Vol. 37 No. 5 Transactions of Information Processing Society of Japan May 1996

Regular Paper

Timed Reachability Analysis Method for Communication Protocols Modeled by Extended Finite State Machines

SHIN'ICHI NAGANO,† YOSHIAKI KAKUDA† and TOHRU KIKUNO†

As communication systems rapidly progress, real-time performance is needed for communication protocols. In order to meet this requirement, communication protocols must incorporate real-time performance. This paper newly proposes a timed reachability analysis method in order to verify timeliness property of communication protocols under the assumption that times needed to execute events may be different from each other, but they are constant. The proposed method is efficiently performed by both enumerating only event sequences that are obtained through parallel execution of all possible events, and then by computing a processing time of each event sequence. This paper also gives proofs for correctness of the proposed method.

1. Introduction

As communication systems become large and complicated, it is required that they have real-time functions for processing among communication entities. For example, the functions are indispensable for multimedia systems which process continuous data, such as sound and image data. In order to meet this demand, communication protocols must incorporate real-time performance[1),9),11),12)].

In the typical previous methods for verification of timeliness property of communication protocols, extended finite state machines (EFSMs) and FIFO queues model communication entities and channels of communication protocols, respectively, as in standard protocol specification languages SDL and Estelle[4),8)]. In addition, temporal logic represents temporal execution orders of events, which are related to timeliness property[2),5)]. Then, the methods produce a reachability graph by the conventional reachability analysis, and then check whether each event sequence satisfies requirements on timeliness property. However, the reachability analysis does not consider a time needed to execute each event, which denotes either an operation in a process or a message transfer in a channel, and causes the state explosion during production of the reachability graph[6),7),10)]. Furthermore, although temporal logic can describe temporal execution orders of events, it cannot represent timeliness property itself.

† Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University

In order to resolve these problems, Kakuda, et al.[8)] have already proposed a verification method for timeliness property of communication protocols. The method consists of the following two analyses: conventional reachability analysis which enumerates all sequences of executable events, and timeliness analysis which computes processing time of each sequence. In addition, they assume that any event is executed in one time unit. However, this assumption is strict and unrealistic.

On the other hand, timed Petri nets are often used to model communication protocols[3),13)] and to verify the timeliness property. Although Petri nets are suitable for modeling the whole behavior of communication entities, it is difficult to concisely describe communication entities and channels individually, in particular FIFO delivery of messages in channels, for practical use.

This paper newly proposes a timed reachability analysis method in order to verify timeliness property of communication protocols under the assumption that times needed to execute events may be different from each other, but they are constant. By relaxing the assumption, the proposed method can model execution of timers and reception of messages, and analyze parallel execution of their events more precisely. The proposed method is efficiently performed by enumerating only event sequences that are obtained through parallel execution of all possible events based on protocol models, and then by computing a processing time of each event sequence. This paper also gives proofs for correctness of the proposed method.

Section 2 gives fundamental definitions on communication protocols modeled by EFSMs and FIFO queues, and Section 3 defines key concepts of this paper: events and system states. Next, Section 4 proposes a timed reachability analysis method, and then Section 5 proves correctness of the proposed method. Finally, Section 6 summarizes the results of this paper with some future work.

2. Communication Protocol

A communication protocol, shortly a protocol, consists of communication entities (called processes) and communication channels (called channels).

Definition 1 (Process)
A process P_i is modeled by an extended finite state machine (EFSM), $\hat{P_i} = (S, s_I, V, V_I, V_T, V_{TO}, OP, B, \delta)$, satisfying the followings:

$S = \{s_1, \ldots, s_p\}$: a finite set of states in P_i.

$s_I \in S$: an initial state in P_i.

$V = \{v_1, \ldots, v_q\}$: a finite set of variables in P_i.

$V_I = \{v_{1I}, \ldots, v_{qI}\}$: a set of initial values assigned to variables in V.

$V_T = \{v_{T_1}, \ldots, v_{T_r}\}$: a finite set of timer variables in P_i.

$T_{TO} = \{v_{TO_1}, \ldots, v_{TO_r}\}$: a set of the time-out values for timer variables in V_T.

$OP = \{op_1, \ldots, op_s\}$: a finite set of operations. Each operation $op_t \in OP$ $(1 \leq t \leq s)$ is "transmission" of messages from P_i to P_j, "reception" of messages from P_j to P_i, "addition" and "subtraction" on variables in P_i, "starting timer" in P_i, or "resetting timer" in P_i.

$B = \{b_1, \ldots, b_u\}$: a finite set of predicates. Each predicate takes the value of true or false. B includes predicates on timer.

$\delta = S \times B \times OP \to S$: a transition function for P_i. If P_i stays in $s_v \in S$, $b_w \in B$ is true and $op_x \in OP$ is executed, then P_i enters a new state $s_y \in S$. □

Definition 2 (Channel)
A channel C_{jk} from process P_j to process P_k is modeled by an FIFO queue $\hat{C_{jk}}$ whose capacity is finite and larger than zero. The capacity of $\hat{C_{jk}}$ and the number of messages being delivered in $\hat{C_{jk}}$ are denoted by $cap(\hat{C_{jk}})$ and $|\hat{C_{jk}}|$, respectively. □

Definition 3 (Communication protocol)
A communication protocol $\mathcal{P}$ is defined by $\mathcal{P} = (\mathbf{P}, \mathbf{C})$ with $\mathbf{P} = \{\hat{P_i} \mid 1 \leq i \leq N\}$ where $\hat{P_i}$ is an EFSM for process P_i, and N is the number of processes and $\mathbf{C} = \{\hat{C_{jk}} \mid 1 \leq j, k \leq N, j \neq k\}$ where $\hat{C_{jk}}$ is an FIFO queue for channel C_{jk}. □

For convenience of denotation, P_i and $\hat{P_i}$ are interchangeably used, and C_{jk} and $\hat{C_{jk}}$ are also done.

Definition 4 (Global state)
Let ps_i denote a state of process P_i and cs_{jk} denote a state of channel C_{jk}. Let *pred* be a predicate consisting of variables and the values. The predicate *pred* does not include timer variables. To define in detail, state $ps_i = \langle s_h : pred \rangle$ implies that P_i reaches state $s_h \in S$ (see Definition 1) and predicate *pred* holds at s_h. On the other hand, state $cs_{jk} = \langle msg_1 : pred_1, \ldots, msg_m : pred_m \rangle$ implies that each message variable msg_n $(1 \leq n \leq m)$ in C_{jk} satisfies predicate $pred_n$, and that $|C_{jk}| = m$. $cs_{jk} = \epsilon$ denotes that C_{jk} is empty.

