Timed Reachability Analysis Method for EFSM-based Communication Protocols and Its Experimental Evaluation

Shin'ichi Nagano, Yoshinori Hatakeyama, Yoshiaki Kakuda and Tohru Kikuno
Department of Informatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, Osaka 560, Japan
E-mail: {s-nagano, kakuda, kikuno}@ics.es.osaka-u.ac.jp

Abstract

We propose a timed reachability analysis method in order to generate system states of two classes of EFSM-based communication protocols; one class $C1$ consists of events such that the lower and upper bounds are all the same and the other class $C2$ consists of events such that they may be different. The proposed method enumerates only sequences of system states that are obtained through parallel execution of all possible events. Then, in order to evaluate the efficiency of the proposed method, we develop a verification tool based on the proposed method, and then apply the tool to a broadcasting protocol. The experimental results show that the total number of states generated by the proposed method is much less than that by the previous method.

Keywords: *Communication protocols, verification, real-time, reachability analysis.*

1. Introduction

A real-time performance is indispensable for multimedia systems which process continuous data, such as sound and image data. In order to meet this demand, communication protocols must possess real-time processing capabilities [4, 7]. In order to design such communication protocols, verification methods for real-time properties are necessary. Although many verification methods have been proposed [1, 2, 4, 5, 7], they cannot deal with time, or put restrictions on time.

In order to resolve this problem, Kakuda et al.[3] have already proposed a verification method for real-time properties of communication protocols. This method assumes that execution times of all events are the same. However, this assumption is still strict for modeling practical communication protocols.

In this paper, a communication protocol is modeled by EFSMs and FIFO queues, in which the lower and upper bounds of execution time are specified for each event, and executions of events, especially executions of timers in process and message transfers in channel, are precisely defined. Under this modeling, we discuss two classes of communication protocols; one class $C1$ consists of events such that for each event the lower and upper bounds are the same and the other class $C2$ consists of events such that they may be different.

As fundamental techniques for verification of real-time properties of class $C1$, we have already proposed a timed reachability analysis method which generates system states [6]. However, class $C1$ is a subset of class $C2$, and is strict for modeling practical communication protocols. Class $C2$ can model more practical events such as message transfer in channel, for which execution time may vary within certain amount of time. Thus, we propose a timed reachability analysis method for class $C2$.

The proposed method enumerates only sequences of system states that can be obtained through parallel execution of all possible events. Since for class $C1$ the execution of each event completes at a time, parallel execution of all possible events at any time is easily obtained by the proposed method. On the other hand, for class $C2$ the execution of each event may complete at any time between the lower and upper bounds of other events which are under execution. The number of potential combinations of parallel execution of all possible events increases more than that for the class $C1$. In order to decrease this number, we introduce a new condition for determining equivalence of system states.

Then we have developed a verification tool based

0-8186-7453-9/96 $05.00

on the proposed method and then applied the tool to a broadcasting protocol. In the experiment, we have obtained the total numbers of states generated by the proposed method and the previous method. The experimental results show that the total number of states generated by the proposed method is much less than that by the previous method.

2. Communication Protocol

A communication protocol, shortly a protocol, consists of communication entities (called processes) and communication channels (called channels) [6]. This paper models each process and channel by an EFSM and an FIFO queue whose capacity is finite, respectively.

Figure 1 shows an example protocol which consists of three processes P_1, P_2, P_3 and four channels C_{13}, C_{31}, C_{23}, C_{32}. Each process and channel is modeled by an EFSM and an FIFO queue shown in Figure 2. $-msg$ and $+msg$ denote message transmission and reception, respectively. An arrow between two circles denotes a state transition, to which both an operation to be executed in process and an enabling condition for its execution are assigned as a label. The predicate $v_T^i = v_{TO}^i$ $(i = 1, 2)$ denotes a condition for time-out of the timer in P_i.

Definition 1 Events in a communication protocol consist of process-type events in process P_i and channel-type events 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, that is, (2-1) "message transmission" from P_i to P_j, (2-2) "message reception" 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 activity: (3) "message transfer" in channel C_{jk}. □

Definition 2 Based on execution times of events, we classify communication protocols into three kinds of classes:

Class $C0$: For any event, its execution time is not specified.

