13

GENERATING TEST CASES FOR A TIMED I/O AUTOMATON MODEL

Teruo Higashino†, Akio Nakata††, Kenichi Taniguchi† and Ana R. Cavalli†††

† : *Dept. of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan*

Tel : +81-6-6850-6590 Fax : +81-6-6850-6594

Email : higashino@ics.es.osaka-u.ac.jp

†† : *Dept. of Computer Science, Hiroshima City University, Japan*

††† : *Institut National des Télécommunications, Evry France*

Abstract Recently various real-time communication protocols have been proposed. In this paper, first, we propose a timed I/O automaton model so that we can simply specify such real-time protocols. The proposed model can handle not only time but also data values. Then, we propose a conformance testing method for the model. In order to trace a test sequence (I/O sequence) on the timed I/O automaton model, we need to execute each I/O action in the test sequence at an adequate execution timing which satisfies all timing constraints in the test sequence. However, since outputs are given from IUTs and uncontrollable, we cannot designate their output timing in advance. Also their output timing affects the executable timing for the succeeding I/O actions in the test sequence. Therefore, in general, the executable timing of each input action in a test sequence can be specified by a function of the execution time of the preceding I/O actions. In this paper, we propose an algorithm to decide efficiently whether a given test sequence is executable. We also give an algorithm to derive such a function from an executable test sequence automatically using a technique for solving linear programming problems, and propose a conformance testing method using those algorithms.

Keywords: real-time protocols, timed automata, conformance testing, traceability

1. INTRODUCTION

Conformance testing is one of methods to improve the reliability of communication protocols [3, 11]. However, conformance testing for models with

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35567-2_25

G. Csopaki et al. (eds.), *Testing of Communicating Systems*

© IFIP International Federation for Information Processing 1999

the notion of time has not sufficiently studied. Recently, some conformance testing methods have been proposed in [4, 5, 13, 15] for real-time models such as timed automata [1].

Alur's timed automata model [1] is a simple and nice model for specifying real-time systems, and a lot of verification techniques have been proposed [2]. However, basically it does not treat data values used in communication protocols. Especially, in multimedia communication protocols such as QoS control for video streams, sometimes one may want to specify different timeout intervals depending on the size of data to be transmitted. Also, it is possible that the timing of each action of the process may depend on the type and/or size of transmitted data. For the above purposes, we need models that can treat not only *time* but also *data values.* It is also desirable that such models have efficient verification and/or testing methods. Here, we need a model which combines timed automata with EFSMs.

In testing EFSMs or real-time systems, there are some problems to be solved. First, a given test sequence is not always executable. In order to execute the test sequence, we must find some appropriate input values or execution timing which satisfy its transition conditions. In contrast to the case of EFSMs, in the case of real-time systems, the tester can designate the input timing. However, in general, the output timing is not controlled by the tester and it is decided by each IUT itself. Moreover, the executable timing of some I/O action may depend on the execution time of its preceding I/O actions. It is desirable that whenever the preceding output actions are executed, there always exists some adequate input timing such that its succeeding sequence is executable. Thus, in this paper we propose a *timed I/O automaton model* for specifying real-time protocols and a conformance testing method for the model which handles the problems described above.

In our timed I/O automaton model, each transition is either input or output action. Moreover, in order to describe timing constraints among actions, we introduce some *variables* and one special *global time variable* which always holds the current time. The *variables* can hold not only time values but also values expressed as linear expressions of the time values and input data. Each *transition condition* can be specified by a *logical conjunction* of linear inequalities of those two types of variables. Note that any Alur's time automata can be specified in this model.

We define two kinds of executability (traceability) of test sequences, *must-traceability* and *may-traceability*. A must-traceable test sequence can be always executed if we specify some appropriate input timing for its input actions, no matter when its output actions are executed. A may-traceable test sequence can be executed only when the execution time of its output actions belongs to the sub-ranges which make the succeeding actions executable. In this paper, we present an efficient algorithm for checking the must/may-traceability of given

test sequences and obtaining the upper and lower bounds for each input action as functions of the execution time of its preceding I/O actions.

Based on UIOv-method [16], we propose a conformance testing method for our model. Our method can be used for improving the reliability of a given IUT. In our method, we assume that the errors in IUTs fall into some specific types, and under the assumption we check the correctness of IUTs by checking (1) the traceability of the derived test sequences and (2) the executability of each transition condition at some specified boundary time where the boundary time denotes the moments when the truth value of the transition condition changes.

2. TIMED I/O AUTOMATON MODEL

2.1 DEFINITIONS

Definition 1 *A* timed I/O automaton *is a 10-tuple* $M =< S, A, I/Otype, t, V, Pred, Def, \delta, s_{init}, \{x_{1init}, x_{2init}, \ldots, x_{kinit}\} >$, *where*

- $S = \{s_0, s_1, \ldots, s_n\}$ *is a finite set of states.*
- *A is a finite set of I/O actions.*
- $I/Otype = \{!, ?\} \cup \{?_{\mathbf{v}} | \mathbf{v}$ *is an input variable*$\}$ *is a set of I/O types, where the symbols ? and ! represent input and output, respectively. The symbol* $?_{\mathbf{v}}$ *represents that the input value is assigned to the variable* $\mathbf{v}$ *whose value may be used in the transition conditions of its succeeding actions. Each variable* $\mathbf{v}$ *can hold a rational number.*
- t *denotes the global clock variable which always holds the current time as a rational number.*
- $V = \{x_1, x_2, \ldots, x_k\}$ *is a finite set of variables which can hold rational numbers.*
- *Pred is a set of linear inequalities* $P[t, x_1, x_2, \ldots, x_k]$ *on rational numbers and their logical* conjunctions.
- *Def is a set of* assignments. *An assignment is a function which maps variables* $x_i \in V$ *to a linear expression* $f(t, \mathbf{v}, x_1, x_2, \ldots, x_k)$, *denoted by* $x_i \leftarrow f(t, \mathbf{v}, x_1, x_2, \ldots, x_k)$.
- $\delta \subseteq S \times A \times I/Otype \times Pred \times Def \times S$ *is a transition relation.*
- $s_{init} \in S$ *is the initial state of* M.
- $\{x_{1init}, x_{2init}, \ldots, x_{kinit}\}$ *is a set of the initial values for variables* $x_1, x_2, \ldots, x_k \in V$.

