parameter condition for executing the already visited paths. This is because we have to perform an exhaustive enumeration of paths. The original DDFS algorithm returns only yes or no for the given verification problem. Therefore, it is enough to check the existence or non-existence of an acceptance sequence on a product Büchi automaton. On the other hand, our aim is to derive the weakest parameter condition (WPC for short) from the given verification problem. In order to do so, we have to calculate the parameter condition representing the set of all parameter values that enable some acceptance sequence of the product Büchi PTIA, which turns to require an exhaustive enumeration of acceptance sequences. Besides, we cannot store WPC to a hash table and a stack, illustrated in the following example.

Suppose we extend the stack used in DFS by additionally storing the weakest parameter condition for each state in order to reach an accepting state. Let $hash(s)$ be the stored weakest parameter condition for the state s. The result of the DFS started from s_{init} and firstly visited the state B (A) is shown in Fig. 13 (Fig. 14, respectively). In Figs. 13 and 14, each P_i represents the weakest parameter condition for executing each single transition. As illustrated in Figs. 13 and 14, the original DDFS returns the wrong results in both of the cases, because the correct values for $hash(A)$ and $hash(B)$ are $P_4 \wedge ((P_6 \wedge P_7) \vee P_8)$ and $P_3 \wedge ((P_5 \wedge P_8) \vee P_7)$, respectively.

This is because, unlike the simple reachability, WPC for each state generally depends on the paths it can be reached. The same is true for finding all the possible loops including some accepting state s_f in order to derive WPC (illustrated in Fig. 15).

Now we describe EDDFS algorithm in the following.

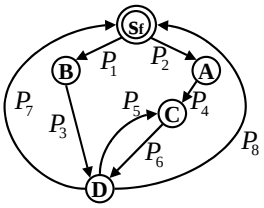

Figure 15: Parametric Double Depth First Search: Searching for Loop including Accepting State

$WPCDFS(M = \langle S, \{t\}, PVar, E, F, s_{init}\rangle)$
$= \text{return } \neg\, WPCDFS_1(s_{init}, \emptyset)$

$WPCDFS_1(s, stack)$
$= \text{let } Nxt = \{i | s \overset{a_i @?t[P_i]}{\longrightarrow} s_i, s_i \notin stack \cup \{s\}\},$
$\quad result = \bigvee_{i \in Nxt} \exists t[P_i(t) \wedge WPCDFS_1(s_i, stack \cup \{s\})]$
$\quad \text{in if } s \in F \text{ then return } result \vee WPCDFS_2(s, \emptyset, s)$
$\qquad\qquad \text{else return } result$

$WPCDFS_2(s, stack_2, s_f)$
$= \text{let } Nxt_2 = \{i | s \overset{a_i @?t[P_i]}{\longrightarrow} s_i, s_i \notin stack_2 \cup \{s, s_f\}\},$
$\quad NxtAccept = \{j | s \overset{a_j @?t[P_j]}{\longrightarrow} s_f\}$
$\quad \text{in return}$
$\qquad \bigvee_{i \in Nxt_2} \exists t[P_i(t) \wedge WPCDFS_2(s_i, stack_2 \cup \{s\}, s_f)]$
$\qquad \vee \bigvee_{j \in NxtAccept} \exists t[P_j(t)]$

Figure 16: Algorithm $WPCDFS(M)$

Definition 16 *The algorithm $WPCDFS(M)$ which takes a Büchi PTIA M as an input and returns the weakest parameter condition P is defined in Fig. 16.* □

To claim the correctness of $WPCDFS_2(s,\ stack_2,\ s_f)$, we need the following notations:

Let $M(S', F')$ denote the PTIA M whose set of all states S and accepting states F are replaced with S' and F', respectively. Let $f_{s_{in},s_{out}}(M)$ denote the PTIA M whose states $s_{in}, s_{out} \in S$ are merged into one state denoted by $[s_{in}, s_{out}]$ and all incoming (outgoing) transitions of $[s_{in}, s_{out}]$ are those of s_{in} (s_{out}, respectively). Formally $f_{s_{in},s_{out}}(M)$ is defined as follows:

Definition 17 *For any Büchi PTIA $M = \langle S,\ \{t\},\ PVar,\ E,\ F,\ s_{init}\rangle$, let $f_{s_{in},s_{out}}(M) \overset{\text{def}}{=} \langle S', \{t\}, PVar, E', F', s'_{init}\rangle$ where*

$$
\begin{array}{rcl}
S' & \overset{\text{def}}{=} & (S \setminus \{s_{in}, s_{out}\}) \cup \{[s_{in}, s_{out}]\}, \\
E' & \overset{\text{def}}{=} & \{(s, a, P, s') | (s, a, P, s') \in E, s \neq s_{out}, s' \neq s_{in}\} \\
 & & \cup \{([s_{in}, s_{out}], a, P, s') | (s_{out}, a, P, s') \in E, s' \neq s_{in}\} \\
 & & \cup \{(s, a, P, [s_{in}, s_{out}]) | (s, a, P, s_{in}) \in E, s \neq s_{out}\} \\
 & & \cup \{([s_{in}, s_{out}], a, P, [s_{in}, s_{out}]) | (s_{out}, a, P, s_{in}) \in E\} \\
F' & \overset{\text{def}}{=} & \begin{cases} (F \setminus \{s_{in}, s_{out}\}) \cup \{[s_{in}, s_{out}]\} & \\ \qquad\qquad \textit{if } F \cap \{s_{in}, s_{out}\} \neq \emptyset, \\ F \qquad\qquad \textit{otherwise.} \end{cases} \\
s'_{init} & \overset{\text{def}}{=} & \begin{cases} s_{init} & \textit{if } s_{init} \notin \{s_{in}, s_{out}\}, \\ [s_{in}, s_{out}] & \textit{otherwise.} \end{cases}
\end{array}
$$

□

The following lemma claims the correctness of $WPCDFS_2()$.