Class $C1$: For any event, the lower and upper bounds of its execution time are the same.

Class $C2$: For any event, the lower and upper bounds of its execution time may be different. □

Since conventional reachability analysis methods do not consider execution times, they can be applied only

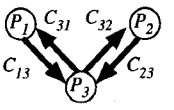


Figure 1. Example of protocol

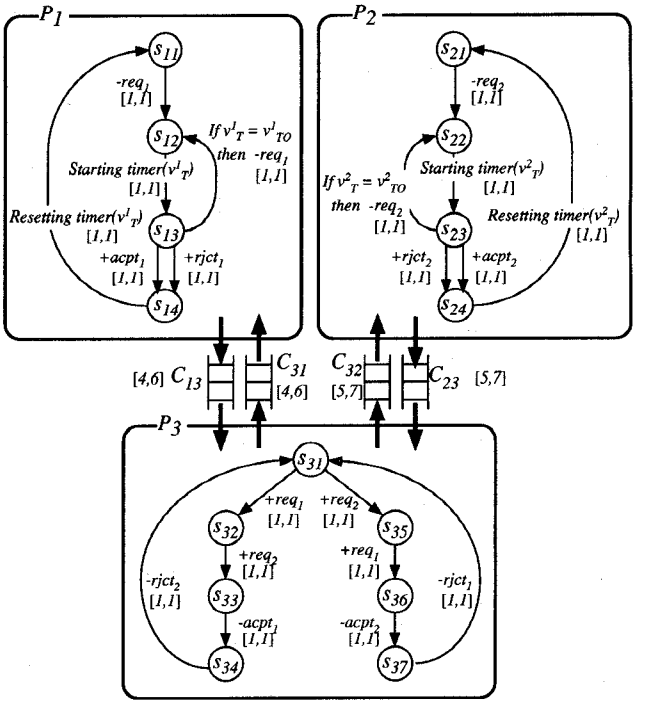


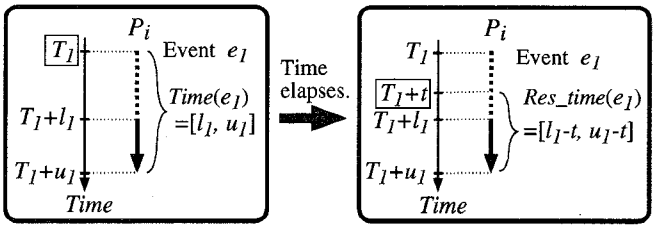
Figure 2. Modeling of processes and channels

for class $C0$. In this paper, we propose a timed reachability analysis method for class $C2$.

In a protocol model shown in Figure 2, the lower and upper bounds for each process-type event except for "counting timer" are one time unit. On the other hand, the lower and upper bounds for each channel-type event in channels C_{13} and C_{31} are four and six time units, and those of each channel-type event in C_{23} and C_{32} are five and seven time units.

Execution of each event is described using Figure 3. Consider an event e_1 in process P_i which has the lower bound l_1 and the upper bound u_1 $(0 < l_1 \leq u_1)$ of a time needed to execute. Intuitively, if e_1 is triggered at the current time T_1, then it e_1 completed between time $T_1 + l_1$ and $T_1 + u_1$ without being forced to stop. We represent the execution time of e_1 by $time(e_1) = [l_1, u_1]$, as shown in Figure 3 (a).

Next, let the current time be $T_1 + t$ $(0 \leq t < u_1)$. If event e_1 is not completed at $T_1 + t$, then we say



(a) Current time is T_1. (b) Current time is T_1+t.

Figure 3. Execution of event

that e_1 is under execution at $T_1 + t$. At that time, the lower bound l'_1 and the upper bound u'_1 of the residual time for event e_1 are defined as follows: $l'_1 = l_1 - t$ if $l_1 - t \geq 0$, and $l'_1 = 0$ otherwise. On the other hand, $u'_1 = u_1 - t$. We represent the residual time of e_1 by $res\_time(e_1) = [l'_1, u'_1]$, as shown in Figure 3 (b).

Since timers are essential for real-time functions, it is important to model and analyze execution of timers accurately.