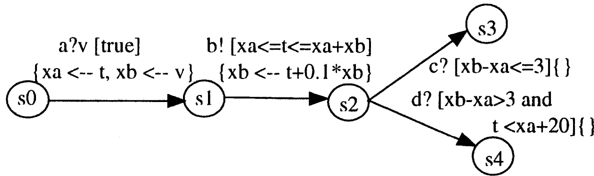

Figure 1 Semantics of the proposed model

- *There is no state from which there are two outgoing transitions with the same I/O actions.*

Note that, from the last constraint in the above definition, the timed I/O automaton is deterministic. Output values are omitted in this model since if they affect succeeding transition conditions, such conditions can be specified using the variables in V, input variables and global clock variable t. An element of a transition relation $(s, a, \$, P, D, s') \in \delta$ of M is denoted by $s \overset{a\$[P]D}{\longrightarrow}_M s'$. If M is understood from the context, we simply write $s \overset{a\$[P]D}{\longrightarrow} s'$.

Intuitively, the semantics of our timed I/O automaton model is as follows. For instance, at state s_0 in Fig. 1, the input action $a?_{\mathrm{v}}$ is executable. If $a?_{\mathrm{v}}$ has been executed, the state moves into s_1 and the execution time of $a?_{\mathrm{v}}$ is assigned to x_a according to the assignment $\{x_a \leftarrow t\}$. Also, the input value **v** is assigned to x_b according to the assignment $\{x_b \leftarrow \mathbf{v}\}$. If the execution time of $a?_{\mathrm{v}}$ is 5 and the input value **v** is 3, then the values of x_a and x_b become 5 and 3, respectively. So the output action $b!$ will be executed on time t such that the transition condition $5 \leq t \leq 5+3$ holds, that is, within time 8. If the execution time of $b!$ is 6, the value of $t + 0.1 * x_b$, which is equal to $6 + 0.1 * 3 = 6.3$, is assigned to variable x_b and the state moves into s_2. At state s_2, there are two choices. If $x_b - x_a \leq 3$ holds, then the input action $c?$ is executable. For the above case, since $x_b - x_a = 6.3 - 5 = 1.3 \leq 3$ holds, $c?$ is executable. In the case that $x_b - x_a > 3$ holds, another input action $d?$ is executable for 20 seconds after $a?$ is executed.

Relation to Timed Automata. Our model can simulate any timed automaton in the original version of Alur's timed automata [1]. Let's consider the example in Fig. 2. Since the global clock variable t has the current time in our model, by replacing each clock variable t_1 in Alur's timed automaton into $t - t_1$, we can obtain the same time constraints in our model. The $reset(t_1)$ operation can be simulated by assigning the current time t to the clock variable t_1 ($\{t_1 \leftarrow t\}$). If Alur's model has several clock variables, then the corresponding our model also has (at most) the same number of variables.

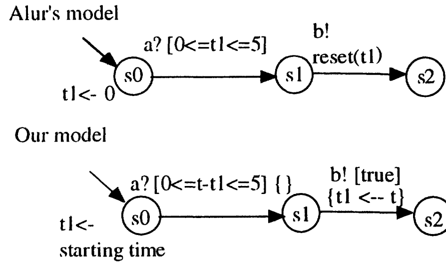

Figure 2 Simulation of Alur's automata

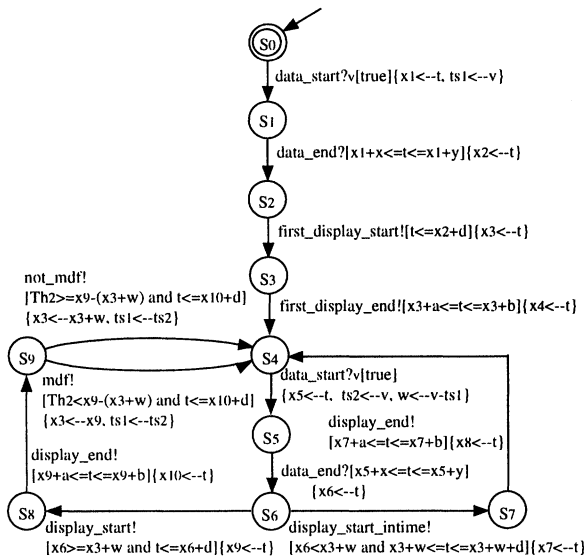

Figure 3 Media synchronization protocol

2.2 EXAMPLE

As an example of the timed I/O automata, we present a slightly modified specification of a receiving node of a media synchronization protocol[8] in Fig. 3. The media synchronization protocol specifies how to synchronize real-time continuous media such as video stream when the transfer rate quickly changes.

In this system, the sending node tries to send data continuously at the fixed rate to the receiving node. First, the sending node sends to the receiving node a video packet with a time-stamp representing its sending time. The receiving node starts to receive the data at time x_1, and assigns its time-stamp to ts_1.

Depending on its network propagation delay (minimum x, maximum y), it finishes to receive the data at time x_2 ($x_1 + x \leq x_2 \leq x_1 + y$). Next, it outputs a signal which instructs to start playing the video packet before time $x_2 + d$ where d denotes the maximum time necessary for preparation to play video. Then, it outputs a signal which instructs to finish playing the video packet after the duration necessary to finish playing (minimum a, maximum b), which depends on the decoding time of the data such as MPEG encoded data. After that, the receiving node carries out the above work continuously. Moreover, it synchronizes the playing rate with the sending node. To do so, it sets the objective starting time for playing the next video packet to time $x_3 + w$, where x_3 is the starting time for playing the previous video packet and w is a difference between two time-stamps of the most recent video packet and its previous one. If the actual completion time of receiving data is earlier than the objective one ($x_6 < x_3 + w$), the system waits until the objective time and then outputs the signal to start playing. Otherwise, it immediately (at time x_9) outputs the signal ("quick recovery of synchronization errors"). However, the error is within the specified tolerance T_{h2} ($T_{h2} \geq x_9 - (x_3 + w)$), it considers that the playing has started in time ($not_mdf!$), and that the system does not adjust the synchronization interval. Otherwise ($T_{h2} \leq x_9 - (x_3 + w)$), the actual starting time x_9 of playing is assigned to x_3, and the system adjusts the synchronization interval ($mdf!$). Then, it repeats the behaviour described above. Note that a, b, d, x, y and T_{h2} in Fig. 3 are constant parameters specified by designers.