If a state of each P_i is ps_i $(1 \leq i \leq N)$ and a state of each C_{jk} is cs_{jk} $(1 \leq j, k \leq N, j \neq k)$, then global state of the communication protocol is represented by $gs = (ps_1, \ldots, ps_N, cs_{12}, \ldots, cs_{N-1N})$. □

Figure 1 shows the example protocol which consists of three processes and four channels. Figures 1 (a), (b) and (c) describe specifications of three processes. In the description, "–" and "+" denote transmission and reception of messages, respectively. Next, predicate "$TO(v_T^i = v_{TO}^i)$" $(i = 1, 2)$ is true if and only if the value of the variable "v_T^i" is the same as time-out value specified by a timer variable "v_{TO}^i" in P_i. Intuitively, this predicate represents a condition for the time-out of the timer. We assume that the time-out value is quite greater than the time needed for messages to reach a receiver process from a sender process and to return to the sender process. There is no predicate other than predicates on timer in this example.

Behavior of the example protocol is intuitively explained as follows: Processes P_1 and P_2 simultaneously send message "*req*" to process P_3 through channels C_{13} and C_{23}, respectively. Then, P_3 waits for "*req*" at state s_{31} for establishment of a connection between P_1 and P_3 or between P_2 and P_3. If P_3 receives "*req*" from P_1 (P_2), then P_3 replies "*acpt*" to P_1 (P_2) through C_{31} (C_{32}) and sends "*rjct*" to P_2 (P_1) through C_{32} (C_{31}).

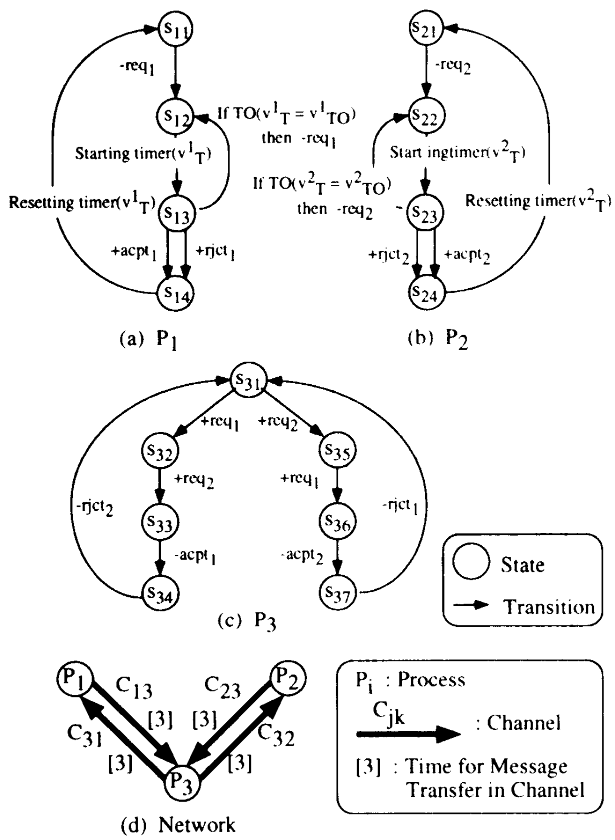


Fig. 1 Example of protocol.

Assume that one time unit is needed for each process P_i to handle any event except for "counting timer", three time units are needed for each channel to transfer any message, and twelve time units are set as the time-out value of each timer.

3. Event and System State

Definition 5 (Event)
Events in a communication protocol consist of process-type events to be executed in process P_i and channel-type events to be executed in channel C_{jk}. Additionally, process-type events are divided into seven kinds of activities: (1) "counting timer" in P_i and (2) six operations in P_i (OP in Definition 1), that is, (2-1) "transmission" of messages from P_i to P_j, (2-2) "reception" of messages from P_j to P_i, (2-3) "addition" on variables in P_i, (2-4) "subtraction" on variables in P_i, (2-5) "starting timer" in P_i and (2-6) "resetting timer" in P_i. On the other hand, channel-type events include only one kind of activities: (3) "message transfer" in channel C_{jk}. □

Remark 1 Although timers are attached to each process, timers are executed independent of the process, and the execution of each timer is represented by a "counting timer".

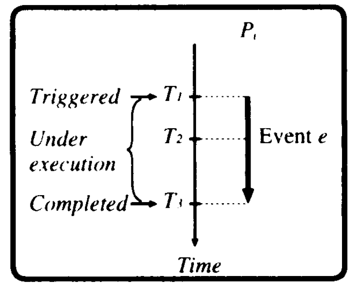


Fig. 2 Execution of event.

For each timer, a pair of a timer variable v_T and a time-out value v_{TO} is given (V_T and V_{TO} in Definition 1). The predicate on each timer is defined as $v_T = v_{TO}$. By executing a "starting timer", zero is assigned to v_T. Next, the value of v_T is incremented by the "counting timer" along with progress of time. Zero is also assigned to v_T by executing a "resetting timer".□

A global time is introduced in order to describe execution of events and states of a protocol. We consider the following assumption in order to make discussions clear.

Assumption 1 We assume that there is a discrete global time for a communication protocol. Any time can thus be represented by T time units, where T is a constant nonnegative integer. Let the initial value of global time be zero. We call such a global time the initial time. We also assume that execution of communication protocols starts at the initial time. In the following, global time is just denoted by time. □

Definition 6 (Execution of event)
Execution of events is described using an event sequence chart as shown in **Fig. 2**. Suppose that an event e is triggered in process P_i at time T_1 and that the event e is completed at time T_3. We define an execution time of e as $time(e) = T_3 - T_1$. Next, assume that the current time is $T_2 (T_1 < T_2 < T_3)$. Then we define a residual time of e as $res\_time(e) = T_3 - T_2$. Additionally, if $0 < res\_time(e) \leq time(e)$ holds at T_2, then we say that e is under execution at T_2. We assume that for each event e except for "counting timer", $time(e)$ is given. The execution time of "counting timer" is the time-out value set by "starting timer" under execution of a protocol. □

In the example protocol shown in Fig. 1, for example, execution times of events except for "message transfer" and "counting timer" are one time unit. Those of "message transfer" are three time units.

An event must be executable before it becomes triggered. Conditions that an event becomes executable are defined as follows.

Definition 7 (Conditions for executable event)

(1) An event e is executable at the initial time T_0 if conditions for Case 1-1 or Case 1-2 hold.

Case 1-1 (e is "addition" or "subtraction" on variables in process P_i, or "starting timer" or "resetting timer" in P_i)

Condition : P_i stays in state s at T_0 and a transition function $\delta(s, b, e)$ is defined in P_i.

Case 1-2 (e is "transmission" of messages from process P_i to process P_j $(i \neq j)$)

Condition : P_i stays in state s at T_0 and a transition function $\delta(s, b, e)$ is defined in P_i.

(2) An event e is executable at the current time $T(> T_0)$ if conditions for Case 2-1, Case 2-2, Case 2-3, Case2-4, or Case 2-5 hold.

Case 2-1 (e is "addition" or "subtraction" on variables in process P_i, or "starting timer" or "resetting timer" in P_i)

Condition : P_i stays in state s at T, and a transition function $\delta(s, b, e)$ is defined in P_i.

Case 2-2 (e is "transmission" of messages from process P_i to process P_j $(i \neq j)$)

Condition : For channel C_{ij}, $|C_{ij}|$ is smaller than $cap(C_{ij})$ at T, P_i stays in state s at T, and a transition function $\delta(s, b, e)$ is defined in P_i.