Definition 3 Although timers are attached to process, they are executed independent of the process, and execution of each timer is represented by "counting timer". Additionally, for each timer, a pair of timer variable v_T and time-out value v_{TO} is given. The predicate on each timer is defined as $v_T = v_{TO}$, and the lower and upper bounds of execution time for "counting timer" are set to v_{TO}.

There are three kinds of events for each timer: "starting timer", "counting timer" and "resetting timer". By completion of "starting timer", zero is assigned to v_T. Then the value of v_T is incremented by "counting timer" along with progress of time. If the value of v_T equals to that of v_{TO} at time T, then "counting timer" completes at T and a predicate on timer becomes true at T. Next, if "resetting timer" completes at T, then "counting timer" is forced to stop at T and zero is assigned to v_T. □

Assume that execution of an event has completed. Then, execution of other events can start if some conditions are satisfied. Such events are said to be executable. When more than one event is executable in a process, one of them must be selected to become triggered. Conditions for executable events and triggered events are omitted due to limited space in this paper, and they are defined in [6].

3. System State

A system state is introduced in order to describe not only states of all processes and channels but also parallel execution of events at a time.

Definition 4 A system state ss at the current global time T in a communication protocol $\mathcal{P}$ is defined by $ss = (gs, ge)$, where $gs = (ps_1, \ldots, ps_N, cs_{12}, \ldots, cs_{NN-1})$ is so called global state of $\mathcal{P}$, that is, state of all processes and channels in $\mathcal{P}$ and $ge = \{\langle e_1, l_1, u_1\rangle, \ldots, \langle e_q, l_q, u_q\rangle\}$. Additionally, l_p and u_p $(1 \leq p \leq q)$ denote lower and upper bounds of a residual time for event e_p. □

Definition 5 We define the next system state of a system state (let it be $ss_i = (gs_i, ge_i)$) at time T depending on whether ss_i is the initial system state or not.

First, we consider the next system state when ss_i is the initial system state. Then $ge_i = \phi$. Assume that E_{trg} is a set of events $e_1, \ldots, e_q$ which are triggered at the initial time. Let $ge_{i+1} = \{\langle e_p, l_p, u_p\rangle \mid e_p \in E_{trg}, time(e_p) = [l_p, u_p]\}$. The next system state of ss_i at the initial time is defined as $ss_{i+1} = (gs_{i+1}, ge_{i+1})$ where $gs_{i+1} = gs_i$. We represent this binary relation as $ss_i \vdash_f ss_{i+1}$ where $f = (\phi, E_{trg}, 0)$.

Next, we consider the next system state when ss_i is not the initial system state. Let $ge_i = \{\langle e_1, l_1, u_1\rangle, \ldots, \langle e_p, l_p, u_p\rangle, \ldots, \langle e_q, l_q, u_q\rangle\}$. Assume that E_{cplt} is a set of events $e_{p_1}, \ldots, e_{p_r}, \ldots, e_{p_s}$ which are completed at the current time $T + t$, and that E_{trg} is a set of events $e'_1, \ldots, e'_u, \ldots, e'_v$ which are triggered at $T + t$ if any. Then, let $ge_{i+1} = \{\langle e_p, l'_p, u'_p\rangle \mid \langle e_p, l_p, u_p\rangle \in ge_i, e_p \notin E_{cplt}, l'_p = l_p - t \geq 0, u'_p = u_p - t \geq 0\} \cup \{\langle e'_u, l'_u, u'_u\rangle \mid e'_u \in E_{trg}, time(e'_u) = [l'_u, u'_u]\}$. The next system state of ss_i is defined as $ss_{i+1} = (gs_{i+1}, ge_{i+1})$, where gs_{i+1} is a global state changed from gs_i by completion of events $e_{p_1}, \ldots, e_{p_r}, \ldots, e_{p_s}$. We represent this binary relation as $ss_i \vdash_f ss_{i+1}$ where $f = (E_{cplt}, E_{trg}, t)$.