3. EXECUTABILITY OF TRANSITION SEQUENCES

3.1 MUST/MAY TRACEABILITY

A *transition sequence* of a timed I/O automaton M is viewed as an execution path of the transition graph of M. However, in a timed I/O automaton, the value of each variable may change by executing each transition. In order to decide whether a given transition sequence is executable, we must consider how to change the values of those variables on execution of the actions in the transition sequence. To reflect such a change, we apply the technique inspired by the general symbolic execution technique (e.g. [5, 10]). First, we assign an unique name for each occurrence of variables in the transition sequence. Then we replace each occurrence of variables with an expression containing the initial values of the variables and the uniquely named variables.

Definition 1 *Let α denote a transition sequence $s_0 \overset{a_1\$_1[P_1]D_1}{\longrightarrow} s_1 \overset{a_2\$_2[P_2]D_2}{\longrightarrow} \ldots \overset{a_n\$_n[P_n]D_n}{\longrightarrow} s_n$ of a timed I/O automaton M where s_0 is the initial state s_{init} of M. Let $t_1, \ldots, t_n$ denote n different variables which represent the execution times of $a_1, \ldots, a_n$, respectively. Similarly, let $\mathbf{v}_1, \ldots, \mathbf{v}_m$ denote*

m different variables which represent the input values of the corresponding m occurrences of data-input actions in α. Let $x_1^{(i)}, \ldots, x_k^{(i)}$ represent the values of variables $x_1, \ldots, x_k \in V$ on i-th state s_i of α. Then, using the following algorithm, we express $x_1^{(i)}, \ldots, x_k^{(i)}$ by expressions containing only variables in $\{t_1, \ldots, t_i, x_{1init}, \ldots, x_{kinit}, \mathbf{v}_1, \ldots, \mathbf{v}_m\}$.

- $j := 0$, *for* $p = 1$ *to* k *do* $x_p^{(0)} := x_{pinit}$
- *for* $i = 1$ *to* n *do*
  - *if* $\$_i = ?_{\mathbf{v}}$ *then* $j := j + 1$; $\$_i := ?_{\mathbf{v}_j}$
  - *for* $p = 1$ *to* k *do*
    - *if* $x_p \leftarrow f(t, \mathbf{v}, x_1, \ldots, x_k) \in D_i$
      $x_p^{(i)} := f(t_i, \mathbf{v}_j, x_1^{(i-1)}, \ldots, x_k^{(i-1)})$
    - *else (there are no assignments to x_p in D_i)*
      $x_p^{(i)} := x_p^{(i-1)}$

For each transition condition $P_i[t, x_1, \ldots, x_k]$ of α, let $\widehat{P_i}$ be a condition defined by

$$\widehat{P_1} \stackrel{\text{def}}{=} P_1[t_1/t, x_{1init}/x_1, \ldots, x_{kinit}/x_k]$$

$$\widehat{P_i} \stackrel{\text{def}}{=} P_i[t_i/t, x_1^{(i-1)}/x_1, \ldots, x_k^{(i-1)}/x_k] \wedge (t_{i-1} \leq t_i)$$

where each $x_k^{(i)}$ is the expression obtained by the above algorithm, and $P_i[t_i/t, u_1/x_1, \ldots, u_k/x_k]$ means the expression obtained from $P_i[t, x_1, \ldots, x_k]$ by substituting $t_i, u_1, \ldots, u_k$ into $t, x_1, \ldots, x_k$, respectively. Each t_i represents the execution time of action a_i. We call $w \stackrel{\text{def}}{=} (a_1\$_1, t_1, \widehat{P_1}) \ldots (a_n\$_n, t_n, \widehat{P_n})$ as a symbolic trace *for α.*

Example 1 *The symbolic trace w for the transition sequence (which contains a loop) $s_0 \xrightarrow{a![t \leq x_b+7]\{x_a \leftarrow t\}} s_1 \xrightarrow{b?_{\mathbf{v}}[t \leq x_a+3]\{x_b \leftarrow t, x_a \leftarrow x_a+\mathbf{v}\}} s_2 \xrightarrow{c?[t \leq x_b+5 \wedge x_a+15 \leq t]\{\}} s_0 \xrightarrow{a![t \leq x_b+7]\{x_a \leftarrow t\}} s_1$ of a timed I/O automaton M is obtained as $w = (a!, t_a, t_a \leq x_{binit} + 7)\ (b?_{\mathbf{v}_1}, t_b, t_b \leq t_a + 3 \wedge t_a \leq t_b)\ (c?, t_c, t_c \leq t_b + 5 \wedge t_a + \mathbf{v}_1 + 15 \leq t_c \wedge t_b \leq t_c)\ (a!, t'_a, t'_a \leq t_b + 7 \wedge t_c \leq t'_a)$.*

Definition 2 *We say that a symbolic trace w is* must-traceable, *if whenever each output action is executed, there always exists some input timing for each input action such that the rest of the sequence can be executed. On the other hand, we say that a symbolic trace w is* may-traceable, *if for some output timing there exists some input timing such that the rest of the sequence can be executed.*

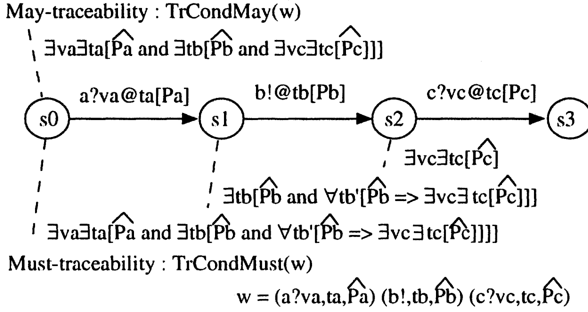

Figure 4 Must/may traceability

For a symbolic trace w, let $TrCondMust(w)$ and $TrCondMay(w)$ denote the expressions representing whether w is must/may traceable or not. That is, w is *must-traceable* (or *may-traceable*) if and only if $TrCondMust(w)$ (or $TrCondMay(w)$) is true. $TrCondMust(w)$ and $TrCondMay(w)$ are calculated recursively like the expressions in Fig. 4. For $TrCondMust(w)$, the sub-expression $\exists t_b[\widehat{P_b}]$ denotes that there exists a timing such that the output action $b!$ is executable. And, the sub-expression $\forall t_{b'}[\widehat{P_b} \Rightarrow \exists v_c \exists t_c[\widehat{P_c}]]$ denotes that there exists an input timing t_c executing $c?$ and an adequate input value v_c for any output timing in which $b!$ is executed.