Case 2-3 (e is "reception" of a message msg from process P_j to process P_i $(j \neq i)$)

Condition : Event "message transfer" of msg in channel C_{ji} has been completed at T, any event in P_i is not under execution at T, P_i stays in state s at T, and a transition function $\delta(s, b, e)$ is defined in P_i.

Case 2-4 (e is "counting timer" in process P_i)

Condition : Event "starting timer" in P_i has been completed at T.

Case 2-5 (e is "message transfer" of a message msg in channel C_{ij})

Condition : Event "transmission" of msg from process P_i to process P_j has been completed at T. □

In the example protocol in Fig. 1, for example, "$-req_1$" and "$-req_2$" are executable at the initial time. Then, four events, "*starting timer*(v_T^1)", "*starting timer*(v_T^2)", and message transfers "(req_1)" and "(req_2)" are executable after completion of "$-req_1$" and "$-req_2$" (that is, time 1), respectively.

We assume that only one event except for "counting timer" in each process can be executed at the current time. Therefore, if more than one event except for "counting timer" is executable at time T, then only one event out of them can be triggered at T.

Definition 8 (Conditions for triggered event)

(1) An event e is triggered at the initial time T_0 if e is executable at T_0.

(2) An event e is triggered at the current time $T(> T_0)$ if conditions for Case 1 or Case 2 hold.

Case 1 (e is a process-type event in process P_i except for event "counting timer" in P_i)

Condition : Any event does not exist under execution in P_i at T except for event "counting timer" in P_i and e has become executable at T.

Case 2 (e is "counting timer" in P_i or a channel-type event in channel C_{jk})

Condition : e has become executable at T. □

Remark 2 Reception must be treated with care. Let event e_1 be "message transfer" of a message msg in channel C_{ji}. Suppose that event e_2 except for "counting timer" is under execution in the receiving process P_i at the current time T (see **Fig. 3**). Then, event "reception" in P_i (say e_3) cannot become executable at T (see Case 2-3 in Definition 7). However, if $res\_time(e_2) = t$ at T and event e_3 can become executable at a new time $T + t$, then e_3 becomes triggered at $T + t$. In the conventional reachability analysis method, e_3 is executed as soon as e_1 has become completed. □

Remark 3 Since timers are essential for the systems to possess real-time function, it is important to model and analyze execution of

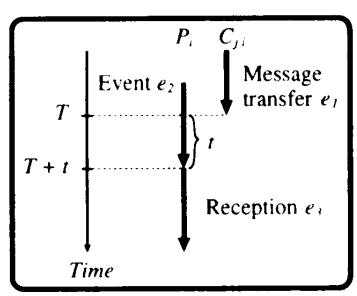


Fig. 3 Reception of message.

timers accurately. In previous method, it cannot be modeled that an event is executed after a certain time elapses.
There are three kinds of events for each timer: "starting timer", "counting timer" and "resetting timer". Suppose that "starting timer" in process P_i becomes completed at the current time T. If "counting timer" in P_i is not under execution at T, then "counting timer" becomes triggered at T. Otherwise, "counting timer" is forced to stop and again becomes triggered at T. Next, when "resetting timer" becomes completed at T, "counting timer" is forced to stop at T. Then, when "counting timer" becomes completed at T, a predicate on the timer (see Remark 1) becomes true at T except for the following case.
Let events e_1 and e_2 be an event "counting timer" in process P_i and an event to be executed in P_i after e_1 is completed, respectively. Suppose that event e_3 except for "counting timer" is under execution in process P_i at time T. Then, time-out occurs at T if e_1 is completed at T. In this case, we consider that the predicate on timer is not true at T and e_2 becomes executable after execution of e_3 is completed. □

In the example protocol in Fig. 1, for example, "$-req_1$" and "$-req_2$" are triggered at the initial time. Then, four events, "$starting\ timer(v_T^1)$" "$starting\ timer(v_T^2)$', "(req_1)" and "(req_2)" are triggered at time 1 because of their executability at time 1.

A system state is introduced in order to describe not only states of each process and channel but also parallel execution of events at a time.

Definition 9 (System state)
A system state ss at the current global time T in a communication protocol $\mathcal{P}$ is defined by $ss = (gs, ge)$, where $gs = (ps_1, \ldots, ps_N, cs_{12}, \ldots, cs_{N-1N})$ is a global state of $\mathcal{P}$ and $ge = \{\langle e_1, res\_time(e_1)\rangle, \ldots, \langle e_m, res\_time(e_m)\rangle\}$ is a set of pairs of event which is under execution at T, and its residual time. □

Definition 10 (Initial system state)
A system state $ss_0 = (gs_0, ge_0)$ with $gs_0 = (ps_1, \ldots, ps_N, cs_{12}, \ldots, cs_{N-1N})$ of a communication protocol $\mathcal{P}$ satisfying three conditions $C1$, $C2$ and $C3$ is called the initial system state. Here, each s_I^i $(1 \le i \le N)$ is the initial state of process P_i and each v_{hI}^i $(1 \le h \le m)$ is the initial value of variable v_h^i in P_i.
Condition $C1: \ \forall i\ (1 \le i \le N)\ [\ ps_i = \langle s_I^i : v_1^i = v_{1I}^i \wedge \ldots \wedge v_m^i = v_{mI}^i\rangle\ ]$
Condition $C2: \ \forall j,k\ (1 \le j,k \le N, j \ne k)\ [\ cs_{jk} = \epsilon\ ]$
Condition $C3: \ ge = \phi$ □

Condition $C3$ implies that no event is under execution.

Definition 11 (Next system state)
We define the next system state of a system state (let it be $ss_i = (gs_i, ge_i)$) depending on whether ss_i is the initial system state or not.

We first consider the next system state when ss_i is the initial system state. Let $ge_{i+1} = \{\langle e_1, time(e_1)\rangle, \ldots, \langle e_q, time(e_q)\rangle\}$ where $e_p (1 \le p \le q)$ is an event triggered at the initial time. Since no event is completed at the initial time, global state gs_i does not change. Then, the next system state of ss_i is defined as $ss_{i+1} = (gs_i, ge_{i+1})$. We define this binary relation as $ss_i \vdash ss_{i+1}$.