Lemma 1 *For any* $\sigma \in Val$, $\sigma \models WPCDFS_2(s,\ stack_2,\ s_f)$ *iff* $\mathcal{L}_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s,s_f\},\{s_f\}))}([s_f, s],\ \sigma) \neq \emptyset$. □

Since $f_{s_{in},s_{out}}(M) = M$ if $s_{in} = s_{out}$ and the state $[s, s]$ is equivalent to s, we have the following corollary:

Corollary 3 *For any* $\sigma \in Val$, $\sigma \models WPCDFS_2(s,\ \emptyset,\ s)$ *iff* $\mathcal{L}_{M(S,\{s\})}(s, \sigma) \neq \emptyset$. □

To claim the correctness of $WPCDFS_1(s, stack)$, we define the following function $g(M)$:

Definition 18 *For any Büchi PTIA* M, *let* $g(M) \stackrel{\text{def}}{=} \langle S, \{t\}, PVar, E', F, s_{init}\rangle$, *where* $E' \stackrel{\text{def}}{=} E \cup \{(s_f,\ \beta,\ WPCDFS_2(s_f,\ \emptyset,\ s_f),\ s_f) \mid s_f \in F\}$ *and* $\beta \in Act$. □

Proposition 5 $\mathcal{L}_M(s, \sigma) \neq \emptyset$ *iff* $\mathcal{L}_{g(M)}(s, \sigma) \neq \emptyset$.

(Proof Sketch) Since the necessity is trivial, we prove the sufficiency. Suppose $\mathcal{L}_{g(M)}(s, \sigma) \neq \emptyset$. Then, there exists $\pi \in \mathcal{L}_{g(M)}(s)$ such that σ sat. π. If there are no transitions of β, clearly $\pi \in \mathcal{L}_M(s)$. Otherwise, $\sigma \models WPCDFS_2(s_f, \emptyset, s_f)$ and thus $\mathcal{L}_{M(S,\{s_f\})}(s, \sigma) \neq \emptyset$ by Corollary 3. Therefore, there exists $\pi' = s_f \stackrel{b_1@?[Q_1]}{\longrightarrow} s_1 \cdots s_{i-1} \stackrel{b_i@?[Q_i]}{\longrightarrow} s_f \cdots$ such that σ sat.π'. Let π'' be the run obtained by replacing every occurrence of the transition of β in π with $s_f \stackrel{b_1@?[Q_1]}{\longrightarrow} s_1 \cdots s_{i-1} \stackrel{b_i@?[Q_i]}{\longrightarrow} s_f$. Then, $\pi'' \in \mathcal{L}_M(s)$ and σ sat. π''. Therefore, $\mathcal{L}_M(s, \sigma) \neq \emptyset$. □

The following lemma claims the correctness of $WPCDFS_1()$.

Lemma 2 *For any* $\sigma \in Val$, $\sigma \models WPCDFS_1(s,\ stack)$ *iff* $\mathcal{L}_{g(M)(S\setminus stack, F\setminus stack)}\ (s,\ \sigma) \neq \emptyset$. □

Since $WPCDFS(M) = \neg WPCDFS_1(s_{init}, \emptyset)$ and Proposition 5 hold, we have the following corollary:

Corollary 4 *For any* $\sigma \in Val$, $\sigma \models WPCDFS(M)$ *iff* $\mathcal{L}_M\ (\sigma) = \emptyset$. □

The proofs of Lemmas 1 and 2 are depicted in A.

From Corollary 4 and the terminating property of the general depth first search algorithm, we obtain the following theorem.

Theorem 4 *The algorithm* $WPCDFS(M)$ *always terminates and returns the weakest parameter condition for emptiness of any given Büchi PTIA* M. □

The time and space complexities of $WPCDFS(M)$ are evaluated as follows. The number of different tuples of arguments for the function $WPCDFS_2(s, stack_2, s_f)$ is bounded by $|S| \times 2^{|S|}$ (the variation of s_f's are not counted here since s_f is fixed to one value during the recursive computation of $WPCDFS_2()$). Assuming that the recursive function is implemented in a dynamic programming fashion, that is, the function call of the same tuple of arguments occurs at most once[5], the number of all recursive calls is also bounded by $|S| \times 2^{|S|}$. Thus, the time complexity of $WPCDFS_2(s, stack_2, s_f)$ is $O(|S| \times 2^{|S|})$. Similarly, the number of all recursive calls of $WPCDFS_1(s, stack)$ is bounded by $|S| \times 2^{|S|}$. Since $WPCDFS_1()$ calls $WPCDFS_2()$ when each $s \in F$ is visited, the time complexity of $WPCDFS_1(s, stack)$ is estimated as $O(|S| \times 2^{|S|}) + |F| \times O(|S| \times 2^{|S|}) = O(|S| \times |F| \times 2^{|S|})$.

7.2 Restriction of EDDFS to Finite PTIAs

As for the case that the input property M_p is given as a finite PTIA, we do not have to apply EDDFS, because we have only to enumerate all paths from the initial state to each reachable accepting state. The algorithm is, therefore, simplified to the following.

