A Global Timed Bisimulation Preserving Abstraction for Parametric Time-Interval Automata

Akio Nakata, Tadaaki Tanimoto, Suguru Sasaki, Teruo Higashino
Department of Information Networking,
Graduate School of Information Science and Technology,
Osaka University, Suita, Osaka 565-0871, Japan

Abstract

In the development of real-time (communicating) hardware or embedded-software systems, it is frequently the case that we want to refine/optimize the system's internal behavior while preserving the external timed I/O behavior (that is, the interface protocol). In such a design refinement, modification of the systems' internal branching structures, as well as re-scheduling of internal actions, may frequently occur. Our goal is, then, to ensure that such branch optimization and re-scheduling of internal actions preserve the systems' external timed behavior, which is typically formalized by the notion of (timed) testing equivalence since it is less sensitive to the difference of internal branching structures than (timed) weak bisimulation. In order to know the degree of freedom of such re-scheduling, parametric analysis is useful. The model suitable for such an analysis is a parametric time-interval automaton(PTIA), which is a subset of a parametric timed automaton[1]. It has only a time interval with upper- and lower-bound parameters as a relative timing constraint between consecutive actions. In this paper, at first, we propose an abstraction algorithm of PTIA which preserves global timed bisimulation[2]. Global timed bisimulation is weaker than timed weak bisimulation and a sufficient condition for timed testing equivalence. Then, we also show that after applying our algorithm, the reduced PTIA has no internal actions, and thus the problem deriving a parameter condition in order that given two models are global timed bisimilar can be reduced to the existing parametric strong bisimulation equivalence checking[3].
Keywords: parametric timed automata, equivalence checking, timed testing equivalence, global timed bisimulation, abstraction

1 Introduction

1.1 Purpose and Objective

In recent years, an effective development methodology for hardware/embedded-software with real time constraints is desired. Precise implementation of timing constraints for I/O behavior is becoming important not only in embedded systems like mobile phones but also in infrastructure systems for transportation, medicine, finance and defense. But, as described in [4], it is almost impossible to verify the timing properties of real time systems by formal methods only. In this paper, we consider the following real time system development methodology: first, a skeleton code including only I/O actions with real time requirements as comments like Esterel with pragmas [5] is given; secondly, correctness of the I/O timing behavior in the skeleton code is verified by some heuristic method, as in [4]; and finally, refinement of the skeleton code for detailed implementation is performed. In this methodology, it is important to verify the equivalence of I/O timing behavior between the initial design code and its refined code.

We also take into consideration in the verification that branch restructuring of codes is performed in the refinement of the skeleton code. Moreover, it would be useful if we put real time constraints containing parameters (e.g. upper-/lower-bounds), and derive automatically the constraint (e.g. the minimum or maximum value allowed) of parameters in which the equivalence is preserved. Such an analysis is called a parametric analysis[1, 6]. To capture the control flow with time constraints and perform a parametric analysis, we propose a parametric time-interval automaton (PTIA), which is a subset of a parametric timed automaton[1] having only a time interval with upper- and lower-bound parameters as a relative timing constraint between consecutive actions. We show that global timed bisimulation (GTB) equivalence checking for PTIAs can be reduced to existing parametric strong timed bisimulation equivalence checking, where GTB is a weakening of timed weak bisimulation, in that internal branch structures are ignored.

1.2 Related Work

There are some proposals of parametric analyses for bisimulation equivalence. For bisimulation without time, parametric strong/weak bisimulation equivalence checking algorithms on STG (Symbolic Transition Graph) and STGA (STG with Assignment) are already proposed[7, 8, 9]. For timed strong bisimulation equivalence (bisimulation equivalence where both time and all actions are considered observable), parametric equivalence checking is proposed in [3]. However, for timed weak bisimulation equivalence (bisimulation equivalence where time is considered observable and internal actions are not considered observable), as far as we know, parametric equivalence checking algorithm has not been proposed. As for research about real time software design methodology, in [5] Esterel is extended to describe software with real time constraints given as comments and then timing properties are verified by model checker.

1.3 Why Global Timed Bisimulation?

In the development of real time software, several optimizations to meet real time requirements are usually done by using a profiler. In particular, branch restructuring plays an important role in the optimization. It is true that timed weak bisimulation was proposed to determine equivalence of processes considering both time and observability [10], but as pointed out by [2], timed weak bisimulation may not be suitable for equivalence checking of real time software in the presence of optimization via branch restructuring. Therefore we employ GTB to determine equivalence of processes, since GTB is a weakening of timed weak bisimulation in that internal branch structures are ignored.

1.4 Brief Description of Proposed Method

We take a skeleton code and a refined code, and convert them to PTIAs as internal representations, where the skeleton code describes I/O behaviors and real time requirements between some of the I/O actions. Note that we also put a parametric timing constraint to each internal or I/O action which has no real time requirement. (Assigning some values to such parameters means giving some concrete scheduling of the behavior.) The refined code is designed by inserting detailed internal actions into the skeleton code and by dividing some real time constraints between I/O actions in the skeleton code into constraints between I/O and internal actions. In this translation,

first, if input actions are described in branch or loop conditions, we delete such actions from branch and loop conditions while preserving real time constraints by inserting temporal variables, and second, all branches are abstracted to nondeterministic ones. Third, we unroll all the loops under the assumption that each loop has an upper bound for the number of loop iterations. Fourth, we convert the obtained nondetermistic and loop-free programs into PTIAs whose transition graphs are DAGs(Directed Acyclic Graphs). Finally, we merge a series of internal actions between I/O actions along with control flow while preserving real time constraints. With this transformation, we can convert the original PTIA to a PTIA without internal actions, so that we can apply an existing parametric timed strong bisimulation equivalence checking algorithm to compare an initial skeleton code and a refined implementation code.

1.5 Paper Organization

In Section 2, we define the PTIA model and its operational semantics by defining a mapping from the model to a timed extension of labelled transition system (timed LTS). Section 3 describes the definition of global timed bisimulation on the timed LTS. We propose a transformation algorithm on the PTIA and prove that the transformation preserves global timed bisimulation equivalence in Section 4. In Section 5, we propose a parametric global timed bisimulation equivalence checking algorithm. Conclusions and future directions are given in Section 6.

2 Parametric Time-Interval Automata

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 $Intvl(Var)$ denote a set of formulas of the form either $e1 \leq t$, $t \leq e2$, or $e1 \leq t \wedge t \leq e2$, 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 special variable representing the elapsed time since the latest visit of the current control state.

Definition 1 *A* parametric time-interval automaton *is a tuple* $\langle S, \{t\}, PVar, E, s_{init}\rangle$*, where* S *is a finite set of* control states *(also referred to as* locations*),* $t \in Var$ *is the* clock variable*,* $PVar \subseteq Var$ *is a finite set of* parameters*,* $E \subseteq S \times (Act \cup \{\tau\}) \times Intvl(PVar) \times S$ *is a transition relation,* 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* $s_i \overset{a@?t[P]}{\longrightarrow} s_j$ *if* $(s_i, a, P, s_j) \in E$. □

Informally, a transition $s_i \overset{a@?[P]}{\longrightarrow} s_j$ means that the action a can be executed from s_i when the values of both the clock variable t and parameters satisfy the formula P (called a *guard condition*), and after executed, the state moves into s_j and the clock variable t is reset to zero. In any state s, the value of the clock variable t increases continuously, representing the time passage.