We next consider the next system state when ss_i is not the initial system state. Let $e_1, \ldots, e_m$ be events under execution which have the smallest residual time (say t) out of all events under execution in a system state ss_i at the current time T. Now, suppose that t time units elapse and $e_1, \ldots, e_m$ have become completed at a new time $T+t$. Then, one executable event for each process, timer and channel can be triggered at $T+t$ if any. Thus, a protocol enters a system state ss_{i+1} at time $T+t$, in which the set of pairs of event and its residual time is changed. Then, we say that ss_{i+1} is a next system state of ss_i, and represent this binary relation by $ss_i \vdash ss_{i+1}$. The method to construct the system state ss_{i+1} is described in Section 4. □

The initial system state of the example protocol is $ss_0 = (ge_0, ge_0)$ at the initial time where $gs_0 = (s_{11}, s_{21}, s_{31}, \epsilon, \epsilon, \epsilon, \epsilon)$ and $ge = \phi$. After making transmission events "$-req_1$" and "$-req_2$" triggered, the system state at that time is changed to $ss_1 = (gs_1, ge_1)$ where $gs_1 = gs_0$ and $ge_1 = \{\langle -req_1, 1\rangle, \langle -req_2, 1\rangle\}$. After completing "$-req_1$" and "$-req_2$", and making four events, "$starting\ timer(v_T^1)$", "$starting\ timer(v_T^2)$", "(req_1)" and "(req_2)" triggered, the system state at time 1 is $ss_2 = (gs_2, ge_2)$ where $gs_2 = (s_{12}, s_{22}, s_{31}, req_1, req_2, \epsilon, \epsilon)$ and $ge_2 = \{\langle starting\ timer(v_T^1), 1\rangle, \langle starting\ timer(v_T^2), 1\rangle, \langle (req_1), 3\rangle, \langle (req_2), 3\rangle\}$. For ss_0, ss_1 and ss_2, $ss_0 \vdash ss_1$ and $ss_1 \vdash ss_2$ hold.

Definition 12 (Reachable system state)
Let $\vdash^*$ be the reflexive and transitive closure

of $\vdash$. Then we say that a system state ss is reachable from a system state ss' if and only if $ss' \vdash^* ss$. If ss' is the initial system state, then we just say that ss is reachable. □

Definition 13 (Equivalence of system states)
Let two system states be $ss_i = (gs_i, \{\langle e_{i1}, res\_time(e_{i1})\rangle, \ldots, \langle e_{im}, res\_time(e_{im})\rangle\})$ at a time T_i and $ss_j = (gs_j, \{\langle e_{j1}, res\_time(e_{j1})\rangle, \ldots, \langle e_{jn}, res\_time(e_{jn})\rangle\})$ at a time T_j. If ss_i and ss_j satisfy four conditions shown below, then we say that ss_i is equivalent to ss_j and represent this equivalence by $ss_i \equiv ss_j$.

Condition $E1$: $gs_i = gs_j$
Condition $E2$: $n = m$
Condition $E3$: $\forall k\ [\ e_{ik} = e_{jk}\ ]$
Condition $E4$: $\exists T_c \forall k\ [\ t_{ik} - t_{jk} = T_c\ ]$ □

The conditions $E1$ through $E4$ in Definition 13 are indispensable in order to terminate the proposed method when a loop exists in an event sequence.

A global state does not include timer variables (see Definition 4), and execution of timers are represented by "counting timer" (see Remarks 1 and 3). Thus, for each timer, a residual time of "counting timer" is checked at condition $E4$ in Definition 13.

4. New Verification Method

This section proposes a timed reachability analysis method.

4.1 Assumptions

This subsection describes assumptions for the proposed method and proofs of its correctness.

Assumption 2 We assume that the times needed to execute events may be different from each other, but they are constant time units. □

The following three assumptions are necessary in order to guarantee that the number of system states is finite.

Assumption 3 Each variable in process takes integer and the number of the values taken by each variable is finite. □

Assumption 4 The number of timers in each process is finite. □

Assumption 5 Once any event e except for "counting timer" becomes triggered at the current time T, it is never forced to stop until $time(e)$ time units elapse. □

This assumption means that execution of the event e is not interrupted by the other events.

The following two kinds of data are input for the timed reachability analysis method: (1) A communication protocol $\mathcal{P}$ to be analyzed, including a time-out value for each timer. (2) $time(e_i)$ for each event e_i except for "counting timer". ($time(e_i)$ and $time(e_j)$ $(i \neq j)$ of two different events e_i and e_j may be different, but all of them are constant.)

The proposed method enumerates all possible sequences of events from these input. Event sequence charts graphically represent sequences of events with respect to time.

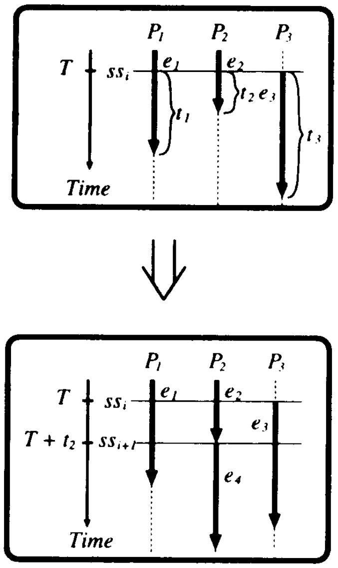
4.2 Construction of Event Sequence Chart

Next, we propose the method for construction of event sequence charts. This method is recursively described in the following.

Method 1 (Construction of event sequence chart)

(1) (Base Step) Assume that a protocol $\mathcal{P}$ stays in the initial system state $ss_0 = (gs_0, ge_0)$ at the initial time T_0 where $ge_0 = \phi$.
  (1-1) Compute events $e_1, \ldots, e_m$ which become executable and triggered in ss_0.
  (1-2) Generate the next system state $ss_1 = (gs_0, ge_1)$ at T_0, where $ge_1 = \{\langle e_i, time(e_i)\rangle \mid 1 \leq i \leq m\}$.

(2) (Induction Step) Assume that $\mathcal{P}$ stays in a system state $ss = (gs, ge)$ at the current time $T (\geq T_0)$, where $ge = \{\langle e_1, t_1\rangle, \ldots, \langle e_m, t_m\rangle\}$.
  (2-1) Select $\{\langle e_{i_1}, t_{i_1}\rangle, \ldots, \langle e_{i_q}, t_{i_q}\rangle\} \subset ge$ such that residual times t_{i_p} $(1 \leq p \leq q)$ are the smallest among t_i $(1 \leq i \leq m)$. Note that $t_{i_1} = \ldots = t_{i_q}$, and let them be t.
  (2-2) Proceed time by t, and then the current time is $T + t$. Then, events $e_{i_1}, \ldots, e_{i_q}$ become completed (see Definition 6).
  (2-3) Let a set of all events which become executable after completion of all e_{i_p} $(1 \leq p \leq q)$ be E. For each process, timer or channel, select one executable event from E. Let it be e'_r $(1 \leq r \leq s)$. For each combination of executable events $(e'_1, \ldots, e'_r, \ldots, e'_s)$, make each e'_r $(1 \leq r \leq s)$ simultaneously triggered in a process, a timer or a channel at $T + t$, and perform substeps (2-4) through (2-5). Thus, if more than one event in a process is executable at $T + t$, then more than one system state is generated at $T + t$. This case is discussed in Section 4.3.
  (2-4) Generate a next system state $ss' = (gs', ge')$ at a new time $T + t$, where gs' is a new global state which $\mathcal{P}$ enters from gs and $ge' = \{\langle e_i, t_i - t\rangle \mid \langle e_i, t_i\rangle \in$



(a)

(b)

Fig. 4 Construction of event sequence charts.

ge, $1 \le i \le m$, $i \ne i_1, \ldots, i \ne i_p\} \cup \{\langle e'_r, time(e'_r)\rangle \mid 1 \le r \le s\}$.