Definition 19 *The algorithm* $WPCDFS^f(M)$ *which takes a finite PTIA* M *as an input and returns the weakest parameter condition* P *is defined in Fig. 17.* □

Corollary 5 *The algorithm* $WPCDFS^f(M)$ *always terminates and returns the weakest parameter condition for emptiness of any given finite PTIA* M. □

Example 6 *The output of the algorithm* $WPCDFS^f()$ *for the product PTIA in Example 5 is* $\neg\exists t(((P_{in} \Theta\ \xi_k(P_{loop})\ \Theta P_{out1}) \vee P_{inout}) \wedge P_1 \wedge \neg P_2)$. □

[5]The return values of the function of a tuple of arguments is cached at the first time it is called, and the cached values are returned at the later times.

$WPCDFS^f(M = \langle S, \{t\}, PVar, E, F, s_{init}\rangle)$
$= \text{return } \neg\, WPCDFS_1^f(s_{init}, \emptyset)$

$WPCDFS_1^f(s, stack)$
$= \text{let } Nxt = \{i | s \overset{a_i@?t[P_i]}{\longrightarrow} s_i, s_i \notin stack \cup \{s\}\},$
$\quad result = \bigvee_{i \in Nxt} \exists t[P_i(t) \wedge WPCDFS_1(s_i, stack \cup \{s\})]$
$\quad \text{in if } s \in F \text{ then return } true$
$\qquad\qquad \text{else return } result$

Figure 17: Algorithm $WPCDFS^f(M)$

8 Conclusion

In this paper, we proposed a subclass of timed automata called PTIA, which is still useful for describing some applications such as web service specifications. We have also proposed the parametric model checking method for PTIAs by extending some traditional algorithms for finite automata and the known efficient LTL model checking algorithm called double depth first search(DDFS).

Similar to DDFS, the proposed EDDFS algorithm can be also applied to *on-the-fly model checking*[1, Chap.9], that is, a parallel composition of PTIAs can be constructed on-the-fly during EDDFS, saving the size of memory where the constructed state space is stored.

If we consider parameter variables as constants, the class of PTIAs is a proper subclass of event-clock automata[22]. The class of event-clock automata is known as a determinizable subclass of timed automata. Thus, the determinization algorithm shown in Sect. 4 is essentially equivalent to [22], although the determinization algorithm is not explicitly described in [22]. In [23], the algorithm to remove epsilon transitions of event-clock automata is proposed. However, the acceleration algorithm shown in Sect. 5 is different from that of [23]. The acceleration algorithm presented in this paper is more efficient than that of [23] since our algorithm make use of the good properties of PTIAs at the expense of its expressive power. For example, unlike [23], our acceleration does not increase the number of states. Similarly, the presented acceleration algorithm is the efficient subset of the partial order reduction for timed automata[24], since only completely synchronous parallel compositions are considered in our acceleration. Note that the methods in [22, 23, 24] do not consider parameters.

The future work is to apply some practical examples to show the usefulness of the proposed method. It is also an interesting future work to investigate the expressive power of a parallel composition of PTIAs, and, if possible, extend the proposed method to apply the case that the parallel composition of PTIAs does not fall into a single PTIA.

Acknowledgment

The authors would like to thank anonymous referees for their helpful suggestions and comments.

References

[1] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[2] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *Int. Journal of Software Tools for Technology Transfer*, vol. 1, pp. 134–152, Oct. 1997.

[3] S. Yovine, "Kronos: A verification tool for real-time systems," *Int. Journal of Software Tools for Technology Transfer*, vol. 1, no. 1/2, pp. 123–133, 1997.

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

[5] P. Matousek, "Tools for parametric verification. a comparison on a case study," in *Proc. of IEEE TC-ECBS and IFIP WG10.1 Joint Workshop on Formal Specifications of Computer-Based Systems (FSCBS2004)*, pp. 45–55, 2004.

[6] 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.

[7] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," *Int. Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-1, pp. 110–122, 1997.

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

[9] F. Wang, "Parametric analysis of computer systems," *Formal Methods in System Design*, vol. 17, no. 1, pp. 39–60, 2000.

[10] A. Annichini, A. Bouajjani, and M. Sighireanu, "TReX: A tool for reachability analysis of complex systems," in *Proc. of the 13th Int. Conf. Computer Aided Verification(CAV 2001)*, vol. 2102 of *Lecture Notes in Computer Science*, pp. 368–372, Springer, June 2001.

[11] A. Nakata and T. Higashino, "Deriving parameter conditions for periodic timed automata satisfying real-time temporal logic formulas," in *Proc. of 21st IFIP TC6/WG6.1 Int'l Conf. on Formal Techniques for Networked and Distributed Systems (FORTE2001)*, pp. 151–166, Kluwer Academic Publishers, Aug. 2001.

[12] V. Bruyere and J.-F. Raskin, "Real-time model-checking: Parameters everywhere," in *Proc. of 23rd Int. Conf. on Foundations of Software Technology and Theoretical Computer Science (FST & TCS 2003)*, vol. 2914 of *Lecture Notes in Computer Science*, pp. 100–111, Springer, 2003.

[13] T. Mori, A. Nakata, and T. Higashino, "Parametric model checking of concurrent periodic EFSMs," *Trans. of the Institute of Electronics, Information and Communication Engineers(IEICE) D-I*, vol. J86-D-I, no. 2, pp. 75–87, 2003. (In Japanese).