Formal semantics of parametric time-interval automata is similar to general parametric timed automata, which is defined as follows. The values of clocks and parameters are given by a function $\sigma : (\{t\} \cup PVar) \mapsto \mathbf{R}$. 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 $P \in Intvl(Var)$ 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. The semantic model is a *timed labelled transition system (timed LTS)*, which is defined as follows. A state of a timed LTS is a concrete state in CS. A transition of a timed LTS is either a *delay-transition* or an *action-transition*. 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, a timed labelled transition system is defined as follows.

Definition 2 *A* timed labelled transition system *(a* timed LTS *for short) for a parametric time-interval automaton is a labelled transition system* $\langle CS, Act \cup \mathbf{R}^+ \cup \{\tau\}, CE, (s_{init}, \sigma_{init}[t \to 0])\rangle$*, where a set of states is* CS*, a set of labels is* $Act \cup \mathbf{R}^+ \cup \{\tau\}$*, an initial state is* $(s_{init}, \sigma_{init}[t \to 0])$*, and a 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* $(s, \sigma) \stackrel{l}{\longrightarrow} (s', \sigma')$ *if* $((s, \sigma), l, (s', \sigma')) \in CE$*):*

- $(s, \sigma) \stackrel{v}{\longrightarrow} (s, \sigma + v)$ *if* $v \in \mathbf{R}^+$,
- $(s, \sigma) \stackrel{a}{\longrightarrow} (s', \sigma[t \to 0])$ *if* $a \in Act \cup \{\tau\}$, $s \stackrel{a@?t[P]}{\longrightarrow} s'$, *and* $\sigma \models P$,

where $\sigma + v$ *and* $\sigma[t \to 0]$ *are the value-assignments derived from* σ*, which is defined as follows:*

$$\text{For } x \in PVar \cup \{t\}$$

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

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

□

3 Global Timed Bisimulation

In this section, we briefly recall the definition of global timed bisimulation (GTB) proposed in Ref. [2], as well as the definition of the traditional timed weak bisimulation[11, 10] (TWB) and its relation to the GTB.

3.1 Timed Weak Bisimulation

In this section, we will briefly give the definition of timed weak bisimulation.

Definition 3 *A* timed weak transition relation $\rightarrow_w$ *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. $\stackrel{\tau}{\longrightarrow}_w \stackrel{\text{def}}{=} (\stackrel{0}{\rightarrow})^* (\stackrel{\tau}{\rightarrow})^*)^*$
2. $(s, \sigma) \stackrel{v}{\longrightarrow}_w (s', \sigma')$ $(v \in \mathbf{R}^+)$
   $\stackrel{\text{def}}{=} \exists v_1, v_2, ..., v_n \in \mathbf{R}^+ \ [\ v = \sum_{i=1}^{n} v_i$
   $\wedge \exists s_1, \sigma_1, \sigma'_1, s_2, \sigma_2, \sigma'_2, ..., s_n, \sigma_n, \sigma'_n$
   $s.t. \ (s, \sigma) \stackrel{\tau}{\longrightarrow}_w (s_1, \sigma_1) \stackrel{v_1}{\longrightarrow} (s_1, \sigma'_1) \cdots \stackrel{\tau}{\longrightarrow}_w (s_n, \sigma_n) \stackrel{v_n}{\longrightarrow}$
   $(s_n, \sigma'_n) \stackrel{\tau}{\longrightarrow}_w (s', \sigma')\ ]$

3. $\xrightarrow{a}_w$ $(a \in Act) \stackrel{\text{def}}{=} \xrightarrow{\tau}_w \xrightarrow{a} \xrightarrow{\tau}_w$ □

By using this transition relation, timed weak bisimulation is defined as follows:

Definition 4 *A binary relation R on states of a timed LTS is a* timed weak bisimulation *if the following condition hold:*

If $(s_1, \sigma_1)R(s_2, \sigma_2)$*, then for any* $\alpha \in Act \cup \mathbf{R}^+ \cup \{\tau\}$*,*

1. $\forall\, s_1', \sigma_1'[\ (s_1, \sigma_1) \xrightarrow{\alpha}_w (s_1', \sigma_1') \Rightarrow$
   $\exists s_2', \sigma_2'\ [\ (s_2, \sigma_2) \xrightarrow{\alpha}_w (s_2', \sigma_2') \wedge (s_1', \sigma_1')R(s_2', \sigma_2')\ ]\ ]$,
   and,
2. $\forall\, s_2', \sigma_2'[\ (s_2, \sigma_2) \xrightarrow{\alpha}_w (s_2', \sigma_2') \Rightarrow$
   $\exists s_1', \sigma_1'\ [\ (s_1, \sigma_1) \xrightarrow{\alpha}_w (s_1, \sigma_1') \wedge (s_1', \sigma_1')R(s_2', \sigma_2')\ ]\ ]$

We say that states (s_1, σ_1) *and* (s_2, σ_2) *are* timed weak bisimulation equivalent, *denoted by* $(s_1, \sigma_1) \equiv_{twb} (s_2, \sigma_2)$ *if and only if there exists a timed weak bisimulation R such that* $(s_1, \sigma_1)\ R\ (s_2, \sigma_2)$*.* □

3.2 Global Timed Bisimulation

Global timed bisimulation[2] is a variant of bisimulation equivalence which consider both timing and observability. Unlike timed weak bisimulation, global timed bisimulation does not distinguish the difference of branching structures of internal transitions.

Firstly, in order to make it suitable for our formalism, we rephrase the definition of a static generalized transition relation $\longrightarrow_{gt}$ and a dynamic generalized transition relation $\xrightarrow{\alpha}_{gt}$, which are proposed in Ref. [2].

The intention of the following definition is that, from the original timed LTS, we want to construct a new timed LTS using $\longrightarrow_{gt}$ and $\xrightarrow{\alpha}_{gt}$ such that the constructed LTS contains many other possible branching structures derived from the original one, which is *practically* indistinguishable by any external observer (that is, any *realistic* tester).

Definition 5 *Let* $OGT((s, \sigma))$ *be the set of all outgoing transitions of* (s, σ)*, that is, the set of all transitions whose source state is* (s, σ)*. A* static generalized transition relation $\longrightarrow_{gt}$ *(also denoted by* $\xrightarrow{\epsilon}_{gt}$*) and a* dynamic generalized transition relation $\xrightarrow{\alpha}_{gt}$ *on states of a timed LTS are defined as follows:*

1. *preserving the timed weak transition relation*
   (a) *If* $(s, \sigma) \xrightarrow{\tau}_w (s', \sigma')$*, then* $(s, \sigma) \longrightarrow_{gt} (s', \sigma')$*.*
   (b) *For each* $\alpha \in Act \cup \mathbf{R}^+$*, if* $(s, \sigma) \xrightarrow{\alpha}_w (s', \sigma')$*, then* $(s, \sigma) \xrightarrow{\alpha}_{gt} (s', \sigma')$*.*
2. *simultaneous choice of internal transitions (see Fig. 1)*
   If $OGT((s, \sigma)) = \bigcup_{i \in I}\{(s, \sigma) \xrightarrow{\alpha_i} (s_{\alpha_i,i}, \sigma_{\alpha_i,i})\} \cup \bigcup_{j \in J}\{(s, \sigma) \xrightarrow{\tau} (s_j, \sigma_j)\}$ $(\alpha_i \in Act \cup \mathbf{R}^+)$*, then* $(s, \sigma) \longrightarrow_{gt} (s_{new}, \sigma)$ *and for all* $j \in J$*,* $(s_{new}, \sigma) \xrightarrow{\tau}_{gt} (s_j, \sigma_j)$*, where* s_{new} *is a newly introduced control state.*
3. *forward resolution of nondeterminism (see Fig. 2)*
   If $OGT((s, \sigma)) = \bigcup_{i \in I}\{(s, \sigma) \xrightarrow{\alpha_i} (s_{\alpha_i,i}, \sigma_{\alpha_i,i})\} \cup \bigcup_{j \in J}\{(s, \sigma) \xrightarrow{a} (s_j, \sigma_j)\}$ $(\alpha_i \in Act \cup \mathbf{R}^+$ *and* $a \in Act)$*, then for each* $j \in J$*,* $(s, \sigma) \longrightarrow_{gt} (s_{new}^{(j)}, \sigma)$*,* $(s_{new}^{(j)}, \sigma) \xrightarrow{a}_{gt}$