In the following, the binary relation $\vdash_f$ is just denoted by $\vdash$. □

Figure 4 shows several typical system states of the example protocol. Now we consider the case where execution of processes P_1 and P_2 start simultaneously. The initial system state of the example protocol is $ss_0 = (gs_0, \phi)$ at the initial time. After making "message transmission" $-req_1$ and $-req_2$ triggered, the system state at the initial time is changed to $ss_1 = (gs_1, ge_1)$ where $gs_1 = gs_0$ and $ge_1 = \{\langle -req_1, 1, 1\rangle, \langle -req_2, 1, 1\rangle\}$. After completing $-req_1$ and $-req_2$, and making four events, $starting\ timer(v_T^1)$, $starting\ timer(v_T^2)$, and "message transfer" (req_1) and (req_2) triggered, the system state at time 1 is $ss_2 = (gs_2, ge_2)$ where $ge_2 = \{\langle starting\ timer(v_T^1), 1, 1\rangle, \langle starting\ timer(v_T^2), 1, 1\rangle, \langle (req_1), 4, 6\rangle, \langle (req_2), 5, 7\rangle\}$. For ss_0, ss_1 and ss_2, $ss_0 \vdash ss_1$ and $ss_1 \vdash ss_2$ hold.

$ss_0 = (gs_0, \phi)$
$ss_1 = (gs_1, \{\langle -req_1, 1, 1\rangle, \langle -req_2, 1, 1\rangle\})$
$ss_2 = (gs_2, \{\langle (req_1), 4, 6\rangle, \langle (req_2), 5, 7\rangle, \langle starting\ timer(v_T^1), 1, 1\rangle, \langle starting\ timer(v_T^2), 1, 1\rangle\})$
$ss_3 = (gs_3, \{\langle (req_1), 3, 5\rangle, \langle (req_2), 4, 6\rangle, \langle counting\ timer(v_T^1), 25, 25\rangle, \langle counting\ timer(v_T^2), 25, 25\rangle\})$

Figure 4. Example of system state

Next, equivalence conditions of system states are indispensable in order to guarantee that the proposed method terminates. Furthermore, by cutting off sequences of system states following the equivalent system state, the number of system states generated by the proposed method can be dramatically decreased.

Definition 6 Assume that a system state $ss_i = (gs_i, \{\langle e_{i1}, l_{i1}, u_{i1}\rangle, \ldots, \langle e_{ip}, l_{ip}, u_{ip}\rangle\})$ at time T_i has already been generated and a system state $ss_j = (gs_j, \{\langle e_{j1}, l_{j1}, u_{j1}\rangle, \ldots, \langle e_{jq}, l_{jq}, u_{jq}\rangle\})$ at time T_j is newly generated. If ss_i and ss_j satisfy either conditions $E1$ and $E2$, or conditions $E1$ and $E3$, then we say that ss_j is equivalent to ss_i.

Condition $E1$: $gs_i = gs_j, p = q$, and $\forall k\ [\ e_{ik} = e_{jk}\ ]$

Condition $E2$: $\exists T_c \forall k\ [\ l_{ik} - l_{jk} = T_c \wedge u_{ik} - u_{jk} = T_c\ ]$

Condition $E3$: $\exists T_c \forall k$

Case 1 (e_{ik} is any event except for "counting timer"):

$$l_{ik} - l_{jk} = T_c \wedge u_{ik} - u_{jk} = T_c,$$

Case 2 (e_{ik} is "counting timer"):

e_{ik} is forced to stop by "resetting timer" in any system state following ss_i, and $[\ l_{ik} \leq l_{jk} + T_c\ \wedge\ u_{ik} \leq u_{jk} + T_c.\ ]$ □

Condition $E2$ is strict for class $C2$ since we must consider for each event all possible completions between its lower and upper bounds. Thus, we define a new condition $E3$ in this paper to decrease the number of generated system states.

4. New Verification Method

In this section, we propose a timed reachability analysis method to generate system states for class $C2$ of communication protocols. The proposed method can be also applied to class $C1$ since class $C1$ is a subset of class $C2$. Although a timed reachability analysis method for class $C1$ has been shown in [6], it cannot be applied to class $C2$ since all possible completions between the lower and upper bounds for each event must be considered.

4.1. Timed Reachability Analysis Method

The following two kinds of data are the input for the timed reachability analysis method: (1) A communication protocol $\mathcal{P}$ to be analyzed, and (2) $time(e_i) = [l_i, u_i]$ for each event e_i except for "counting timer". The proposed method generates system states according to these input.