Example 2 *A symbolic trace* $w = (a!, t_a, 1 \leq t_a \leq 5)(b?_{\mathbf{v}}, t_b, t_b \leq t_a + \mathbf{v} \wedge t_b \leq 5 \wedge t_a \leq t_b)$ *is must-traceable because the following expression is true.*

$$\exists t_a[1 \leq t_a \leq 5 \wedge \forall t'_a[1 \leq t'_a \leq 5 \Rightarrow \exists \mathbf{v} \exists t_b[t_b \leq t'_a + \mathbf{v} \wedge t_b \leq 5 \wedge t'_a \leq t_b]]]$$

A symbolic trace $w' = (a!, t_a, 1 \leq t_a \leq 6)(b?_{\mathbf{v}}, t_b, t_b \leq t_a + \mathbf{v} \wedge t_b \leq 5 \wedge t_a \leq t_b)$ *is not must-traceable, since the input action* $b?_{\mathbf{v}}$ *is not executable if the output action* $a!$ *is executed at time* t_a *such that* $5 < t_a \leq 6$. *However,* w' *is may-traceable since* $b?_{\mathbf{v}}$ *is executable if* $a!$ *is executed at time* t_a *such that* $1 \leq t_a \leq 5$. *That is,*

$$\begin{aligned} &\exists t_a[1 \leq t_a \leq 6 \wedge \\ &\qquad \forall t'_a[1 \leq t'_a \leq 6 \Rightarrow \exists \mathbf{v} \exists t_b[t_b \leq t'_a + \mathbf{v} \wedge t_b \leq 5 \wedge t'_a \leq t_b]]] \\ &\equiv false \end{aligned}$$

$$\exists t_a[1 \leq t_a \leq 6 \wedge \exists \mathbf{v} \exists t_b[t_b \leq t_a + \mathbf{v} \wedge t_b \leq 5 \wedge t_a \leq t_b]] \equiv true$$

Decision of must/may traceability. In general, $TrCondMust(w)$ and Tr $CondMay(w)$ become rational Presburger sentences and it is known that there exists a decision procedure for the general class [7]. However, the decision

problem is NP-hard in the general class. Here, we only use inequalities on rational numbers and their logical conjunctions. Hereafter, for this restricted class, we will propose an efficient decision algorithm and decide the must/may traceability efficiently.

3.2 EFFICIENT DECISION OF MUST/MAY TRACEABILITY

For simplicity, we regard a linear inequality $f(t, x_1, x_2, \ldots) < t$ as $f(t, x_1, x_2, \ldots) + \rho \leq t_n$ for a sufficiently small positive rational number ρ and consider only two inequality relations $\leq$ and $\geq$. We may omit the ρ in the rest of the paper.

First, we will consider must-traceability. Intuitively, the proposed algorithm for checking must-traceability of a symbolic trace $w = (a_1\$_1, t_1, \widehat{P_1}) \ldots (a_n\$_n, t_n, \widehat{P_n})$ works as follows.

Since the last action a_n has no succeeding actions, the executable time t_n of a_n is a solution of the constraint P_n. In our model, we can transform the constraint into the conjunctions of the following three types of linear inequalities by appropriate transposition: (1) $\{f_i(t_1, \ldots, t_{n-1}) \leq t_n | i \in I\}$, (2) $\{t_n \leq g_j(t_1, \ldots, t_{n-1}) \; | j \in J\}$, and (3) a conjunction $R_n(t_1, \ldots, t_{n-1})$ of inequalities which contain no t_n's. Therefore, we can obtain the lower bound $t_n^{inf} = max\{f_i(t_1, \ldots, t_{n-1}) | i \in I\}$ and the upper bound $t_n^{sup} = min\{g_j(t_1, \ldots, t_{n-1}) | j \in J\}$ as the interval of the executable time t_n. In order that there exists an executable time t_n of a_n, the expression $[t_n^{inf} \leq t_n^{sup}$ and $R_n(t_1, \ldots, t_{n-1})]$ must be true. Thus, let $TrCondMust_n(w)$ denote $[t_n^{inf} \leq t_n^{sup} \wedge R_n(t_1, \ldots, t_{n-1})]$. Since we can also express $t_n^{inf} \leq t_n^{sup}$ by $\bigwedge_{i \in I} \bigwedge_{j \in J} [f_i(t_1, \ldots, t_{n-1}) \leq g_j(t_1, \ldots, t_{n-1})]$, $TrCondMust_n(w)$ can be obtained as a conjunction of linear inequalities which contain only variables $t_1, \ldots, t_{n-1}$.[1] If $TrCondMust_n(w)$ is true, then the executable time t_n of a_n belongs to the interval $[t_n^{inf}, t_n^{sup}]$.

Now we will treat the general case. For each input action a_k such that $\$_k =?$ and $k < n$, the executable time t_k of a_k satisfies the constraint $P_k \wedge TrCondMust_{k+1}(w)$. From this constraint, we can obtain t_k^{sup}, t_k^{inf} and $TrCondMust_k(w)$ similarly. Here, $TrCondMust_k(w)$ represents the condition to make the transition sequence $a_k, a_{k+1}, \ldots, a_n$ in w must-traceable.

For each output action $a_{k'}$ such that $\$_{k'} =!$ and $k' < n$, we can obtain $TrCondMust_{k'}(w)$ as follows. Unlike input actions, the output action $a_{k'}$ can be executed at any output timing $t_{k'}$ satisfying $P_{k'}$, since the output timing is uncontrollable. Thus, first, from the constraint $P_{k'}$, we obtain $t_{k'}^{inf} = l_{P_{k'}}(t_1, \ldots, t_{k'-1})$, $t_{k'}^{sup} = u_{P_{k'}}(t_1, \ldots, t_{k'-1})$ and the conjunction of remain-

[1] If a_n is a data-input action $a_n?\mathrm{v}$, $TrCondMust_n(w)$ may also contain the variable v.