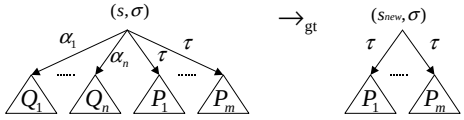

Figure 1: Simultaneous Choice of Internal Actions

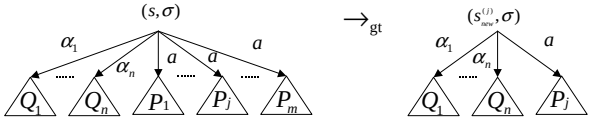

Figure 2: Forward Resolution of Nondeterminism

(s_j, σ_j), *and* $(s^{(j)}_{new}, \sigma) \xrightarrow{\alpha_i}_{gt} (s_{\alpha_i,i}, \sigma_{\alpha_i,i})$ *for any* $i \in I$, *where* $s^{(j)}_{new}$ *is a newly introduced control state depending on the choice of* j.

4. *forward execution of an internal transition (see Fig. 3)*

   If $OGT((s,\sigma)) = \bigcup_{i\in I}\{(s,\sigma) \xrightarrow{\alpha_i} (s_{\alpha_i,i}, \sigma_{\alpha_i,i})\} \cup \{(s,\sigma) \xrightarrow{\beta}, (s_\beta, \sigma_\beta)\}$ *(*$\alpha_i \in Act \cup \mathbf{R}^+$ *and* $\beta \in Act \cup \mathbf{R}^+$*) and if* $(s_\beta, \sigma_\beta) \longrightarrow_{gt} (s'_\beta, \sigma'_\beta)$ *for some* β, *then* $(s,\sigma) \longrightarrow_{gt} (s^{(\beta)}_{new}, \sigma)$, $(s^{(\beta)}_{new}, \sigma) \xrightarrow{\beta}_{gt} (s'_\beta, \sigma'_\beta)$ *and for any* $i \in I$, $(s^{(\beta)}_{new}, \sigma) \xrightarrow{\alpha_i}_{gt} (s_{\alpha_i,i}, \sigma_{\alpha_i,i})$, *where* $s^{(\beta)}_{new}$ *is a newly introduced control state depending on the choice of* β.

5. *simultaneous execution of an observable action (see Fig. 4)*

   If $OGT((s,\sigma)) = \bigcup_{i\in I}\{(s,\sigma) \xrightarrow{a} (s_i, \sigma_i)\} \cup \bigcup_{j\in J}\{(s,\sigma) \xrightarrow{\alpha_j} (s_{\alpha_j,j}, \sigma_{\alpha_j,j})\}$ *(*$a \in Act$ *and* $\alpha_j \in Act \cup \mathbf{R}^+$*), then for any non-empty subset* $I' \subseteq I$, $(s,\sigma) \xrightarrow{a}_{gt} (s^{(I')}_{new}, \sigma)$ *and for any* $i' \in I'$ $(s^{(I')}_{new}, \sigma) \xrightarrow{\tau}_{gt} (s_{i'}, \sigma_{i'})$, *where* $s^{(I')}_{new}$ *is a newly introduced control state depending on the choice of* I'.

6. *synchronized forward time passage of internal transitions (see Fig. 5)*

   Let $\Longrightarrow_{gt} \stackrel{\text{def}}{=} (\longrightarrow_{gt})^*$, $\stackrel{a}{\Longrightarrow}_{gt} \stackrel{\text{def}}{=} \Longrightarrow_{gt} \xrightarrow{a}_{gt} \Longrightarrow_{gt}$ *for* $a \in Act$, *and* $\stackrel{v}{\Longrightarrow}_{gt} \stackrel{\text{def}}{=} \xrightarrow{v_1}_{gt} \cdots \xrightarrow{v_n}_{gt}$ *for* $v \in \mathbf{R}^+$ *if there exists some* $v_1, \ldots, v_n$ *such that* $v = v_1 + \cdots + v_n$.

   If $OGT((s,\sigma)) = \bigcup_{i\in I}\{(s,\sigma) \xrightarrow{\alpha_i} (s_{\alpha_i,i}, \sigma_{\alpha_i,i})\} \cup \bigcup_{j\in J}\{(s,\sigma) \xrightarrow{\tau} (s_j, \sigma_j)\}$ *(*$\alpha_i \in Act \cup \mathbf{R}^+$*), and if for* $v \in \mathbf{R}^+$, $(s_j, \sigma_j) \stackrel{v}{\Longrightarrow}_{gt} (s^{(v)}_j, \sigma^{(v)}_j)$ *for all* $j \in J$, *then* $(s,\sigma) \xrightarrow{v}_{gt} (s^{(v)}_{new}, \sigma^{(v)}_{new})$ *and* $(s^{(v)}_{new}, \sigma^{(v)}_{new}) \xrightarrow{\tau}_{gt} (s^{(v)}_j, \sigma^{(v)}_j)$ *for all* $j \in J$, *where*

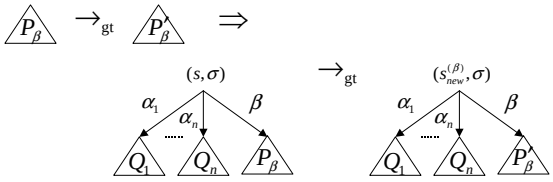

Figure 3: Forward Execution of an Internal Transition

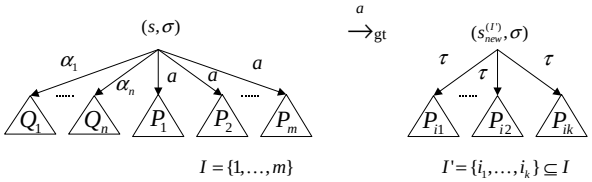

Figure 4: Simultaneous Execution of an Observable Action

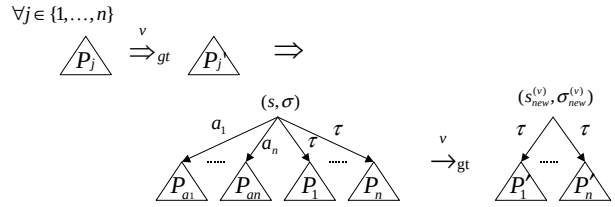

Figure 5: Synchronized Forward Time Passage of Internal Transitions

$s^{(v)}_{new}$ is a newly introduced control state depending on the time value v, and $\sigma^{(v)}_{new}$ is a value assignment depending on v, s, and σ such that $(s,\sigma) \xrightarrow{v} (s, \sigma^{(v)}_{new})$. □

Using these transition relations, global timed bisimulation is defined as follows.