(2-5) If $ge' = \phi$, or the system state equivalent to ss' has been already generated, stop generation of the next system states of ss'. The case of stopping by the equivalence is also explained in Section 4.3. Otherwise, go to (2) with system state ss' at the current time $T + t$.

An example of the event sequence chart in **Fig. 4** briefly explains the method to construct an event sequence. Assume that a system state at the current time T is $ss_i = (gs_i, ge_i)$ where $ge_i = \{\langle e_1, t_1\rangle, \langle e_2, t_2\rangle, \langle e_3, t_3\rangle\}$ as shown in Fig. 4 (a). Then, events e_1 and e_2 in ss_i are under execution at T and event e_3 in ss_i is triggered at T. Assume that t_2 is the smallest of three residual times t_1, t_2 and t_3, and only event e_4 is executable after completion of e_2. After t_2 time units elapse, e_4 becomes triggered and a new system state ss_{i+1} at a time $T + t_2$ is generated. Then, ss_{i+1} is defined by $ss_{i+1} = (gs_{i+1}, ge_{i+1})$ where $ge_{i+1} = \{\langle e_1, t_1 - t_2\rangle, \langle e_4, t_4\rangle, \langle e_3, t_3 - t_2\rangle\}$ as shown in Fig. 4 (b).

4.3 Branching and Merging

In construction of event sequence charts, two special cases should be considered: branching from a system state and merging into a system state.

The branching from a system state is explained by using the same example in Section 4.2 (see **Fig. 5** (a)). Assume that after completion of e_2, two events in process P_2 become executable. Let them be events e_4 and e_5. In this case, the following two new system states are generated at time $T + t_2$: (1) system state $ss_{i+1} = (gs_{i+1}, ge_{i+1})$ where $ge_{i+1} = \{\langle e_1, t_1 - t_2\rangle, \langle e_4, t_4\rangle, \langle e_3, t_3 - t_2\rangle\}$, (2) system state $ss_j = (gs_j, ge_j)$ where $ge_j = \{\langle e_1, t_1 - t_2\rangle, \langle e_5, t_5\rangle, \langle e_3, t_3 - t_2\rangle\}$. By completion of event e_2, the event sequence is branching out to two event sequences as shown in Fig. 5 (b). In the following, we call such a system state the branching system state.

On the other hand, the merging of system states is explained by using an example in **Fig. 6**. Consider two system states ss_i at time T_i and ss_j at time T_j in two different event sequences as shown in Fig. 6 (a). One is $ss_i = (gs_i, ge_i)$ where $ge_i = \{\langle e_1, t_1\rangle, \langle e_3, t_3\rangle\}$, and no event becomes triggered at ss_i by completion of event e_2. The other is $ss_j = (gs_j, ge_j)$ where $ge_j = \{\langle e_1, t'_1\rangle, \langle e_3, t'_3\rangle\}$, and no event becomes triggered by completion of event e_4. Assume that relations $t'_1 - t_1 = t'_3 - t_3$ and $gs_i = gs_j$ hold. In this case, ss_i is equivalent to ss_j and the two event sequences following system states ss_i and ss_j become the same (this will be proved by Lemma 3 in Section 5). Then, two event sequences are merged by continuing construction of only one event sequence chart (and stopping further extension of other event sequence chart) as shown in Fig. 6 (b).

4.4 Application Results

We explain the result obtained by applying the timed reachability analysis method to the example protocol in Fig. 1. A part of event sequence charts are shown in **Fig. 7**. The processes and channels, and the system states are enumerated on the horizontal axis and the vertical axis, respectively. A system state corresponds exactly to a horizontal line and an event sequences are shown by vertical lines.

First, the proposed method starts at the initial system state ss_0 in Fig. 7 (a) at the initial time. Since the current time is the initial time, two events "$-req_1$" in P_1 and "$-req_2$" in P_2 are triggered at the initial time and the protocol enters the next system state ss_1 at that time. After one time unit elapses, these two events are completed and four events become triggered: "*starting* $timer(v_T^1)$" in P_1, "*starting* $timer(v_T^2)$" in P_2, message transfer "(req_1)" in C_{13}, and message transfer "(req_2)" in C_{23}. Then the protocol enters the next system state ss_2.

In a system state ss_3 at time 2, message transfer "(req_1)" in C_{13} and "(req_2)" in C_{23} are

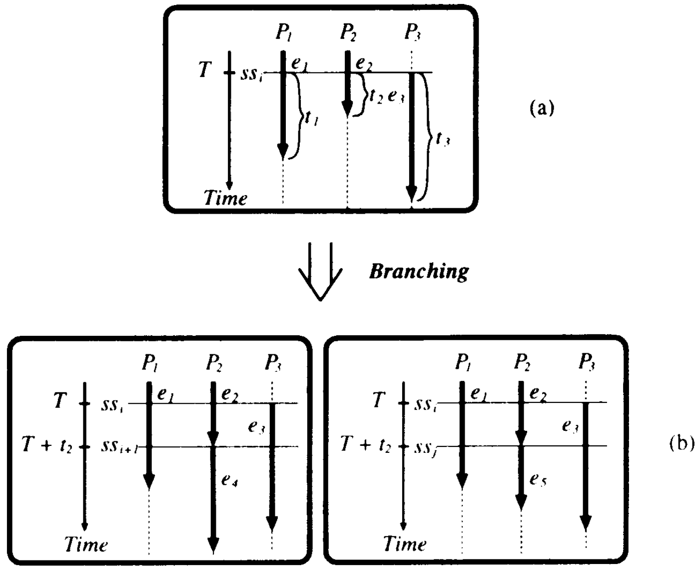


Fig. 5 Branching.

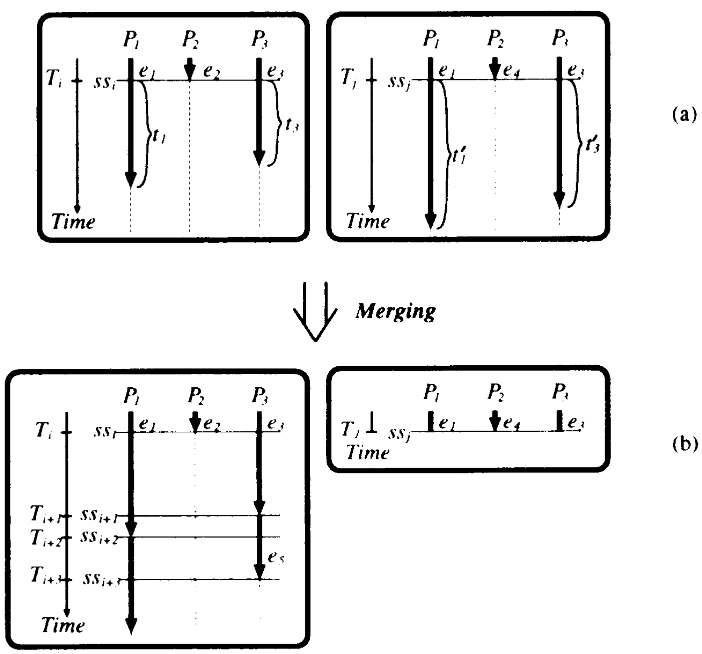


Fig. 6 Merging.

under execution. Since their residual times are 2 time units, they become completed at time 4. Then, "$+req_1$" and "$+req_2$" in P_3 become executable. However, they cannot be simultaneously triggered in P_3. Therefore, two system states ss_4 in which "$+req_1$" is triggered, and ss_4' in which "$+req_2$" is triggered, are generated at time 4. Thus, the event sequence chart in Fig. 7 (a) is branching out to two event sequence charts in Figs. 7 (a) and (b).