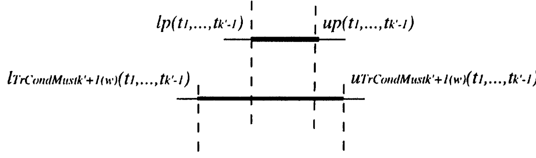

Figure 5 Must traceability for output actions

der clauses $R_{k'}(t_1, \ldots, t_{k'-1})$ which does not contain $t_{k'}$. Here, $[t_{k'}^{inf}, t_{k'}^{sup}]$ denotes the interval of executable output timing for $a_{k'}$. Next, we transform $TrCondMust_{k'+1}(w)$ into the conjunction of the following three forms : (1) $\{f_i(t_1, \ldots, t_{k'-1}) \leq t_{k'} | i \in I'\}$, (2) $\{t_{k'} \leq g_j(t_1, \ldots, t_{k'-1}) \ | j \in J'\}$, and (3) a conjunction of inequalities $R'_{k'}(t_1, \ldots, t_{k'-1})$ which does not contain $t_{k'}$. Then, we obtain the lower bound $l_{TrCondMust_{k'+1}(w)}(t_1, \ldots, t_{k'-1}) = max\{f_i(t_1, \ldots, t_{k'-1}) | i \in I'\}$ and the upper bound $u_{TrCondMust_{k'+1}(w)}(t_1, \ldots, t_{k'-1}) = min\{t_{k'} \leq g_j(t_1, \ldots, t_{k'-1}) \ | j \in J'\}$ as the interval of executable time $t_{k'}$ which satisfies $TrCondMust_{k'+1}(w)$. The remainder $R'_{k'}(t_1, \ldots, t_{k'-1})$ for $TrCondMust_{k'+1}(w)$ is also obtained as an expression with variables $t_1, \ldots, t_{k'-1}$.

Then, $a_{k'}$ is executable and the succeeding sequence is also executable for any output timing $t_{k'}$ of $a_{k'}$ if and only if $t_1, \ldots, t_{k'-1}$ satisfy the following three conditions :

- $l_P(t_1, \ldots, t_{k'-1}) \leq u_P(t_1, \ldots, t_{k'-1})$ holds (i.e. $a_{k'}$ is executable),
- as shown in Fig. 5, the interval $[l_P(t_1, \ldots, t_{k'-1}), u_P(t_1, \ldots, t_{k'-1})]$ is included in the interval $[l_{TrCondMust_{k'+1}(w)}(t_1, \ldots, t_{k'-1}), u_{TrCondMust_{k'+1}(w)}(t_1, \ldots, t_{k'-1})]$ (that is, $l_{TrCondMust_{k'+1}(w)}(t_1, \ldots, t_{k'-1}) \leq l_P(t_1, \ldots, t_{k'-1}) \wedge u_P(t_1, \ldots, t_{k'-1}) \leq u_{TrCondMust_{k'+1}(w)}(t_1, \ldots, t_{k'-1})$ holds), and
- $R_{k'}(t_1, \ldots, t_{k'-1}) \wedge R'_{k'}(t_1, \ldots, t_{k'-1})$ holds.

We use the logical conjunction of these conditions as the condition $TrCondMust_{k'}(w)$ and carry it over to the preceding actions.

By applying this method recursively, we can finally obtain $TrCondMust_1(w)$. In general, $TrCondMust_1(w)$ is a logical combination of linear inequalities which may contain the input variables $\mathbf{v}_1, \ldots, \mathbf{v}_m$ and variables in V. By assigning the initial values to the variables in V, we obtain a formula $D(\mathbf{v}_1, \ldots, \mathbf{v}_m)$ which contains $\mathbf{v}_1, \ldots, \mathbf{v}_m$ only. If this formula is satisfiable, we conclude that the given symbolic trace is must-traceable. The formula corresponds to $TrCondMust(w)$. Its satisfiability can be checked using the

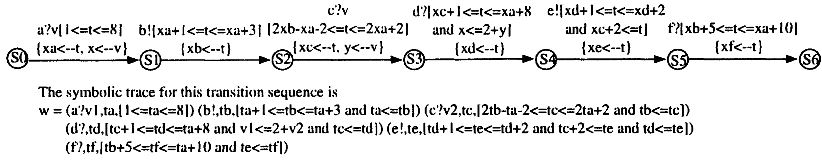

Figure 6 A transition sequence

technique for solving linear programming problems. A solution of $\mathbf{v}_1, \ldots, \mathbf{v}_m$ can be also obtained by the technique.

Sufficient condition for more efficient decision of traceability. Note that the formula $l_{TrCondMust_{k'+1}(w)}\ (t_1, \ldots, t_{k'-1}) \leq l_P(t_1, \ldots, t_{k'-1})$, which is generally expressed by $max\{f_i(t_1,\ \ldots,\ t_{k-1})\ |i = 1, ..., p\} \leq max\{f'_j(t_1,\ \ldots, t_{k-1})\ |j = 1, ..., q\}$, is actually checked by dividing into q cases, $max\{f_1,\ \ldots, f_p\} \leq f'_1 \vee max\{f_1, \ldots, f_p\} \leq f'_2 \vee \ldots \vee max\{f_1, \ldots, f_p\} \leq f'_q$. In each case, $max\{f_1, \ldots, f_p\} \leq f'_j$ can be expressed by a logical conjunction of inequalities such as $\bigwedge_{i=1,\ldots,p} (f_i \leq f'_j)$, so the presented algorithm can be applied.

Here, if we replace the condition $max\{f_i(t_1, \ldots, t_{k-1})\ |i = 1, ..., p\} \leq max\{f'_j(t_1, \ldots,\ t_{k-1})\ |j = 1, ..., q\}$ with its sufficient condition $max\{f_i(t_1,\ \ldots, t_{k-1})\ |i = 1, ..., p\} \leq min\{f'_j(t_1,\ \ldots,\ t_{k-1})\ |j = 1, ..., q\}$, the $TrCond\ Must_1$ obtained using the replaced condition is also a sufficient condition that the given symbolic trace is must-traceable. In this case, the replaced condition can be expressed by one logical conjunction of inequalities (without disjunction) such as $\bigwedge_{i=1,\ldots,p} \bigwedge_{j=1,\ldots,q} (f_i(t_1, \cdots, t_{n-1}) \leq f'_j(t_1, \cdots, t_{n-1}))$. So we do not have to divide into cases and can efficiently check must-traceability by applying the presented algorithm (the sufficient condition works well for our example cases in Section 5).