Definition 6 *A binary relation R on states of a timed LTS is a* global timed bisimulation *if the following condition holds:*

If $(s_1,\sigma_1)R(s_2,\sigma_2)$, then for any $\alpha \in Act \cup \mathbf{R}^+ \cup \{\epsilon\}$,

1. $\forall\, s'_1, \sigma'_1[\ (s_1,\sigma_1) \xrightarrow{\alpha}_{gt} (s'_1,\sigma'_1)\ \Rightarrow$
   $\exists s'_2, \sigma'_2\ [\ (s_2,\sigma_2) \overset{\alpha}{\Longrightarrow}_{gt} (s'_2,\sigma'_2) \wedge (s'_1,\sigma'_1)R(s'_2,\sigma'_2)\ ]\ ]$,
   and,
2. $\forall\, s'_2, \sigma'_2[\ (s_2,\sigma_2) \xrightarrow{\alpha}_{gt} (s'_2,\sigma'_2)\ \Rightarrow$
   $\exists s'_1, \sigma'_1\ [\ (s_1,\sigma_1) \overset{\alpha}{\Longrightarrow}_{gt} (s_1,\sigma'_1) \wedge (s'_1,\sigma'_1)R(s'_2,\sigma'_2)\ ]\ ]$

We say that states (s_1,σ_1) and (s_2,σ_2) are global timed bisimulation equivalent, *denoted by $(s_1,\sigma_1) \equiv_{gtb} (s_2,\sigma_2)$ if and only if there exists a global timed bisimulation R such that $(s_1,\sigma_1)\ R\ (s_2,\sigma_2)$.* □

As concerned to the relationship between timed weak bisimulation and global timed bisimulation, the following proposition holds[2].

Proposition 1 *For any two states (s_1,σ_1) and (s_2,σ_2) of a timed LTS, if $(s_1,\sigma_1) \equiv_{twb} (s_2,\sigma_2)$, then $(s_1,\sigma_1) \equiv_{gtb} (s_2,\sigma_2)$.* □

4 Abstraction Algorithm

In this section, we propose some abstraction rules to eliminate internal actions of the PTIA, and show that their rules preserve global timed bisimulation equivalence.

In the following, firstly, we describe some restrictions on PTIAs which ensure the correctness of the proposed abstraction rules. Then, we describe the key idea and the details of the proposed abstraction rules. Finally, we show that the abstraction rules preserve global timed bisimulation equivalence.

4.1 Restrictions for Parametric Time-Interval Automata

In order to apply our proposed abstraction rules, we impose the following restrictions [RLoopFree], [RInitStability], and [RObsBounded] for an input PTIA M .

[RLoopFree] M contains no loops, that is, the transition graph of M is a DAG[1].

[RInitStability] The initial state s_{init} of M must be either a *stable* state, or reachable to a stable state by deterministic internal transitions (i.e. there are no branch from initial state to a stable state). Here, a *stable* state is a state whose every outgoing transition is observable.

[RObsBounded] Any internal transition contained in M is *observably bounded*, that is, for any execution path π of M, there exists an extension π' of the path π in that any internal transition is appeared between some observable transitions.

Formally, an internal transition $s \xrightarrow{\tau@t[P]} s'$ contained in M is *observably bounded* if for any transition sequence $s_{init} \xrightarrow{\alpha_1@?t[P_1]} s_1 \cdots s_{k-1} \xrightarrow{\alpha_k@?t[P_k]} s \xrightarrow{\tau@t[P]} s'$ ($\alpha_1, \ldots, \alpha_k \in Act \cup \{\tau\}$, $P_1, \ldots, P_k, P \in Intvl(Var)$) of M, there exists some $i \in \{1, \ldots, k\}$ such that $\alpha_i \in Act$, and for any transition sequence $s \xrightarrow{\tau@t[P]} s' \xrightarrow{\beta_1@?t[Q_1]} s_1 \cdots s_{k-1} \xrightarrow{\beta_k@?t[Q_k]} s_k$ ($\beta_1, \ldots, \beta_k \in Act \cup \{\tau\}$, $Q_1, \ldots, Q_k \in Intvl(Var)$), there is an extension of the sequence $s_k \xrightarrow{\beta_{k+1}@?t[Q_{k+1}]} s_{k+1} \cdots s_{k+l-1} \xrightarrow{\beta_{k+l}@?t[Q_{k+l}]} s_{k+l}$ ($\beta_{k+1}, \ldots, \beta_{k+l} \in Act \cup \{\tau\}$, $Q_{k+1}, \ldots, Q_{k+l} \in Intvl(Var)$) and some $i \in \{1, \ldots, k+l\}$ such that $\beta_i \in Act$.

4.2 Abstraction Rules for Parametric Time-Interval Automata

For any internal transition that is directly preceding by some observable transition, we can eliminate the internal transition based on the following principles:

1. The internal nondeterminism caused by the internal transition can be converted into the corresponding nondetermistic choice of the directly preceding observable transition, just the same as the classical equational theory of testing equivalence[12].

2. On the contrary, the time passage caused by the internal transition can be moved into those of the directly succeeding transitions which are either internal or observable.

This is the key idea for preserving testing equivalence considering time. Since any internal action is observably bounded, by the restriction [RObsBounded], the sequence of internal actions can be completely eliminated from the beginning internal action (which is directly preceding by some observable action) to the ending one (which is directly succeeding by some observable action).

However, if we want to transform some subgraph of the entire transition graph of M, we must ensure that such a subgraph transformation preserves equivalence of

[1]For any loop contained in input programs, we assume that there exists a maximum number k of iteration of the loop. Note that all branchs in the input programs are abstracted to nondeterministic ones

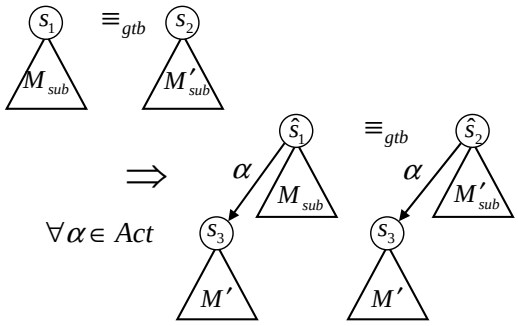

Figure 6: Congruence Property w.r.t. GTB

the entire transition graph. To make the discussion simple, we impose the restriction [RLoopFree], we have only to consider the case that the transition graph is a DAG. Furthermore, we prove that if the context of the subgraph under transformation is in some form, such subgraph transformation preserves equivalence of the entire transition system M. To ensure that all internal transition can be eliminated by such a context-sensitive transformation, we need the restriction [RInitStability].

The proposed abstraction rules are the followings:

1. abstraction for sequential structures
2. abstraction for branching structures

The details are described in the following sections.

4.2.1 Context Sensitivity of Abstraction Rules

Consider that some subgraph M_{sub} of the entire transition graph of PTIA M is replaced into some equivalent subgraph M'_{sub}. Such a transformation does not always preserve equivalence of the entire transition graph. We will show that if M_{sub} appears the context shown in Fig 6, then the subgraph transformation from M_{sub} to M'_{sub} preserves global timed bisimulation equivalence.