[14] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of web service compositions," in *Proc. of the 18th IEEE Int. Conf. on Automated Software Engineering (ASE 2003)*, p. 152, Oct. 2003.

[15] X. Fu, T. Bultan, and J. Su, "Analysis of interacting BPEL web services.," in *Proc. of the 13th World Wide Web Conference(WWW 2004)*, p. 621, May 2004.

[16] T. A. Henzinger, P. W. Kopke, and H. Wong-Toi, "The expressive power of clocks," in *Proc. of 22nd Int'l Colloq. on Automata, Languages and Programing (ICALP'95)* (Z. Fülöp and F. Gécseg, eds.), vol. 944 of *Lecture Notes in Computer Science*, pp. 417–428, EATCS, Springer-Verlag, July 1995.

[17] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Protocol Specification, Testing and Verification*, pp. 3–18, Chapman & Hall, 1995.

[18] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis, "Memory-efficient algorithms for the verification of temporal properties," *Formal Methods in System Design*, vol. 1, pp. 275–288, 1992.

[19] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, pp. 279–295, May 1997.

[20] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[21] M. Sipser, *Introduction to the Theory of Computation*. PWS Publishing Company, 1st ed., 1996.

[22] R. Alur, L. Fix, and T. A. Henzinger, "Event-clock automata: A determinizable class of timed automata," in *Proc. of Int. Conf. on Computer Aided Verification (CAV'94)*, vol. 818 of *Lecture Notes in Computer Science*, pp. 1–13, Springer-Verlag, 1994.

[23] C. Dima, "Removing silent transitions from event-clock automata," in *Proc. of Conferinta de Informatica Teoretica si Tehnologii Informatice (CITTI)*, pp. 75–81, 2000. http://www.univ-paris12.fr/lacl/dima/work/epsilon.html.

[24] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, "Partial order reductions for timed systems," in *Proc. of 9th Int. Conf. on Concurrency Theory (CONCUR'98)*, vol. 1466 of *Lecture Notes in Computer Science*, pp. 485–500, 1998.

A Full Proofs of Lemmas 1 and 2

In the proofs of Lemmas 1 and 2, we use the notion of *depth* for each state of a (Büchi) PTIA M, defined as follows:

Definition 20 *For a PTIA M and its state $s \in S$, $depth_M(s)$ is the length of the longest simple path[6] beginning with s in the transition graph of M.* □

A.1 Proof of Lemma 1

We prove that "for any $\sigma \in Val$, $\sigma \models WPCDFS_2(s, stack_2, s_f)$ iff $\mathcal{L}_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s], \sigma) \neq \emptyset$" by induction of
$depth_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s])$.

[Case of $depth_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s]) = 0$**]**

[$\Longrightarrow$] From Definition 17, the number of states in $f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f, s\}, \{s_f\}))$ must be one. If there are no outgoing transitions from $[s_f, s]$ in $f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f, s\}, \{s_f\}))$, $WPCDFS_2(s, stack_2, s_f) = false$ from Definition 16. Thus, clearly $\sigma \models WPCDFS_2(s, stack_2, s_f)$ implies
$\mathcal{L}_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s], \sigma) \neq \emptyset$.

Suppose the set of the transitions of outgoing transitions from s in M is $\{s \xrightarrow{a_i@?[P_i]} s_i | i \in I\}$. From Definition 17, the set of the transitions of outgoing transitions from $[s_f, s]$ in $f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f, s\}, \{s_f\}))$ is $\{[s_f, s] \xrightarrow{a_i@?[P_i]} s_i | i \in I\}$. Choose arbitrary $\sigma \in Val$ such that $\sigma \models WPCDFS_2(s, stack_2, s_f)$. Since $f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f, s\}, \{s_f\}))$ contains only one state, $s_i = [s_f, s]$ for any $i \in I$. From Definition 16, $Nxt_2 = \emptyset$ and thus $WPCDFS_2(s, stack_2, s_f) = \bigvee_{j\in NxtAccept} \exists t[P_j(t)]$. Therefore, there exists some $j \in NxtAccept$ and $i \in I$ such that $j = i$ and $\sigma \models \exists t[P_i(t)]$. Thus, there exist some $t_i \in \mathbf{R}^+$ and a concrete path $([s_f, s], \sigma) \xrightarrow{t_i} \xrightarrow{a_i} ([s_f, s], \sigma)$ for some $i \in I$. Since from Definition 17, the state $[s_f, s]$ is an accepting state of $f_{s_f,s}(M((S \setminus stack_2) \cup \{s, s_f\}, \{s_f\}))$, $\mathcal{L}_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s], \sigma) \neq \emptyset$. □

[$\Longleftarrow$] If there are no outgoing transitions from $[s_f, s]$ in $f_{s_f,s}(M((S \setminus stack_2)\cup\{s_f, s\}, \{s_f\}))$, from Definition 6, $\mathcal{L}_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s], \sigma) = \emptyset$. Thus, clearly $\mathcal{L}_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s], \sigma) \neq \emptyset$ implies $\sigma \models WPCDFS_2(s, stack_2, s_f)$.