Example 3 *For example, we will apply the algorithm to the symbolic trace in Fig. 6. The result is shown in Fig. 7. According to the result, if we give input values* $\mathbf{v}_1$ *and* $\mathbf{v}_2$ *which satisfy* $\mathbf{v}_1 \leq 2 + \mathbf{v}_2$*, the sequence is must-traceable. As understood from the result, the executable interval for each action is expressed by the function of the execution time of preceding actions. On the actual execution of the sequence, we can choose any timing within the interval* $[t_i^{inf}, t_i^{sup}]$ *for input action* $a_i?$.[2] *Once the actual timing* t'_i *is decided, every occurrence of the variable* t_i *in the interval expression of every succeeding action is replaced with the actual value* t'_i*. Then the resulting expression is*

[2]For each output action, we observe its output timing and check whether the timing is in the interval.

$$
\begin{aligned}
(t_f^{inf}, t_f^{sup}) &= (max\{t_e, t_b + 5\}, t_a + 10) \\
TrCondMust_f(w) &= [max\{t_e, t_b + 5\} \leq t_a + 10] \\
(t_e^{inf}, t_e^{sup}) &= (max\{t_d, t_d + 1, t_c + 2\}, t_d + 2) \\
TrCondMust_e(w) &= [max\{t_d, t_d + 1, t_c + 2\} \leq t_d + 2 \wedge t_d + 2 \leq t_a + 10 \\
&\quad \wedge t_b + 5 \leq t_a + 10] \\
(t_d^{inf}, t_d^{sup}) &= (max\{t_c, t_c + 1\}, min\{t_a + 8, t_a + 10\}) \\
TrCondMust_d(w) &= [max\{t_c, t_c + 1\} \leq min\{t_a + 8, t_a + 10\} \wedge t_b + 5 \leq t_a + 10 \\
&\quad \wedge \mathbf{v}_1 \leq 2 + \mathbf{v}_2] \\
(t_c^{inf}, t_c^{sup}) &= (max\{t_b, 2t_b - t_a - 2\}, min\{2t_a + 2, t_a + 7\}) \\
TrCondMust_c(w) &= [max\{t_b, 2t_b - t_a - 2\} \leq min\{2t_a + 2, t_a + 7\} \\
&\quad \wedge t_b + 5 \leq t_a + 10 \wedge \mathbf{v}_1 \leq 2 + \mathbf{v}_2] \\
(t_b^{inf}, t_b^{sup}) &= (max\{t_a, t_a + 1\}, t_a + 3) \\
TrCondMust_b(w) &= [max\{t_a, t_a + 1\} \leq t_a + 3 \\
&\quad \wedge t_a + 3 \leq min\{t_a + 5, 2t_a + 2, t_a + 7, 1.5t_a + 2, t_a + 4.5\} \\
&\quad \wedge \mathbf{v}_1 \leq 2 + \mathbf{v}_2] \\
(t_a^{inf}, t_a^{sup}) &= (max\{1, 2\}, 8) \\
TrCondMust_a(w) &= [max\{1, 2\} \leq 8 \wedge \mathbf{v}_1 \leq 2 + \mathbf{v}_2] \\
TrCondMust(w) &= \exists \mathbf{v}_1 \exists \mathbf{v}_2 [\mathbf{v}_1 \leq 2 + \mathbf{v}_2]
\end{aligned}
$$

Figure 7 Checking must traceability

partially computed. The actual interval for each succeeding action is gradually fixed by substituting the actual timing for preceding actions.

May-traceability. The "may-traceability" case is quite similar to the must case with a slight modification. We only treat each output action as an input action and derive the intervals of execution time for those I/O actions. If the execution time of each output action belongs to the derived time interval, which is generally narrow than the original executable time interval for the output action, then its succeeding actions can be executed. Thus, we omit the detail.

4. CONFORMANCE TESTING FOR TIMED I/O AUTOMATA

In this section, we will propose a conformance testing method. Here, we use UIOv-method[16] where, for each state, a transfer sequence and a succeeding UIO sequence are treated as a test case and we derive executable time interval for each input action in the test case using the algorithm explained above.

4.1 PROPOSED TESTING METHOD

Correctness of implementation of states and transitions. In the proposed method, we identify all states in the IUT as follows.

1 For a given timed I/O automaton, we consider the corresponding FSM in which all transition conditions are removed. We compose a set of UIO sequences $U = \{u_1, \ldots, u_n\}$ $(i = 0, \ldots, n)$ for each state s_i in the FSM. Also we generate the set of transfer sequences $V = \{v_1, \ldots, v_n\}$, each of which leads M from its initial state to a state s_i.

2 Let $V.U_\circ = \bigcup_i v_i.u_i$ and $V.U_\times = \bigcup_{i \neq j} v_i.u'_j.a_{ij}$, where u'_j denotes the longest executable prefix of u_j after v_i is executed, and a_{ij} denotes the first unexecutable I/O action of u_j. Intuitively, $V.U_\circ$ is a set of sequences which check the *executability* of the UIO sequence for each state, and $V.U_\times$ is a set of sequences which check the *un-executability* of UIO sequences for the different states.

3 We decide the must/may-traceability for all sequences in $V.U_\circ$. If all sequences are must-traceable, or at least may-traceable, we can use those sequences as test cases.[3] If not, we return to step (1) and construct different U and V again.

4 We give the generated test cases $V.U_\circ$ and $V.U_\times$ to the IUT. At that time, we execute each input action of a test sequence in $V.U_\circ$ with an executable timing which is calculated from the function obtained by the must/may-traceability checking algorithm described in Section 3 by substituting the actual execution time of preceding actions into the function. For each output action, we check whether the output action is observed at a time which satisfies the timing constraints described in the specification.For each test sequence $v_i.u'_j.a_{ij}$ in $V.U_\times$, we confirm that it is impossible to execute a_{ij} after execution of $v_i.u'_j$ with various timing for more than F (sufficiently large number) times.[4]

5 If we can observe the same response from the IUT as that of the specification (that is, all the sequences of $V.U_\circ$ are executable and those of $V.U_\times$ are not executable), we conclude that we have identified all states in the IUT.