Theorem 1 (Congruence Property for Subgraph Transformation of PTIA w.r.t GTB) *Let $M = \langle S, \{t\}, PVar, E, s_{init}\rangle$ be a PTIA, and $s_1, s_2 \in S$ .Let $\hat{s_1}$ and $\hat{s_2}$ be new states obtained by adding the same observable branch from s_1 and s_2, that is, $\hat{s_1} \overset{\alpha@?t[P]}{\longrightarrow} s_3$ and $\hat{s_2} \overset{\alpha@?t[P]}{\longrightarrow} s_3$ for $\alpha \in Act$, $P \in Intvl(Var)$, $\hat{s_1} \overset{\beta@?t[Q]}{\longrightarrow} s'_1$ if $s_1 \overset{\beta@?t[P]}{\longrightarrow} s'_1$ for $\beta \in Act \cup \{\tau\}$, $Q \in Intvl(Var)$, and $\hat{s_2} \overset{\beta@?t[Q]}{\longrightarrow} s'_2$ if $s_2 \overset{\beta@?t[P]}{\longrightarrow} s'_2$ for $\beta \in Act \cup \{\tau\}$, $Q \in Intvl(Var)$.*

Then, for any $\sigma_1, \sigma_2 \in Val$, $(\hat{s_1}, \sigma_1) \equiv_{gtb} (\hat{s_2}, \sigma_2)$ if $(s_1, \sigma_1) \equiv_{gtb} (s_2, \sigma_2)$.

Proof. *(sketch) Consider the executable transitions from $(\hat{s_1}, \sigma_1)$ and $(\hat{s_2}, \sigma_2)$. If some observable transition $\beta \neq \alpha$ is executed (after some time consumption) from $(\hat{s_1}, \sigma_1)$, the behavior of M_{sub} must be chosen (otherwise, α must be observed). From the assumption $(s_1, \sigma_1) \equiv_{gtb} (s_2, \sigma_2)$, and the definition of $\hat{s_1}$ and $\hat{s_2}$, β is also executable from $(\hat{s_2}, \sigma_2)$ (after the same time consumption) and the behavior of M'_{sub} is chosen. The following behaviors are obviously global timed equivalent since $(s_1, \sigma_1) \equiv_{gtb} (s_2, \sigma_2)$. The same is true if $(\hat{s_1}, \sigma_1)$ and $(\hat{s_2}, \sigma_2)$ are exchanged.*

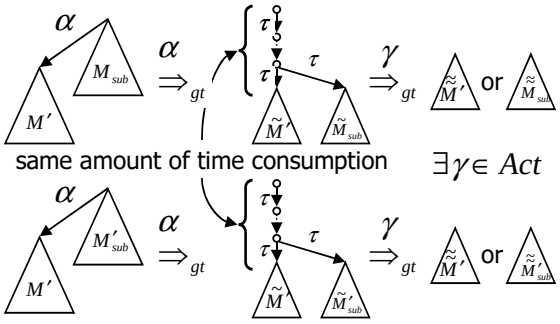

Figure 7: Illustration of Proof of Theorem 1

Next, consider that α is executed from $(\hat{s_1}, \sigma_1)$. If the behavior of M' is chosen, obviously $(\hat{s_2}, \sigma_2)$ also choose M' by the similar discussion above, and vice versa.

Finally, suppose that choice was not made after the execution of α. As illustrated in Fig. 7, we can derive a generalized transition relation for such a transition from $(\hat{s_1}, \sigma_1)$ by applying rules 2,4,5,6 of Definition 5. From the assumption $(s_1, \sigma_1) \equiv_{gtb} (s_2, \sigma_2)$, we can also derive the generalized transition relation from $(\hat{s_2}, \sigma_2)$ which makes weakly bisimilar to $(\hat{s_1}, \sigma_1)$, and vice versa. Once some observable action γ is executed, obviously they can simulate the following behaviors each other. From the discussions above, we can prove the theorem.

4.2.2 Abstraction for Sequential Structures

The abstraction for sequential structures is illustrated in Fig. 8. In the left half of Fig 8, the transition $s_2 \overset{\tau@?t[x_2 \le t \le y_2]}{\longrightarrow} s_3$ is internal and its directly preceding transition $s_1 \overset{\alpha@?t[x_1 \le t \le y_1]}{\longrightarrow} s_2$ is observable. Its sibling $s_2 \overset{\beta@?t[x_3 \le t \le y_3]}{\longrightarrow} s_4$ and its directly succeeding transition $s_3 \overset{\gamma@?t[x_4 \le t \le y_4]}{\longrightarrow} s_5$ may be either internal or observable. In this case, we eliminate the internal transition $s_2 \overset{\tau@?t[x_2 \le t \le y_2]}{\longrightarrow} s_3$ as the right half of Fig 8. In this abstraction, the nondetermistic behavior of the eliminated internal transition is converted into the nondeterministic choice of the directly preceding observable transitions $s_1 \overset{\alpha@?t[x_1 \le t \le y_1]}{\longrightarrow} s_2$ and $s_1 \overset{\alpha@?t[x_1 \le t \le y_1]}{\longrightarrow} s_3$. On the other hand, the time passage of the eliminated internal transition is merged into the directly succeeding transitions $s_2 \overset{\gamma@?t[x_2+x_4 \le t \le y_2+y_4]}{\longrightarrow} s_5$ and $s_3 \overset{\gamma@?t[x_2+x_4 \le t \le y_2+y_4]}{\longrightarrow} s_5$.

Formally, the operation for merging two time intervals is defined as follows:

Definition 7 *Θ is a binary operator on $Intvl(Var)$ such that for any $P, Q \in Intvl(Var)$, $(P \Theta Q)(t) \stackrel{\text{def}}{=} \exists t_1, t_2[P(t_1) \wedge Q(t_2) \wedge t = t_1 + t_2]$.* □

Then, the following lemma holds:

Lemma 4.1 *Let $\alpha \in Act$ and $\beta, \gamma \in Act \cup \{\tau\}$. Let M and M' be PTIAs whose sets of transtions are $\{ s_1 \overset{\alpha@?t[P_1]}{\longrightarrow} s_2,\ s_2 \overset{\tau@?t[P_2]}{\longrightarrow} s_3,\ s_2 \overset{\beta@?t[P_3]}{\longrightarrow} s_4,\ s_3 \overset{\gamma@?t[P_4]}{\longrightarrow} s_5 \}$ and $\{ s'_1 \overset{\alpha@?t[P_1]}{\longrightarrow} s'_2,\ s'_1 \overset{\alpha@?t[P_1]}{\longrightarrow} s'_3,\ s'_2 \overset{\beta@?t[P_3]}{\longrightarrow} s_4,\ s'_2 \overset{\gamma@?t[P_2 \Theta P_4]}{\longrightarrow} s_5\ s'_3 \overset{\gamma@?t[P_2 \Theta P_4]}{\longrightarrow} s_5 \}$, respectively. Then, for any assignment $\sigma \in Val$, $(s_1, \sigma) \equiv_{gtb} (s'_1, \sigma)$.*