Suppose the set of the transitions of outgoing transitions from $[s_f, s]$ in $f_{s_f,s}(M((S \setminus stack_2) \cup \{s, s_f\}, \{s_f\}))$ is $\{[s_f, s] \xrightarrow{a_i@?[P_i]} s_i | i \in I\}$. Since $f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f, s\}, \{s_f\}))$ contains only one state, $s_i = [s_f, s]$ for any $i \in I$. Choose arbitrary $\sigma \in Val$ such that $\mathcal{L}_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s], \sigma) \neq \emptyset$. From Definition 6 and $depth_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s]) = 0$, there is an infinite accepting run $\pi_i = [s_f, s] \xrightarrow{a_i@?[P_i]} [s_f, s] \cdots$ for some $i \in I$ such that σ sat. π_i. Thus, $\sigma \models \exists t[P_i(t)]$. From Definition 17, there exists a transition $s \xrightarrow{a_i@?[P_i]} s_f$ in M. From Definition 16, $Nxt_2 = \emptyset$ and $WPCDFS_2(s, stack_2, s_f) = \bigvee_{j\in NxtAccept} \exists t[P_j(t)]$. Since there exists some $i \in NxtAccept$, $\sigma \models WPCDFS_2(s, stack_2, s_f)$. □

[Case of $depth_{f_{s_f,s}(M((S\setminus stack_2)\cup\{s_f,s\},\{s_f\}))}([s_f, s]) = k + 1$**]**

[$\Longrightarrow$] Choose arbitrary $\sigma \in Val$ such that $\sigma \models WPCDFS_2(s, stack_2, s_f)$. From Definition 16, there are only the following two cases:

(a) $\sigma \models \bigvee_{i\in Nxt_2} \exists t[P_i(t) \wedge WPCDFS_2(s_i, stack_2 \cup \{s\}, s_f)]$ holds.

[6] A *simple path* is a path which does not contain any loops.

(b) $\sigma \models \bigvee_{j \in NxtAccept} \exists t[P_i(t)]$ holds.

For the case (b), there exist $j \in NxtAccept$ and $t_j \in \mathbf{R}^+$ such that $s \overset{a_j@?t[P_j]}{\longrightarrow} s_f$ in M and $\sigma \models P_j(t_j)$. From Definition 17, $[s_f, s] \overset{a_j@?t(P_j),s_f,s}{\longrightarrow}$ and $[s_f, s]$ is an accepting state in $f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f, s\}, \{s_f\}))$. Thus, there is an accepting run $\pi = [s_f, s] \overset{a_j@?t[P_j]}{\longrightarrow} [s_f, s] \cdots$ and σ sat. π in $f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f, s\}, \{s_f\}))$. Therefore, $\mathcal{L}_{f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f,s\},\{s_f\}))}(s, \sigma) \neq \emptyset$.

For the case (a), there exists $i \in Nxt_2$ and $t_i \in \mathbf{R}^+$ such that $s \overset{a_i@?t[P_i]}{\longrightarrow} s_i$ in M, $\sigma \models P_i(t_i)$, and $\sigma \models WPCDFS_2(s_i, stack_2 \cup \{s\}, s_f)$. From the assumption of induction, $\mathcal{L}_{f_{s_f,s_i}(M((S \setminus stack_2 \cup \{s\}) \cup \{s_f,s_i\},\{s_f\}))}(s_i, \sigma) \neq \emptyset$. Thus, there exists an accepting run $\pi_i = [s_f, s_i] \overset{b_1@?[Q_1]}{\longrightarrow} s^{(1)} \cdots s^{(l-1)} \overset{b_l@?t[Q_l]}{\longrightarrow} s^{(l)} \cdots$, such that σ sat. π_i, $s^{(l)} = [s_f, s_i]$ for an infinite number of indices l, and σ sat. π_i. Since $f_{s_f,s_i}(M((S \setminus stack_2 \cup \{s\}) \cup \{s_f, s_i\}, \{s_f\}))$ does not contain the states in $(stack_2 \cup \{s\}) \setminus \{s_f, s_i\}$, $s^{(l)} \notin (stack_2 \cup \{s\}) \setminus \{s_f, s_i\}$ for any l. Let l_i be the minimum indices l such that $s^{(l_i)} = [s_f, s_i]$ and π_i^f be the finite prefix of π_i such that $\pi_i^f = [s_f, s_i] \overset{b_1@?[Q_1]}{\longrightarrow} s^{(1)} \cdots s^{(l_i-1)} \overset{b_{l_i}@?t[Q_{l_i}]}{\longrightarrow} [s_f, s_i]$. Then, for any $l \in \{1, \ldots, l_i - 1\}$, $s^{(l)} \notin (stack_2 \cup \{s\}) \setminus \{s_f, s_i\}$. Thus, from Definition 17, there is a finite run $\pi_i^{f\prime} = s_i \overset{b_1@?[Q_1]}{\longrightarrow} s^{(1)} \cdots s^{(l-1)} \overset{b_l@?t[Q_l]}{\longrightarrow} s_f$ in M, and σ sat. $\pi_i^{f\prime}$. Then, again from Definition 17 and $\sigma \models P_i(t_i)$, $\pi^f \overset{\text{def}}{=} [s_f, s] \overset{a_i@?t[P_i]}{\longrightarrow} s_i \overset{b_1@?[Q_1]}{\longrightarrow} s^{(1)} \cdots s^{(l-1)} \overset{b_l@?t[Q_l]}{\longrightarrow} [s_f, s]$ is a finite run in $f_{s_f,s_i}(M((S \setminus stack_2 \cup \{s\}) \cup \{s_f, s_i\}, \{s_f\}))$ and σ sat. π^f. Since the state $[s, s_f]$ appeared twice in π^f, π^f can be extended to the infinite run π such that $\pi = [s_f, s] \overset{a_i@?t[P_i]}{\longrightarrow} s_i \overset{b_1@?[Q_1]}{\longrightarrow} s^{(1)} \cdots s^{(l-1)} \overset{b_l@?t[Q_l]}{\longrightarrow} [s, s_f] \overset{a_i@?t[P_i]}{\longrightarrow} s_i \cdots$, the state $[s_f, s]$ is an accepting state of $f_{s_f,s_i}(M((S \setminus stack_2 \cup \{s\}) \cup \{s_f, s_i\}, \{s_f\}))$ and appears infinitely often in π, and σ sat. π. Therefore, $\mathcal{L}_{f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f,s\},\{s_f\}))}([s_f, s], \sigma) \neq \emptyset$ holds. □