4.5 Comparison between Reachability Analysis and Timed Reachability Analysis

The well known reachability analysis is to enumerate all global states reachable from the initial state while the timed reachability analysis is to enumerate all system states reachable from the initial state. The reachability analysis is done by investigating temporal execution orders of events without considering the time needed to execute each event. It is thus diffi-

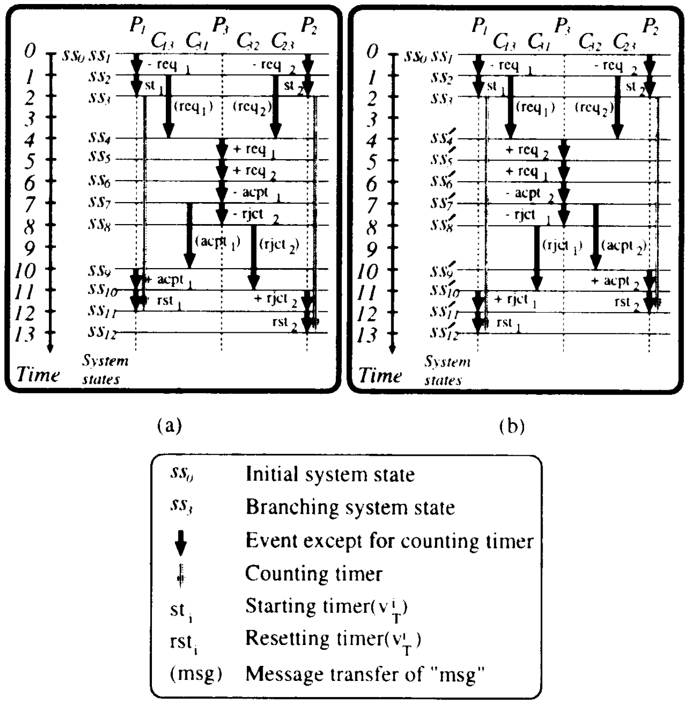


Fig. 7 Event sequence charts of example protocol.

cult to analyze parallel execution of events, especially those on timers and reception of messages. By introducing residual times of events in Definition 6 and giving conditions for executable events and triggered events in Definitions 7 and 8, the proposed timed reachability analysis method enables precise analysis of parallel execution of events for communication protocols with constant times to execute events.

The typical method for reachability analysis finds all executable events at each reachable global state and generates all permutations of such executable events. As a result, the number of generated global states becomes very large along with increase of the size of the communication protocols. On the other hand, since the proposed method uses not only execution orders of events but also the times needed to execute events, it can precisely analyze parallel execution of events. It is thus expected that the number of system states to be generated becomes smaller than that by the reachability analysis method.

5. Correctness of Proposed Method

This section proves the correctness of the timed reachability analysis method.

Definition 14 (Correctness)
We say that for a given communication protocol $\mathcal{P}$ the timed reachability analysis method is correct if it can enumerate all system states reachable from the initial system state of $\mathcal{P}$. □

Lemma 1 For any system state ss in the protocol $\mathcal{P}$, the proposed method need not generate any new system state for enumeration of reachable system states until at least one of events in ss under execution becomes completed.

Proof: Assume that the protocol $\mathcal{P}$ stays in a system state $ss = (gs, ge)$ at time T where $ge = \{\langle e_1, t_1\rangle, \ldots, \langle e_m, t_m\rangle\}$. Let a system state at time $T + t$ $(0 \leq t < t_{min})$ be $ss' = (gs', ge')$ where $t_{min} = min\{t_i \mid \langle e_i, t_i\rangle \in ge, 1 \leq i \leq m\}$. From Assumption 5, if e_i $(1 \leq i \leq m)$ is not "counting timer" in process P_j, then e_i cannot be completed at any time $T + t$ $(0 \leq t < t_{min})$. If e_i is "counting timer" in P_j, e may be forced to stop by completion of event "starting timer" or "resetting timer" in P_j. However, if an event e_k $(1 \leq k \leq m, k \neq i)$ is "starting timer" or "resetting timer" in P_j, e_k cannot become completed at $T + t$ as mentioned above. On the other hand, any event including "starting timer" and "resetting timer" in P_j cannot become triggered at $T + t$ $(0 < t < t_{min})$ because it can become triggered after completion of the other event (see substep (2-3) in the proposed method). Therefore, "counting timer" is not forced to stop at $T + t$. Thus, global state gs does not change, that is, $gs = gs'$. Furthermore, since $ge' = \{\langle e_1, t_1 - t\rangle, \ldots, \langle e_m, t_m - t\rangle\}$, the system state ss' at $T + t$ is equivalent to system state ss at T, that is, $ss' \equiv ss$ according to Definition 13. Therefore, during $T+t$ $(0 \leq t < t_{min})$ the proposed method need not generate any new system state for enumeration of reachable system states. After event e_i $(1 \leq i \leq m)$ such as $res\_time(e_i) = t_{min}$ becomes completed and new events become executable and triggered if any, a new system state is generated at a new time $T + t_{min}$ as the next system state of ss. □

Lemma 2 For any system state ss of the protocol $\mathcal{P}$, the proposed method enumerates all possible next system states of ss.

Proof: Assume that the protocol $\mathcal{P}$ stays in a system state $ss = (gs, ge)$ at time T, where $ge = \{\langle e_1, t_1\rangle, \ldots, \langle e_m, t_m\rangle\}$.

(1) Assume that only one event e_i becomes completed at a new time $T + t_i$ where t_i is the smallest of all t_h $(1 \leq h \leq m)$, and that $\mathcal{P}$ enters a global state gs' after completion of e_i. Let $ge'_0 = \{\langle e_h, t_h - t_i\rangle \mid \langle e_h, t_h\rangle \in ge, 1 \leq h \leq m, h \neq i\}$. This corresponds to $p = 1$ in (2-1) of the proposed method.

Because of Lemma 1, based on e_i and executable events after completion of e_i, the next system state is determined. In the following, kinds of activities of e_i are classified into Case 1, Case 2, Case 3, and Case 4 according to Definition 5. Then, which events become executable after completion of e_i are analyzed.