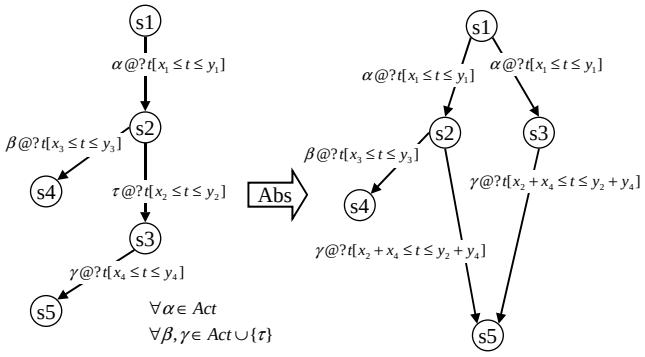

Figure 8: Abstraction for Sequential Structures

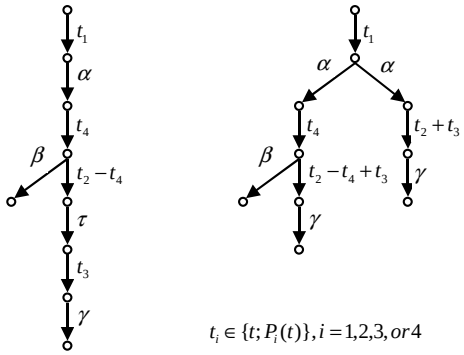

Figure 9: Semantic Models for M and M' of Lemma 4.1

Proof. *(sketch) The semantic model of M and M′ are illustrated in Fig. 9. In Fig. 9, time transitions are represented symbolically by the variables t_i ($i \in \{1,2,3,4\}$). Each t_i can be any possible value in the set $\{t|P_i(t)\}$. For simplicity, we assume that $t_4 \le t_2$. The case $t_4 < t_2$ is similar. From Definition 5, the transition graphs based on the generalized transition relation of M′ is constructed as follows. First, rule 5 of Definition 5 is applied (Fig. 10). Next, rule 6 is applied twice (Fig. 11) . It is easy to see that the obtained generalized transition graphs of M in Fig. 9 and M′ in Fig. 11 are weakly bisimilar. Therefore, from Definition 6, (s_1, σ) and (s'_1, σ) are global timed bisimilar.*

More general case is that there are some outgoing/incoming transitions on s_1, s_2 and s_3, as shown in Fig 12. In this case, all the source states of the incoming transition of s_2 must be stable in order to satisfy the congruence property in Theorem 1. In Fig 12, the outgoing transitions of the state s_1 and s_6 are observable. If this is satisfied, then all the incoming transitions of s_2 are converted into the nondeterministic choice, as shown in Fig 13. The internal transition $s_2 \overset{\tau@?t[x_5 \le t \le y_5]}{\longrightarrow} s_3$ is eliminated similarly, but since s_3 has some incoming transitions $s_8 \overset{\alpha'@?t[x_6 \le t \le y_6]}{\longrightarrow} s_3$ and $s_9 \overset{\beta'@?t[x_7 \le t \le y_7]}{\longrightarrow} s_3$, these transitions and the original outgoing transitions $s_3 \overset{\gamma'@?t[x_8 \le t \le y_8]}{\longrightarrow} s_5$ and $s_3 \overset{\delta'@?t[x_9 \le t \le y_9]}{\longrightarrow} s_{10}$ of the state s_3 are preserved. Then, the time passage of the transition $s_2 \overset{\tau@?t[x_5 \le t \le y_5]}{\longrightarrow}$

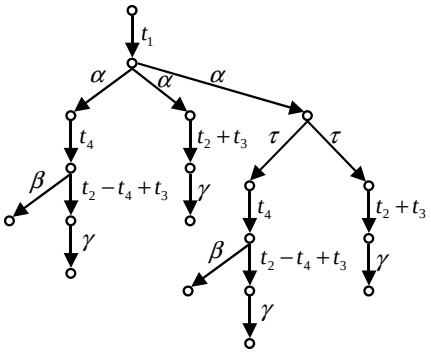

Figure 10: Generalized Transition Relation of M' derived from Fig. 9 by rule 5 of Definition 5

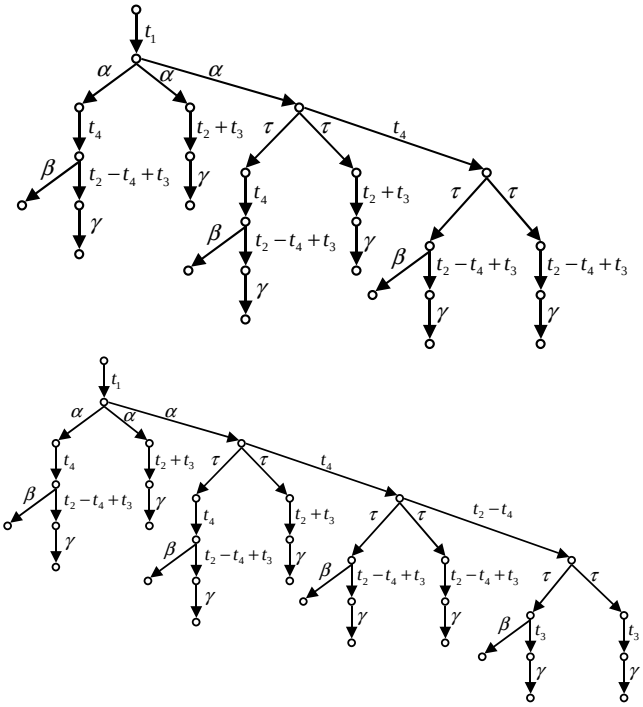

Figure 11: Generalized Transition Relation of M' derived from Fig. 10 by rule 6 of Definition 5

s_3 are moved into the transitions $s_2 \xrightarrow{\gamma'@?t[x_5+x_8\le t\le y_5+y_8]} s_5$, $s_2' \xrightarrow{\gamma'@?t[x_5+x_8\le t\le y_5+y_8]} s_5$, $s_2 \xrightarrow{\delta'@?t[x_5+x_9\le t\le y_5+y_9]} s_{10}$, and $s_2' \xrightarrow{\delta'@?t[x_5+x_9\le t\le y_5+y_9]} s_{10}$, similarly.

Formally, the abstraction rule for sequential structures is defined as follows:

Definition 8 *An* Abstraction rule for Sequential Structures *for PTIA is defined as a transformation function AbsS eq on PTIA as follows:*

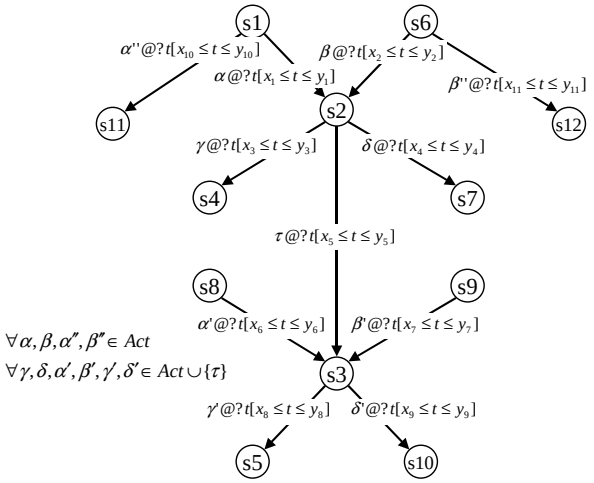

Figure 12: Abstraction for Sequential Structures in Presence of Outgoing/Incoming Transitions (Before Abstraction)

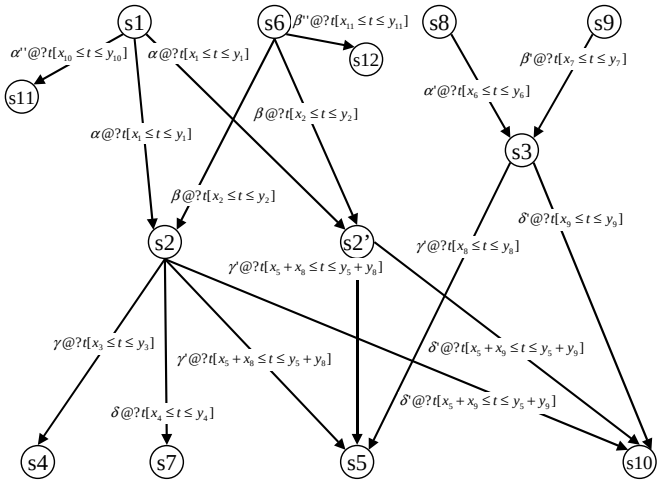

Figure 13: Abstraction for Sequential Structures in Presence of Outgoing/Incoming Transitions (After Abstraction)