We also identify all transitions in the IUT like the above method. At that time, we construct $V.A.U_\circ = \{v_i.a_{ij}.u_j | a_{ij}$ is an outgoing transition (action)

[3] It is desirable that, for each test sequence $v_i.u_i$, the UIO sequence part u_i is must-traceable even if the transfer sequence part v_i is only may-traceable.

[4] Note that we may confirm that we can execute only a partial sequence of u'_j.

from state s_i }, $V.A_{\times}= \{v_i.a_{ij}|a_{ij}$ is not an outgoing transition (action) from state $s_i\}$ where the set V of transfer sequences is the same one as used in the above state identification. Then we confirm that we can observe the same response from the IUT for $V.A.U_{\circ}$ as that of the specification, and that we cannot execute a_{ij} in each test sequence of $V.A_{\times}$ after giving v_i with various timing for more than F (sufficiently large number) times. If we can observe the same response from the IUT as that of the specification, we conclude that we have identified all transitions in the IUT.

When we give $V.U_{\circ}$ and $V.U_{\times}$ to the IUT and observe the same response as the specification, we regard that the IUT has n different states, which have the same set of UIO sequences as those of the corresponding states of the specification. For the identification of transitions, we use the same set V of transfer sequences as used for state identification. Similar to the traditional UIOv-method, we also regard that for each state in the specification, the corresponding state in the IUT has the same outgoing transitions. Giving an UIO sequence to the destination state of the outgoing transition, we regard that the destination state corresponds to that of the specification.

Correctness of implementation of transition conditions. Here, we present a method to detect faults in transition conditions of IUTs if the faults are restricted to some typical ones.

On testing of the correctness of transition conditions, we find transitions from the initial state step by step using the breadth first search and test them. For testing of the last k-th transition of a transition sequence whose length is k, we assume that we have already tested the correctness of all the transitions from the first one to the $(k-1)$-th one, and that they are correctly implemented.

To simplify our explanation, we assume that the timing constraint of the k-th transition on a test sequence is specified as $(A_1 \leq t_k) \wedge (A_2 \leq t_k) \wedge \ldots \wedge (A_n \leq t_k) \wedge (t_k \leq B_1) \wedge (t_k \leq B_2) \wedge \ldots \wedge (t_k \leq B_m)$, and that only one of them, for example, $(A_h \leq t_k)$ may be incorrectly implemented and its fault is either one of the following six types of errors : (1) $(A_h < t_k)$, (2) $(A_h \geq t_k)$, (3) $(A_h > t_k)$, (4) $(A_h = t_k)$, (5) $(A_h + C \leq t_k)$ and (6) $(A_h - C \leq t_k)$ (c: positive constant).

As we mentioned in Section 3, the execution time of k-th transition on a given test sequence is specified as $max(A_1, \ldots, A_n) \leq t_k \leq min(B_1, \ldots, B_m)$. Then if A_h is less than $max(A_1, \ldots, A_n)$, it makes no effects even if $(A_h \leq t_k)$ is implemented as $(A_h < t_k)$ incorrectly. For example, even if the timing constraint $(t_{k-1} + 1 \leq t_k) \wedge (t_{k-1} \leq t_k)$ on a specification is implemented as $(t_{k-1} + 1 \leq t_k) \wedge (t_{k-1} < t_k)$ incorrectly, the incorrect part $(t_{k-1} < t_k)$ makes no effects on the correctness of the whole transition condition because the faulty constraint is equal to the correct one $(t_{k-1} + 1 \leq t_k)$. Because we cannot test the correctness of such a condition, we try to test only the correctness

of the transition condition which is executable even if we add the constraint $A_p + \epsilon < A_h$ (ϵ: sufficiently small constant, $\epsilon \ll C$) for all other A_p.

For the execution time of k-th transition, we select three types of execution time, (a) $max(A_1, \ldots, A_n) - \epsilon$, (b) $max(A_1, \ldots, A_n)$, (c) $max(A_1, \ldots, A_n) + \epsilon$. The response from the specification (correct IUT) and wrong IUTs with one fault in (1)–(6) is described as follows (the marks $\circ$ and $\times$ mean "executable" and "unexecutable", respectively).

Response	(a)	(b)	(c)
Spec.	$\times$	$\circ$	$\circ$
(1)	$\times$	$\times$	$\circ$
(2)	$\circ$	$\circ$	$\times$
(3)	$\circ$	$\times$	$\times$
(4)	$\times$	$\circ$	$\times$
(5)	$\times$	$\times$	$\times$
(6)	$\circ$	$\circ$	$\circ$

Clearly as this table, we can detect every fault because those responses are different from the response in the specification. For the above example, we generate a symbolic trace w corresponding to the transition sequence from the first transition to k-th transition. Then, if k-th transition is an input action, we try to check that it is possible to execute k-th transition with the above execution timing (b) and (c), and that it is impossible to execute k-th transition for more than F times with the above execution timing (a) after execution of w with various timings. If we can confirm the above, we conclude that the implementation of k-th transition is correct. While if k-th transition is an output action, we try to check that the output action can be executed with the execution timing (b) and (c), and that it cannot be executed with (a) for many times. If we can confirm them, we regard that the transition condition is correct. In general, there may exist other types of errors than the above (1)–(6). Also there may exist multiple faults. So, we cannot detect all types of errors as easily as we proposed above. However we are sure to be able to detect a significant number of errors by our method because these types of errors are typical.