Case 1 (e_i is "counting timer" in process P_j, "addition' or "subtraction" on variables in P_j, or "reception" of messages from process P_k to P_j $(k \neq j)$)

The events which can become executable after completion of e_i are restricted to process-type events in P_j (say e') except for "counting timer" in P_j and "reception" of messages from P_j to P_k because of Cases 2-1 and 2-2 in Definition 7.

(a) If e' becomes triggered at $T + t_i$, then the next system state $ss'_1 = (gs', ge'_1)$ is generated at $T + t_i$, where $ge'_1 = ge'_0 \cup \{\langle e', time(e')\rangle\}$.

(b) Otherwise, the next system state $ss'_2 = (gs', ge'_2)$ is generated at $T + t_i$, where $ge'_2 = ge'_0$

Case 2 (e_i is "transmission" of message from process P_j to process P_k $(j \neq k)$)

The events which can become executable after completion of e_i are process-type events in P_j (say e'_1) except for "counting timer" in P_j and a channel-type event in channel C_{jk} (say e'_2) because of Cases 2-1, 2-2, 2-5 in Definition 7.

(a) If both e'_1 and e'_2 become triggered at $T + t_i$, then the next system state $ss'_1 = (gs', ge'_1)$ is generated at $T + t_i$, where $ge'_1 = ge'_0 \cup \{\langle e'_1, time(e'_1)\rangle, \langle e'_2, time(e'_2)\rangle\}$.

(b) If only e'_1 becomes triggered at $T + t_i$, then the next system state $ss'_2 = (gs', ge'_2)$ is generated at $T + t_i$, where $ge'_2 = ge'_0 \cup \{\langle e'_1, time(e'_1)\rangle\}$.

(c) If only e'_2 becomes triggered at $T + t_i$, then the next system state $ss'_3 = (gs', ge'_3)$ is generated at $T + t_i$, where $ge'_3 = ge'_0 \cup \{\langle e'_2, time(e'_2)\rangle\}$.

(d) Otherwise, the next system state $ss'_4 = (gs', ge'_0)$ is generated at $T + t_i$.

Case 3 (e_i is "starting timer" in process P_j)

The events which can become executable after completion of e_i are restricted to a process-type events in P_j (say e'_1) except for "counting timer" in P_j and "counting timer" in P_j (say e'_2) because of Cases 2-1, 2-2 and Case 2-4 in Definition 7.

(a) If both e'_1 and e'_2 become triggered at $T + t_i$ and "counting timer" in P_j (say e_k $(1 \leq k \leq m, k \neq i)$) is under execution, then the next system state $ss'_1 = (gs', ge'_1)$ is generated at $T + t_i$, where $ge'_1 = (ge'_0 - \{\langle e_k, t_k - t_i\rangle\}) \cup \{\langle e'_1, time(e'_1)\rangle, \langle e'_2, time(e'_2)\rangle\}$. Note that for a timer if "counting timer" in P_j is under execution, it is forced to stop by "starting timer" in P_j and again becomes triggered (see Remark 1).

(b) If both e'_1 and e'_2 become triggered at $T + t_i$ and "counting timer" in P_j is not under execution, then the next system state $ss'_2 = (gs', ge'_2)$ is generated at $T + t_i$, where $ge'_2 = ge'_0 \cup \{\langle e'_1, time(e'_1)\rangle, \langle e'_2, time(e'_2)\rangle\}$.

(c) If only e'_2 becomes triggered at $T + t_i$ and "counting timer" in P_j (say e_k $(1 \leq k \leq m, k \neq i)$) is under execution, then the next system state $ss'_3 = (gs', ge'_3)$ is generated at $T + t_i$, where $ge'_3 = (ge'_0 - \{\langle e_k, t_k - t_i\rangle\}) \cup \{\langle e'_2, time(e'_2)\rangle\}$.

(d) If only e'_2 becomes triggered at $T + t_i$ and "counting timer" in P_j is not under execution, then the next system state $ss'_4 = (gs', ge'_4)$ is generated at $T + t_i$, where $ge'_4 = ge'_0 \cup \{\langle e'_2, time(e'_2)\rangle\}$.

Case 4 (e_i is a channel-type event, that is, "message transfer" in channel C_{jk})

The events which can become executable after completion of e_i are restricted to "reception" of messages in process P_k (say e') because of Case 2-3 in Definition 7.

(a) If no event is under execution in P_k at $T + t_i$ and e' becomes triggered at $T + t_i$, then the next system state $ss'_1 = (gs', ge'_1)$ is generated at $T + t_i$, where $ge'_1 = ge'_0 \cup \{\langle e', time(e')\rangle\}$.

(b) If an event e'' is under execution in P_k at $T + t_i$ and $res\_time(e'')$ at $T + t_i$ is t'', then the next system state $ss'_2 = (gs', ge'_0)$ is generated at $T+t_i$. Note that since e'' is under execution in P_k at $T+t_i$, e' cannot become executable at $T+t_i$ (see Case 3 in Definition 8). Additionally, if e' can become triggered at $T + t_i + t''$, a new system state $ss'_3 = (gs'', ge'_3)$ is generated at $T + t_i + t''$, where $ge'_3 = \{\langle e_j, t_j - t''\rangle \mid \langle e_j, t_j\rangle \in ge'_0, e_j \neq e''\} \cup \{\langle e', time(e')\rangle\}$ (see (3) in Definition 9).

(c) Otherwise, the next system state $ss'_4 = (gs', ge'_0)$ is generated at $T + t_i$.

(2) We assume that more than one event becomes completed at $T + t$, where t is the smallest of all t_j $(1 \leq j \leq m)$, and that $\mathcal{P}$ enters a global state gs' after their completion. This corresponds to $p > 1$ in (2-1) of the proposed method. Let such events be $e_{i_1}, \ldots, e_{i_p}, \ldots, e_{i_q}$. Since each e_{i_p} $(1 \leq p \leq q)$ is executed in a process, a timer or a channel, all e_{i_p} $(1 \leq p \leq q)$ are independently executed. For each event e_{i_p} $(1 \leq p \leq q)$, executable events after completion of e_{i_p} are obtained in the same way as (1). Thus, at most one executable event becomes triggered in each process, timer or channel. Let a combination of such events triggered simultaneously be $(e'_1, \ldots, e'_r, \ldots, e'_s)$. The next system states are generated for all the combinations. □

Lemma 3 If two system states generated as the next system states are equivalent, the sequences of system states reachable from one of the two system states are the same as those reachable from the other.

Proof: Without loss of generality, the two equivalent system states are denoted by $ss = (gs, \{\langle e_1, t_1\rangle, \ldots, \langle e_m, t_m\rangle\})$ at time T and $ss' = (ge, \{\langle e_1, t_1 + T_c\rangle, \ldots, \langle e_m, t_m + T_c\rangle\})$ at time T' according to Definition 13. If the smallest residual time out of $t_1, \ldots, t_m$ is t_{min}, then that out of $t_1 + T_c, \ldots, t_m + T_c$ is $t_{min} + T_c$. We can proceed time T' by T_c without completion of any event $e_1, \ldots, e_m$ because $T_c < t_{min} + T_c$. Then, we obtain a system state $ss'' = (gs, \{\langle e_1, t_1\rangle, \ldots, \langle e_m, t_m\rangle\})$ at a new time $T'+T_c$. Since all components of ss'' are the same as those of ss, sequences of system states reachable from ss'' are the same as those reachable from ss. Therefore, this theorem holds. □

Theorem 1 The proposed method enumerates all possible system states reachable from the initial system state if the number of system states is finite.