$[\bigwedge_{i\in I}\{s_{1,i} \xrightarrow{\alpha_i@?t[P_{1,i}]} s_2\} \wedge s_2 \xrightarrow{\tau@?[P_2]} s_3 \wedge \bigwedge_{j\in J}\{s_2 \xrightarrow{\beta_j@?[P_{3,j}]} s_{4,j}\} \wedge \bigwedge_{k\in K}\{s_3 \xrightarrow{\gamma_k@?[P_{4,k}]} s_{5,k}\}] \overset{AbsSeq}{\Longrightarrow} [\bigwedge_{i\in I}\{s_{1,i} \xrightarrow{\alpha_i@?t[P_{1,i}]} s_2 \wedge s_{1,i} \xrightarrow{\alpha_i@?t[P_{1,i}]} s_3\} \wedge \bigwedge_{j\in J}\{s_2 \xrightarrow{\beta_j@?[P_{3,j}]} s_{4,j}\} \wedge \bigwedge_{k\in K}\{s_2 \xrightarrow{\gamma_k@?[P_2 \Theta P_{4,k}]} s_{5,k} \wedge s_3 \xrightarrow{\gamma_k@?[P_2 \Theta P_{4,k}]} s_{5,k}\}]$, *where each* $s_{1,i}$ *(*$i \in I$*) is a stable state,* $\alpha_i \in Act$, $\beta_j, \gamma_k \in Act \cup \{\tau\}$, $P_{1,i}, P_2, P_{3,j}, P_{4,k} \in Intvl(Var)$ *for any* $i \in I$, $j \in J$, $k \in K$.

If there are some other incoming transitions $s_6^l \xrightarrow{\delta^l@?t[Q^l]} s_3$ $(l \in L)$, *then a new state* s_3' *is created and all the incoming and outgoing transitions of* s_3

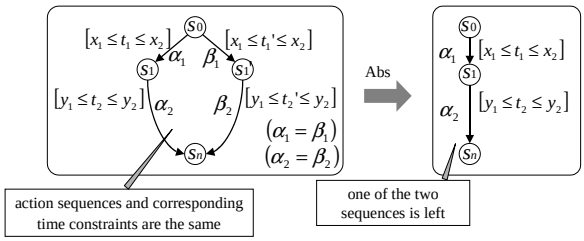

Figure 14: Abstraction for Branching Structures

is copied into those of s'_3 before applying this rule.

□

We consider the abstraction function $AbsSeq()$ as the mapping from a PTIA M to the modified PTIA M' by applying one of the abstraction rules in Definition 8. We also consider $AbsSeq()$ as the mapping from control states of M to the corresponding states of M'.

Then, we have the following theorem:

Theorem 2 *For any stable state s, if $s \overset{\alpha@?t[P]}{\longrightarrow} s_1 \overset{\tau@?t[Q]}{\longrightarrow} s_2$, any path π beginning with s contains no loops, and the internal transition $s_1 \overset{\tau@?t[Q]}{\longrightarrow} s_2$ is observably bounded, then for any σ, $(s,\sigma) \equiv_{gtb} (AbsSeq(s),\sigma)$.*

Proof. *(sketch) From Theorem 1 and Lemma 4.1, we can easily prove the general case by using induction on the number of the branches and using the congruence property (Theorem 1).*

4.2.3 Abstraction for Branching Structures

The abstraction for branching structures is illustrated in Fig. 14. It is clear that any external observer cannot find which branch is selected if the observable action sequences and the corresponding time constraints are the same. Thus, we leave just one of these sequences[2]. If there are some outgoing/incoming transitions at some eliminated control state, we move those transitions to the corresponding control state of another branch which is left.

Formally, the abstraction rule for branching structures is defined as follows:

Definition 9 *An* Abstraction rule for Branching Structures *for PTIA is defined as a transformation function AbsBranch on PTIA as follows:*

$$\begin{aligned}&[s_0 \overset{\alpha_1@?t[P_1]}{\longrightarrow} s_1 \overset{\alpha_2@?t[P_2]}{\longrightarrow} s_2 \overset{\alpha_2@?t[P_3]}{\longrightarrow} \cdots \overset{\alpha_n@?t[P_n]}{\longrightarrow} s_n\\ &\wedge\ s_0 \overset{\beta_1@?t[Q_1]}{\longrightarrow} s'_1 \overset{\beta_2@?t[Q_2]}{\longrightarrow} s'_2 \overset{\beta_2@?t[Q_3]}{\longrightarrow} \cdots \overset{\beta_n@?t[Q_n]}{\longrightarrow} s_n)\end{aligned}$$

[2]Note that the abstraction rule for branching structures is essentially not required for eliminating internal actions— they can be eliminated only by using the abstraction rules for sequantial and loop structures. The abstraction rule for branching structures is just for reducing the complexity (size) of the model. It is also possible to eliminate branches having different time constraints by using the operator "∪". However, here we do not include such a rule since it does not generally reduce the essential complexity of the model (for example, $s \overset{a@?t[P_1 \cup P_2]}{\longrightarrow} s'$ is always the same as $s \overset{a@?t[P_1]}{\longrightarrow} s' \wedge s \overset{a@?t[P_2]}{\longrightarrow} s'$).

$\overset{AbsBranch}{\Longrightarrow}$ $[s_0 \overset{\alpha_1@?t[P_1]}{\longrightarrow} s_1 \overset{\alpha_2@?t[P_2]}{\longrightarrow} s_2 \overset{\alpha_3@?t[P_3]}{\longrightarrow} \cdots \overset{\alpha_n@?t[P_n]}{\longrightarrow} s_n]$,
where $\alpha_1, \alpha_2, ..., \alpha_n, \beta_1, \beta_2, ..., \beta_n \in Act \cup \{\tau\}$ *and* $\alpha_i = \beta_i$ *and* $P_i = Q_i$ *for all* $i \in \{1, \ldots, n\}$.

If there are some other outgoing transition $s_i' \overset{\gamma@?t[R]}{\longrightarrow} s_i''$ *from* s_i' $(2 \leq i \leq n)$, *then we replace it with* $s_i \overset{\gamma@?t[R]}{\longrightarrow} s_i''$. *The case of incoming transition to* s_i' *is similar.*

□

Similar to *AbsSeq*(), we consider the abstraction function *AbsBranch*() as the mapping from a PTIA M to the modified PTIA M'

Since the following theorem is rather straightforward, here we only show the results.

Theorem 3 *If* $s_0 \overset{\alpha_1@?t[P_1]}{\longrightarrow} s_1 \overset{\alpha_2@?t[P_2]}{\longrightarrow} s_2 \overset{\alpha_2@?t[P_3]}{\longrightarrow} \cdots \overset{\alpha_n@?t[P_n]}{\longrightarrow} s_n$ *and* $s_0 \overset{\beta_1@?t[Q_1]}{\longrightarrow} s_1' \overset{\beta_2@?t[Q_2]}{\longrightarrow} s_2' \overset{\beta_2@?t[Q_3]}{\longrightarrow} \cdots \overset{\beta_n@?t[Q_n]}{\longrightarrow} s_n$*), then for any* σ, $(s_0, \sigma) \equiv_{gtb} (AbsBranch(s_0), \sigma)$.
□

4.3 Terminating Property of Abstraction Algorithm

Our proposed abstraction algorithm is to apply repeatedly the abstraction rules in Section 4.2 until no changes occur. In this section, we show that this abstraction algorithm is ensured to terminate.