[$\Longleftarrow$] Choose arbitrary $\sigma \in Val$ such that $\mathcal{L}_{f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f,s\},\{s_f\}))}([s_f, s], \sigma) \neq \emptyset$. Then, there is an accepting run $\pi = [s_f, s] \overset{b_1@?[Q_1]}{\longrightarrow} s^{(1)} s^{(1)} \cdots s^{(l-1)} \overset{b_l@?[Q_l]}{\longrightarrow} s^{(l)} \cdots$ in $f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f, s\}, \{s_f\}))$ such that σ sat. π. Since the accepting state of $f_{s_f,s}(M((S \setminus stack_2) \cup \{s_f, s\}, \{s_f\}))$ is $[s_f, s]$ only, $s^{(l)} = [s_f, s]$ for an infinite number of l. Let l_i be the minimum indices l such that $s^{(l)} = [s_f, s]$. There are only the following two cases

(a) $l_i = 1$.

(b) $l_i \geq 2$.

For the case (a), $\pi = [s_f, s] \overset{b_1@?[Q_1]}{\longrightarrow} [s_f, s] \cdots$. From Definition 17, the transition $s \overset{b_1@?[Q_1]}{\longrightarrow} s_f$ exists in M. Thus, there exists $j \in NxtAccept$ such that $b_1 = a_j$, $Q_1 = P_j$. Since σ sat. π, $\sigma \models \exists t[P_j(t)]$. Thus, $\sigma \models \bigvee_{j \in NxtAccept} \exists t[P_j(t)]$. Therefore, from Definition 16, $\sigma \models WPCDFS_2(s, stack_2, s_f)$.

For the case (b), let π^f be the finite prefix of π such that $\pi^f = [s_f, s] \overset{b_1@?[Q_1]}{\longrightarrow} s^{(1)} \cdots s^{(l_i-1)} \overset{b_{l_i}@?[Q_{l_i}]}{\longrightarrow} [s_f, s]$. since σ sat. π, $s^{(1)} \neq [s_f, s]$, and $s^{(1)} \notin stack_2 \setminus \{s_f, s\}$, there exists some $i \in Nxt_2$ such that $b_1 = a_i$, $Q_1 = P_i$, $s^{(1)} = s_i$, and $\sigma \models \exists t[P_i(t)]$. And from Definition 17, there exists a finite run $\pi^{f\prime} = s \overset{a_i@?[P_i]}{\longrightarrow} s_i \overset{b_2@?t[Q_2]}{\longrightarrow} s^{(2)} \cdots s^{(l_i-1)} \overset{b_{l_i}@?[Q_{l_i}]}{\longrightarrow} s_f$ in M such that σ sat. $\pi^{f\prime}$. Then, let $\pi_i^f = s_i \overset{b_2@?t[Q_2]}{\longrightarrow} s^{(2)} \cdots s^{(l_i-1)} \overset{b_{l_i}@?[Q_{l_i}]}{\longrightarrow} s_f$. Again from Definition 17, there exists a finite run $\pi_i^{f\prime} = [s_f, s_i] \overset{b_2@?t[Q_2]}{\longrightarrow} s^{(2)} \cdots s^{(l_i-1)} \overset{b_{l_i}@?[Q_{l_i}]}{\longrightarrow} [s_f, s_i]$ in $f_{s_f,s_i}(M((S \setminus (stack_2 \cup \{s\}) \cup \{s_f, s_i\}, \{s_f\}))$ such that σ sat. $\pi_i^{f\prime}$. Since the state $[s_f, s_i]$ is an accepting state of $f_{s_f,s_i}(M((S \setminus (stack_2 \cup \{s\}) \cup \{s_f, s_i\}, \{s_f\}))$ and $[s_f, s_i]$ appeared twice in $\pi_i^{f\prime}$, $\pi_i^{f\prime}$ can be extended to the infinite run such that $\pi_i{}' = [s_f, s_i] \overset{b_2@?t[Q_2]}{\longrightarrow} s^{(2)} \cdots s^{(l_i-1)} \overset{b_{l_i}@?[Q_{l_i}]}{\longrightarrow} [s_f, s_i] \cdots$ and $\sigma \models \pi_i{}'$. Thus, $\mathcal{L}_{f_{s_f,s_i}(M((S \setminus (stack_2 \cup \{s\}) \cup \{s_f,s_i\},\{s_f\})))}(s_i, \sigma) \neq \emptyset$. From the assumption of induction, $\sigma \models WPCDFS_2(s_i, stack_2 \cup \{s\}, s_f)$. Therefore, $\sigma \models \bigvee_{i \in Nxt_2} \exists t[P_i(t) \wedge WPCDFS_2(s_i, stack_2 \cup \{s\}, s_f)$. From Definition 16, $\sigma \models WPCDFS_2(s, stack_2, s_f)$. □

A.2 Proof of Lemma 2

We prove that "for any $\sigma \in Val$, $\sigma \models WPCDFS_1(s, stack)$ iff $\mathcal{L}_{g(M)(S \setminus stack, F \setminus stack)}(s, \sigma) \neq \emptyset$" by induction of $depth_{g(M)(S \setminus stack, F \setminus stack)}(s)$.