Proof: The proof is recursively done with respect to time. The base step is obvious. In the induction step, we assume that a system state ss at time T, which is a next system state of the initial system state ss_0, is generated. Then, Lemma 2 guarantees that all the next system states at time T' $(\geq T)$ are generated. These system states are reachable from ss_0 according to Definition 12. We also assume that after a system state ss at T is generated, a system state ss' at time T'' which is equivalent to ss is generated. Lemma 3 guarantees that sequences of system states reachable from ss are the same as those from ss' and the latter is no longer necessary for generation. From the base step and the induction step, this theorem holds. □

Lemma 4 The number of system states is finite.

Proof: A system state consists of a global state and a set of pairs of an event and its residual time. Since the number of states in each process, the value of each variable, the capacity of each channel, and the number of kinds of messages are finite, then the number of global states is finite. On the other hand, since the number of timers and the operations in process are finite (see Assumption 4 and OP in Definition 1), and the number of kinds of message is finite, then the number of kinds of events is finite. Additionally, a residual time of each event is bounded by the maximum execution time out of all events. Since the maximum execution time is finite, the residual time of each event is finite. Therefore, the number of system states is finite.

Theorem 2 The proposed method enumerates all possible system states reachable from the initial system state.

Proof: It is obvious from Theorem 1 and Lemma 4. □

6. Conclusion

This paper has proposed a timed reachability analysis method for communication protocols with times needed to execute events and given proofs for correctness of the proposed method. Timeliness property of communication protocols can be verified by the proposed method because executable events are defined using times and parallel execution of such events are precisely analyzed. Compared with the conventional reachability analysis method, we expect that the proposed method has the following advantages: (1) The number of event sequences to be enumerated is decreased due to parallel execution of events, and (2) The length of event sequences is decreased due to introduction of equivalent system states. In order to show the expected advantages of the proposed method, experimental evaluation should be done as future work.

The proposed method assumes that for each event the upper bound is the same as the lower bound. The more general case is interesting from a viewpoint of practice. However, if the upper and lower bounds of a time needed to execute each event are given distinctly, the number of execution orders of events to be considered increases more than that in the proposed method. This consideration is reduced to the increase of branching event sequences. Although the complexity of enumeration of event sequences becomes high, the key ideas of the proposed method are applicable to this case. Further considerations on this more practical case also remain as future work.

References

1) Agrawal, G., Chen, B., Zhao, W. and Davari, S.: Guaranteeing Synchronous Message Deadlines with the Timed Token Protocol, *Proc. 12th Int'l. Conf. Distributed Computing Systems*, pp.468–475 (1992).
2) Alur, R. and Dill, D.: The Theory of Timed Automata, *LNCS 600 Real-Time: Theory in Practice*, Springer-Verlag, pp.45–73 (1992).
3) Berthomieu, B. and Diaz, M.: Modeling and Verification of Time Dependent Systems Using Time Petri Nets, *IEEE Trans. Softw. Eng.*, Vol.17, No.13, pp.259–273 (1991).
4) Brand, D. and Zafiropulo, P.: On Communicating Finite-state Machines, *Journal of the Association for Computing Machinery*, Vol.30, No.2, pp.323–342 (1983).
5) Godefroid, P.G. and Holzmann, G.J.: On the Verification of Temporal Properties, *Proc. 13th Symp. Protocol Specification, Testing and Verification*, pp.109–124 (1993).
6) Holzman, G.J., Godefroid, P. and Pirottin, D.: Coverage Preserving Reduction Strategies for Reachability Analysis, *Proc. 12th IFIP Symp. Protocol Specification, Testing and Verification*, pp.349–363 (1992).
7) Holzmann, G.J.: *Design and Validation of Computer Protocols*, Prentice Hall, NJ (1991).
8) Kakuda, Y., Kikuno, T. and Kawashima, K.: Automated Verification of Responsive Protocols Modeled by Extended Finite State Machines, *Real Time Systems Journal*, Vol.7, No.3, pp.275–289 (1994).
9) Kopetz, H. and Kakuda, Y. (Eds.): Responsive Computer Systems, *Dependable Computing and Fault-Tolerant Systems*, Vol.7, pp.17–26 (1993).
10) Liu, M.T.: Protocol Engineering, *Advances in Computers*, Vol.29, pp.79–195 (1989).
11) Nagano, S., Kakuda, Y. and Kikuno, T.: Timed Reachability Analysis and Its Application to Responsive Broadcasting Protocols, *Proc. 10th Int'l. Conf. Information Networking (ICOIN-10)*, pp.193–202 (1996).
12) Ostroff, J.S.: Verification of Safety Critical Systems Using TTM/RTTL, *Real-Time: Theory in Practice, LNCS*, Vol.600, Springer-Verlag, pp.573–602 (1991).
13) Yoneda, T., Shibayama, A., Schlingloff, H. and Clark, E.M.: Efficient Verification of Parallel Real-Time Systems, *Computer Aided Verification, LNCS*, Vol.697, Springer-Verlag, pp.321–332 (1993).

(Received September 25, 1995)
(Accepted March 12, 1996)

Shin'ichi Nagano was born in 1971. He received the M.E degree from Osaka Univ. in 1996. He is currently studying towards the Ph.D. degree in the Graduate School of Engineering Science in the same university. He have been engaged in research on verification of communication protocols. He is a student member of IEICE and IEEE.

Yoshiaki Kakuda was born in 1955. He received the M.S. and Ph.D. degrees from Hiroshima Univ. in 1980 and 1983, respectively. From 1983 to 1991, he was with Research and Development Laboratories, Kokusai Denshin Denwa Co., Ltd. (KDD), where he last held the position of Senior Research Engineer. From 1991 he has been an Associate Professor with the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka Univ. His current research interests include protocol engineering and responsive systems. He is a member of IPSJ, IEICE and IEEE. He received the Telecom. System Technology Award from Telecommunications Advancement Foundation in 1992.

Tohru Kikuno was born in 1947. He received the M.Sc. and Ph.D. degrees from Osaka Univ. in 1972 and 1975, respectively. He joined Hiroshima Univ. from 1975 to 1987. He is currently a Professor in the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka Univ., since 1990. His research interests include analysis and design of fault-tolerant systems, quantitative evaluation of software development process and design of testing procedure of communication protocols. He has been active in the program committee of many international conferences such as FTCS, PRFTS, COMPSAC, ICDCS, RDS. He is a member of IPSJ, IEICE, IEEE, ACM. He received the Paper Award from the Institute of Electronics, Information, and Communication Engineers of Japan in 1993.