5. EXAMPLE

We can apply our testing method to the example in Fig. 3 by constructing the set of UIO sequences U as follows.

$$
\begin{aligned}
U = \{ \; & data_start?data_end?first_display_start!, \\
& data_end?first_display_start!, \\
& first_display_start!, first_display_end! \\
& data_start?data_end?display_start!,
\end{aligned}
$$

$$data_end?display_start!, display_start!,$$
$$display_start_intime!, display_end!not_mdf!,$$
$$display_end!data_start?, not_mdf!\}$$

The set of shortest paths from the initial state to all states can be used as the set of transfer sequences V. We have confirmed we can generate the set of must-traceable test sequences : (A) $V.U_\circ$ and $V.U_\times$ for identifying states, and (B) $V.A.U_\circ$ and $V.A_\times$ for identifying transitions. For example, for a test sequence $w = data_start?_{\mathbf{v}_1}$ $data_end?$ $first_display_start!$ $first_display_end!$ $data_start?_{\mathbf{v}_2}$ $data_end?$ $display_start!$ $display\ _end!$ $not_mdf!$ for identifying state s_9, the condition to make the sequence w must-traceable is $D(\mathbf{v}_1, \mathbf{v}_2) = [b \leq T_{h2} + \mathbf{v}_2 - \mathbf{v}_1 - x]$. If the constants b, T_{h2} and x are actually $10, 2$ and 10, respectively, any input data $\mathbf{v}_1, \mathbf{v}_2$ which satisfy the inequality $10 \leq 2 + \mathbf{v}_2 - \mathbf{v}_1 - 10$ (that is, $18 \leq \mathbf{v}_2 - \mathbf{v}_1$) can be given in order to make w must-traceable. The executable time interval for each input action in w can be specified as a function of the execution time of its preceding actions and input values $\mathbf{v}_1$ and $\mathbf{v}_2$.

We have developed a tool to decide whether a formula like formulas in Fig. 4 is true or not. By using the tool, we can check the must/may-traceability of test sequences like the above example within a few seconds for most cases (Pentium II 200MHz). In general, if we apply the general algorithms checking the satisfiability of rational Presburger sentences [7], it takes much more time to decide the must/may-traceability of a given test sequence (in some cases, a few hours or more). Also, the sufficient condition introduced in Section 3.3 makes the decision time further short. For the example in Fig. 3, we can prove all the test sequences are must-traceable using the sufficient condition introduced in Section 3.

We have applied our technique to the following examples which can be modeled as our timed I/O automata using our test case derivation tool :

- the multiplexing protocol for multimedia communication like H.223[9] which sends packets consisting of three media, that is, movie (MPEG2 etc.), sound and text data.

- Ethernet protocol[14] and its time-out mechanism.

For the above examples, we have derived must-traceable test sequences for most cases. However, for checking some time-out actions (or interruption), we can only derive may-traceable test sequences since such time-out actions can be executed only when their preceding output actions are executed very late. Even for such cases, we can recognize when the preceding output actions should be executed in order to make a time-out action executable.

6. CONCLUSION

In this paper, we have proposed a timed I/O automaton model for specifying real-time communication protocols, and a method to generate conformance test cases with execution timing for their I/O actions. In general, from a state, if there is a choice (branch) between two output actions and/or between an input action and an output action, then the tester cannot control which output action should be executed, that is, the decision of such a choice is made by the IUT itself. If an undesirable branching output action is executed during a test run, we must reset the test run and try the same test run again. The paper [12] treats such a problem and has proposed a suitable testing method. The method for treating such uncontrollable output actions is also applicable to our method.

As the future work, we are planning to develop a tool which translates given specifications written in time extended SDL and E-LOTOS into the proposed timed I/O automata, and to derive test sequences from those specifications directly.

References

[1] R. Alur and D.L. Dill : "A Theory of Timed Automata", Theoretical Computer Science, Vol. 126, pp.183-235 (1994).

[2] R. Alur and D.L. Dill : "Automata-Theoretic Verification of Real-Time Systems", In Formal Methods for Real-Time Computing, Trends in Software Series, pp.55-82 (1996).

[3] B. S. Bosik and M. U. Uyar : "Finite State Machine Based Formal Methods in Protocol Conformance Testing", Computer Networks ISDN Systems, 22, pp. 7-33 (1991).

[4] D. Clarke and I. Lee : "Automatic Generation of Tests for Timing Constraints from Requirements", Proc. of 3rd Int. Workshop on Object-Oriented Real-Time Dependable Systems (1997).

[5] L. K. Dillon : "Using Symbolic Execution for Verification of Ada Tasking Programs", ACM Trans. on Programming Languages and Systems, Vol. 12, No. 4, pp.643-669 (1990).

[6] A. En-Nouaary, R. Dssouli, F. Khendek and A. Elqortobi : "Timed Test Cases Generation Based on State Characterization Technique", Proc. of 19th IEEE Real-Time Systems Symposium (RTSS'98) (1998).

[7] J. E. Hopcroft and J. D. Ullman : "Introduction to Automata Theory, Languages, and Computation", Addison-Weslay (1979).

[8] Y. Ishibashi, S. Tasaka and E. Minami : "Performance Measurement of a Stored Media Synchronization Mechanism: Quick Recovery Scheme", Proc. of GLOBECOM'95, pp.811-817 (1995).

[9] ITU-T : "- Multiplexing Protocol for Low Bit Rate Multimedia Communication", Recommendation H.223 (1996).

[10] R. Kneuper : "Symbolic Execution: A Semantic Approach", Science of Computer Programming, Vol. 16, No. 3, pp.207-249 (1991).

[11] D. Lee and M. Yannakakis : "Principles and Methods of Testing Finite State Machines - A survey", Proc. of the IEEE, Vol. 84, No. 8 (1996).

[12] Q. M. Tan and A. Petrenko : "Test Generation for Specifications Modeled by Input/Output Automata", Proc. of 11th IFIP Workshop on Testing of Communicating Systems (IWTCS'98), pp.83-99 (1998).

[13] D. Mandrioli, S. Morasca, A. Morzenti: "Generating Test Cases for Real-Time Systems from Logic Specifications", ACM Trans. on Computer Systems, Vol. 13, No. 4, pp.365-398 (1995).

[14] R. Metcalfe and D. Boggs : "Ethernet: Distributed Packet Switching for Local Computer Networks", Communication of the ACM, Vol. 19, No. 7 (1976).

[15] J. Springintveld, F. Vaandrager, and P. R. D'Argenio : "Testing Timed Automata", Technical Report, CTIT 97-17, University of Twente (1997).

[16] S. T. Vuong, W.L. Chan and M. R. Ito : "The UIOv-Method for Protocol Test Sequence Generation", Proc. of 2nd IFIP Workshop on Protocol Test Systems (IWPTS'89), pp.161-175 (1989).