[Case of $depth_{g(M)(S\setminus stack, F\setminus stack)}(s) = 0$**]**

[$\Longrightarrow$] Since $depth_{g(M)(S\setminus stack,F\setminus stack)}(s) = 0$, $g(M)(S \setminus stack, F \setminus stack)$ has only one state s. Thus, from Definition 16, $Nxt = \emptyset$ and $result = false$. If $s \notin F$, then $WPCDFS_1(s, stack) = false$. Therefore, clearly, for any $\sigma \in Val$, $\sigma \models WPCDFS_1(s,\ stack)$ implies $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}\ (s,\ \sigma) \neq \emptyset$.

Suppose $s \in F$. Then, $WPCDFS_1(s, stack) = WPCDFS_2(s, \emptyset, s)$. Choose arbitrary $\sigma \in Val$ such that $\sigma \models WPCDFS_1(s, stack)$. Then, $\sigma \models WPCDFS_2(s, \emptyset, s)$. From Definition 18, there exists a transition $s \xrightarrow{\beta@?t[WPCDFS_2(s,\emptyset,s)]} s$. Thus, $s \xrightarrow{\beta@?t[WPCDFS_2(s,\emptyset,s)]} s \cdots$ is an infinite accepting run of $g(M)(S \setminus stack, F \setminus stack)$. Therefore, $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}\ (s,\ \sigma) \neq \emptyset$. □

[$\Longleftarrow$] If $s \notin F$, $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}(s, \sigma) = \emptyset$ since $g(M)(S \setminus stack, F \setminus stack)$ has only one state s. Therefore, clearly for any $\sigma \in Val$, $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}\ (s,\ \sigma) \neq \emptyset$ implies $\sigma \models WPCDFS_1(s,\ stack)$.

Suppose $s \in F$. Choose arbitrary $\sigma \in Val$ such that $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}(s, \sigma) \neq \emptyset$. Then, there exists an infinite accepting run $s \xrightarrow{a@?t[P]} s \cdots$ such that $\sigma \models P$. If $a = \beta$, then from Definition 18, $P = WPCDFS_2(s, \emptyset, s)$. Since $WPCDFS_1(s,\ stack) = WPCDFS_2(s, \emptyset, s)$ if $s \in F$, $\sigma \models WPCDFS_1(s,\ stack)$. If $a \neq \beta$, then from Definition 18, the transition $s \xrightarrow{a@?t[P]} s$ is in M. Thus, $\mathcal{L}_{M(S\setminus stack,\{s\}\setminus stack)}(s, \sigma) \neq \emptyset$. By Corollary 3, $\sigma \models WPCDFS_2(s, \emptyset, s)$. Since $WPCDFS_1(s,\ stack) = WPCDFS_2(s, \emptyset, s)$ if $s \in F$, $\sigma \models WPCDFS_1(s,\ stack)$. □

[Case of $depth_{g(M)(S\setminus stack,F\setminus stack)}(s) = k + 1$**]**

[$\Longrightarrow$] Choose arbitrary $\sigma \in Val$ such that $\sigma \models WPCDFS_1(s,\ stack)$. Suppose $s \notin F$. Then, there exists $i \in Nxt$ and $t_i \in \mathbf{R}^+$ such that $s \xrightarrow{a_i@?t[P_i]} s_i$ in M, $\sigma \models P_i(t_i)$, and $\sigma \models WPCDFS_1(s_i,\ stack \cup \{s\})$. From the assumption of induction, $\mathcal{L}_{g(M)(S\setminus(stack\cup\{s\}),F\setminus(stack\cup\{s\}))}\ (s_i,\ \sigma) \neq \emptyset$. Thus, there exists an accepting run $\pi_i = s_i \xrightarrow{b_1@?t[Q_1]} s^{(1)} \cdots s^{(l-1)} \xrightarrow{b_l@?t[Q_l]} s^{(l)} \cdots$ in $g(M)(S\setminus(stack\cup\{s\}), F\setminus(stack\cup\{s\}))$ such that $\exists s_f \in F[s^{(l)} = s_f]$ for an infinite number of indices l and σ sat. π_i. Then, let $\pi = s \xrightarrow{a_i@?t[P_i]} s_i \xrightarrow{b_1@?t[Q_1]} s^{(1)} \cdots s^{(l-1)} \xrightarrow{b_l@?t[Q_l]} s^{(l)} \cdots$. Clearly, π is an accepting run of $g(M)(S \setminus stack, F \setminus stack)$. Therefore, $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}\ (s,\ \sigma) \neq \emptyset$.

Suppose $s \in F$. If $\sigma \not\models WPCDFS_2(s, \emptyset, s)$, then the proof is the same as the case $s \notin F$ since $\sigma \models \bigvee_{i\in Nxt} \exists t[P_i(t) \wedge WPCDFS_1(s_i, stack \cup \{s\})]$. If $\sigma \models WPCDFS_2(s, \emptyset, s)$, then from Definition 18, there exists a transition $s \xrightarrow{\beta@?t[WPCDFS_2(s,\emptyset,s)]} s$ in $g(M)(S \setminus stack, F \setminus stack)$. Thus, $s \xrightarrow{\beta@?t[WPCDFS_2(s,\emptyset,s)]} s \cdots$ is an accepting run of $g(M)(S \setminus stack, F \setminus stack)$ under σ. Therefore, $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}\ (s,\ \sigma) \neq \emptyset$. □

[$\Longleftarrow$] Choose arbitrary $\sigma \in Val$ such that $\mathcal{L}_{g(M)(S\setminus stack,F\setminus stack)}\ (s,\ \sigma) \neq \emptyset$.