Firstly, we define the abstraction algorithm more precisely.

Definition 10 Abstraction Algorithm *is defined as follows:*

1. *Input PTIA M.*
2. *Apply Abstraction Rule for Sequential Structures to M.*
3. *Apply Abstraction Rule for Branching Structures to M.*
4. *Repeat (2)-(3) until no changes occurred in M.*
5. *Output PTIA M.* □

Definition 11 *Let Abs*() *be the abstraction function which represents the application of either AbsSeq*() *or AbsBranch*(). □

Then, the following theorem holds.

Theorem 4 *For any PTIA M, there exists some natural number n such that* $Abs^n(M)$ *contains no internal transitions. Here,* $Abs^n(M)$ *means the PTIA to which the abstraction rules are applied n times.*

Proof. *(sketch) From Definition 11, it can be proven that the function Abs*() *generally monotonically decreases the number of internal transitions. Moreover, it can be shown that any internal transitions can be eliminated by the proposed abstraction rules if their directly preceding transitions are observable and they are observably bounded. Furthermore, since the transition graph is a DAG and the initial state is stable, we can repeatedly apply the abstraction rules from the top to the bottom of the DAG. From the fact above, we can prove the theorem.*

From this theorem, the following corollary immediately holds.

Corollary 1 *The abstraction algorithm in Definition 10 eventually terminates for any input M.* □

5 Equivalence Checking

In this section, we show that parametric global timed bisimulation equivalence checking on PTIA is reduced to parametric timed strong bisimulation checking on PTIA without internal transitions.

By applying the algorithm of Definition 10 to two PTIAs M_1 and M_2, we obtain two PTIAs $Abs(M_1)$ and $Abs(M_2)$, which have no internal transitions and global timed bisimulation equivalent to M_1 and M_2, respectively. On the other hand, from the result of Ref.[3], we can obtain the parameter condition in order that $Abs(M_1)$ and $Abs(M_2)$ are timed strong bisimulation equivalent. Since timed strong bisimulation equivalence implies global timed bisimulation equivalence, and global timed bisimulation equivalence satisfies the transitive law, the obtained parameter condition is also the parameter condition in order that M_1 and M_2 are global timed bisimulation equivalent.

Definition 12 *A binary relation R on states of a timed LTS is a* timed strong bisimulation *if the following condition hold:*

If $(s_1, \sigma_1)R(s_2, \sigma_2)$, then for any $\alpha \in Act \cup \mathbf{R}^+ \cup \{\tau\}$,

1. $\forall\, s_1', \sigma_1' [\ (s_1, \sigma_1) \xrightarrow{\alpha} (s_1', \sigma_1')\ \Rightarrow$
   $\exists s_2', \sigma_2'\ [\ (s_2, \sigma_2) \xrightarrow{\alpha} (s_2', \sigma_2') \wedge (s_1', \sigma_1')R(s_2', \sigma_2')\ ]\ ]$,
   and,
2. $\forall\, s_2', \sigma_2' [\ (s_2, \sigma_2) \xrightarrow{\alpha} (s_2', \sigma_2')\ \Rightarrow$
   $\exists s_1', \sigma_1'\ [\ (s_1, \sigma_1) \xrightarrow{\alpha} (s_1, \sigma_1') \wedge (s_1', \sigma_1')R(s_2', \sigma_2')\ ]\ ]$

We say that states (s_1, σ_1) and (s_2, σ_2) are timed strong bisimulation equivalent, *denoted by $(s_1, \sigma_1) \equiv_{tsb} (s_2, \sigma_2)$ if and only if there exists a timed strong bisimulation R such that $(s_1, \sigma_1)\ R\ (s_2, \sigma_2)$.*

The following relationship holds among timed strong bisimulation, timed weak bisimulation, and global timed bisimulation.

Proposition 2 *For any two states (s_1, σ_1) and (s_2, σ_2) of a timed LTS, $(s_1, \sigma_1) \equiv_{tsb} (s_1, \sigma_1)$ implies $(s_1, \sigma_1) \equiv_{twb} (s_1, \sigma_1)$, and $(s_1, \sigma_1) \equiv_{tsb} (s_1, \sigma_1)$ implies $(s_1, \sigma_1) \equiv_{gtb} (s_1, \sigma_1)$.* □

From the discussions above, the following theorem holds.

Theorem 5 *For any PTIAs M_1 and M_2, if there exists some natural numbers n and m such that $Abs^n(M_1)$ and $Abs^m(M_2)$ contain no internal transitions, and $Abs^n(M_1) \equiv_{tsb} Abs^m(M_2)$ if and only if $M_1 \equiv_{gtb} M_2$.* □

6 Conclusion

In this paper, we proposed a parametric time-interval automaton(PTIA) and its transformation algorithm to eliminate internal actions while preserving global timed bisimulation, and showed that parametric global timed bisimulation equivalence checking on PTIA can be reduced to the existing parametric timed strong bisimulation equivalence checking method without internal transitions.

The future work is to relax some of the restrictions imposed on target PTIAs, especially for the loops. For preserving GTB, we confirmed that abstraction is still possible by the proposed abstraction rules in some cases containing loops, but there are some weird examples the proposed abstraction rules cannot be applied. On the other hand, for preserving timed trace equivalence, we are successfully developed the abstraction algorithm for unrestricted PTIAs. We are currently working on PTIAs containing various loop structures and developing more general abstraction algorithms for preserving GTB and/or timed trace equivalence.

References

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

[2] de Frutos, D., López, N., Núñez, M.: Global timed bisimulation: An introduction. In: Proc. of Joint Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols XII, and Protocol Specification, Testing, and Verification XIX (FORTE/PSTV'99). (1999) 401–416

[3] Nakata, A., Higashino, T., Taniguchi, K.: Time-action alternating model for timed LOTOS and its symbolic verification of bisimulation equivalence. In Gotzhein, R., Bredereke, J., eds.: 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), IFIP, Chapman & Hall (1996) 279–294

[4] Xu, J.: On inspection and verification of software with timing requirements. IEEE Trans. on Software Engineering **29** (2003) 705–720

[5] Sifakis, J., Tripakis, S., Yovine, S.: Building models of real-time systems from application software. Proceedings of the IEEE **91** (2003) 100–111

[6] Wang, F.: Parametric timing analysis for real-time systems. Information and Computation **130** (1996) 131–150

[7] Hennessy, M., Lin, H.: Symbolic bisimulations. Theoretical Computer Science **138** (1995) 353–389

[8] Lin, H.: Symbolic transition graph with assignment. In: Proc. of CONCUR'96. Lecture Notes in Computer Sciences, Springer-Verlag (1996)

[9] Li, Z., Chen, H.: Computing strong/weak bisimulation equivalences and observation congruence for value-passing processes. In: Proc. of Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). (1999) 300–314

[10] Larsen, K.G., Wang, Y.: Time abstracted bisimulation: Implicit specifications and decidability. In Brookes, S., Main, M., Melton, A., Mislove, M., Schmidt, D., eds.: Proc. of 9th Int'l Conf. on Mathematical Foundations of Programming Semantics (MFPS'93). Volume 802 of Lecture Notes in Computer Science., Springer-Verlag (1993) 160–175

[11] Alur, R., Courcoubetis, C., Henzinger, T.A.: The observational power of clocks. In: Proc. of CONCUR'94. Volume 836 of Lecture Notes in Computer Science., Springer-Verlag (1994) 162–177

[12] de Nicola, R., Hennessy, M.: Testing equivalence for processes. Theoretical Computer Science **34** (1984) 83–133