Then, there exists an accepting run $\pi = s \xrightarrow{b_1@?t[Q_1]} s^{(1)} \cdots s^{(l-1)} \xrightarrow{b_l@?t[Q_l]} s^{(l)} \cdots$ of $g(M)(S \setminus stack, F \setminus stack)$ such that $\exists s_f \in F[s^{(l)} = s_f]$ for an infinite number of indices l and σ sat. π. Since the set of states of $g(M)(S \setminus stack, F \setminus stack)$ is $S \setminus stack$, $s^{(l)} \notin stack$ for any l.

Suppose $s \notin F$. Let l' be the minimum index l such that $s^{(l)} = s_f \in F$. Since $s^{(l)} \notin stack$ for any l, $s_f \in F \setminus stack$. Without loss of generality, we assume that $s^{(l)} \neq s$ for any $l \in \{1, \ldots, l'\}$, since if there exists some $l'' \in \{1, \ldots, l'\}$ such that $s^{(l'')} = s$, we can replace π with another accepting run $s \xrightarrow{b_1@?t[Q_1]} s^{(1''+1)} \cdots s^{(l-1)} \xrightarrow{b_l@?t[Q_l]} s^{(l)} \cdots$. Moreover, since $\mathcal{L}_{g(M)(S\setminus stack,\{s^{(l')}\})}(s^{(l')}, \sigma) \neq \emptyset$, by Proposition 5, $\mathcal{L}_{M(S\setminus stack,\{s^{(l')}\})}(s^{(l')}, \sigma) \neq \emptyset$. Since adding states does not affect non-emptiness, $\mathcal{L}_{M(S,\{s^{(l')}\})}(s^{(l')}, \sigma) \neq \emptyset$. By Corollary 3, $\sigma \models WPCDFS_2(s^{(l')}, \emptyset, s^{(l')})$. Therefore, without loss of generality, we assume that $s^{(l)} \neq s$ for any $l \geq l' + 1$, since if not, we can replace π with another accepting run $s \xrightarrow{b_1@?t[Q_1]} s^{(1)} \cdots s^{(l')} \xrightarrow{\beta@?t[WPCDFS_2(s^{(l')},\emptyset,s^{(l')})]} s^{(l')} \cdots$ such that σ sat. π. Thus, we safely assume that there is an accepting run $\pi = s \xrightarrow{b_1@?t[Q_1]} s^{(1)} \cdots s^{(l-1)} \xrightarrow{b_l@?t[Q_l]} s^{(l)} \cdots$ of $g(M)(S \setminus stack, F \setminus stack)$ such that for any l, $s^{(l)} \notin stack \cup \{s\}$ and σ sat. π. From Definition 16, there exists $i \in Nxt$ such that $a_i = b_1$, $Q_1 = P_i$, $s_i = s^{(1)}$, and $s \xrightarrow{a_i@?t[P_i]} s_i$ in M since $s_i \notin stack \cup \{s\}$. Moreover, there exists $t_i \in \mathbf{R}^+$ such that $\sigma \models P_i(t_i)$ since σ sat. π. Let $\pi_i = s_i \xrightarrow{b_2@?t[Q_2]} s^{(2)} \cdots s^{(l-1)} \xrightarrow{b_l@?t[Q_l]} s^{(l)} \cdots$. Then, σ sat. π_i, $s_i \notin stack \cup \{s\}$, $s^{(l)} \notin stack \cup \{s\}$ for any l, and $s_l = s_f \in F \setminus stack$ for an infinite number of indices l. Thus, π_i is an accepting run of $g(M)(S \setminus (stack \cup \{s\}), F \setminus (stack \cup \{s\}))$. Therefore, $\mathcal{L}_{g(M)(S\setminus(stack\cup\{s\}),F\setminus(stack\cup\{s\}))}(s_i) \neq \emptyset$. From the assumption of induction, $\sigma \models WPCDFS_1(s_i,\ stack \cup \{s\})$. Since $s \xrightarrow{a_i@?t[P_i]} s_i$ in M and $\sigma \models P_i(t_i)$, $\sigma \models \exists t[P_i \wedge WPCDFS_1(s_i,\ stack \cup \{s\})]$ for $i \in Nxt$. Therefore, from Definition 16, $\sigma \models WPCDFS_1(s,\ stack)$.

Suppose $s \in F$. If $s_f \neq s$ then the proof is the same as the case $s \notin F$. Otherwise, $\mathcal{L}_{g(M)(S \setminus stack, \{s\})}(s, \sigma) \neq \emptyset$. By Proposition 5, $\mathcal{L}_{M(S \setminus stack, \{s\})}(s, \sigma) \neq \emptyset$. Since adding states does not affect non-emptiness, $\mathcal{L}_{M(S, \{s\})}(s, \sigma) \neq \emptyset$. By Corollary 3, $\sigma \models \mathit{WPCDFS}_2(s, \emptyset, s)$. Therefore, from Definition 16, $\sigma \models \mathit{WPCDFS}_1(s, \mathit{stack})$. □