We recursively describe the proposed method in the following.

(1. Base Step) Assume that a communication protocol $\mathcal{P}$ stays in the initial system state $ss_0 = (gs_0, \phi)$ at the initial time T_0.

(1-1) Compute events $e_1, \ldots, e_m$ which become executable and triggered in ss_0.

(1-2) Generate the next system state $ss_1 = (gs_1, ge_1)$ at T_0, where $gs_1 = gs_0$ and $ge_1 = \{\langle e_i, l_i, u_i\rangle \mid 1 \leq i \leq m, time(e_i) = [l_i, u_i]\}$.

(2. Induction Step) Assume that a communication protocol $\mathcal{P}$ stays in a system state $ss = (gs, ge)$ at the current time T ($\geq T_0$), where $ge = \{\langle e_1, l_1, u_1\rangle, \ldots, \langle e_q, l_q, u_q\rangle\}$.

(2-1) Compute t which is the smallest among l_p ($1 \leq p \leq q$), and proceed time by t, and set the current time to $T + t$. If $t > 0$, then let $E_1 = \{\langle e_p, l_p - t, u_p - t\rangle \mid \langle e_p, l_p, u_p\rangle \in ge\}$, and $E_2 = \{\langle e_p, l'_p, u'_p\rangle \in E_1 \mid l'_p = 0\}$.

If $t = 0$, then proceed time by 1, and set the value of t and the current time to 1 and $T + 1$, respectively. Additionally, let $E_1 = \{\langle e_p, l_p - 1, u_p - 1\rangle \mid \langle e_p, l_p, u_p\rangle \in ge, l_p > 0\} \cup \{\langle e_p, 0, u_p - 1\rangle \mid \langle e_p, l_p, u_p\rangle \in ge, l_p = 0\}$, and $E_2 = \{\langle e_p, l'_p, u'_p\rangle \in E_1 \mid l'_p = 0\}$.

(2-2) Select events from E_2, and make all combinations of the selected events, (e_{p_1}, ..., e_{p_r}, ..., e_{p_s}). Note that event e_{p_r} with $l_{p_r} = u_{p_r} = 0$ must be inevitably selected in these combinations. Additionally, if e_{p_r} is "message transfer" in channel C_{jk} and E_2 includes "message transfer" e ($\neq e_{p_r}$) in C_{jk} which was triggered before e_{p_r}, then e_{p_r} must

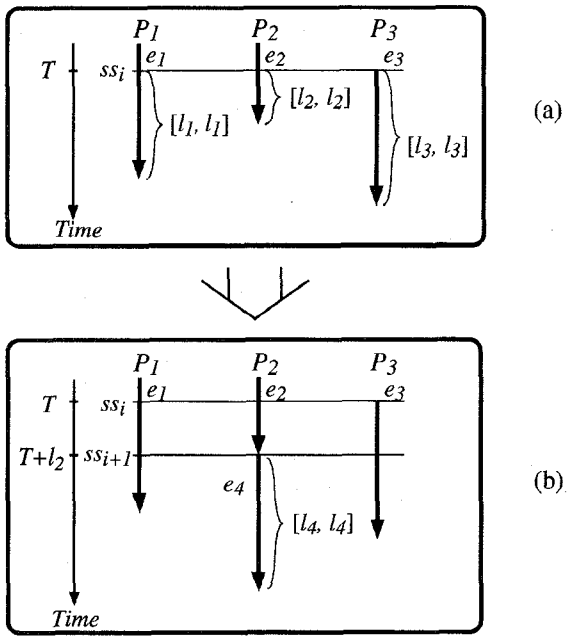


Figure 5. Construction of event sequence chart

not be selected because of FIFO delivery of messages in channel.

For each combination $(e_{p_1}, \ldots, e_{p_r}, \ldots, e_{p_s})$, make each e_{p_r} $(1 \leq r \leq s)$ simultaneously completed in a process or a channel at $T + t$, and perform substeps (2-3) through (2-5). This case is called *type*2 branching. Note that if event e_{p_r} is "resetting timer" in process P_i and for "counting timer" in P_i (let it be e) $\langle e, l, u \rangle \in E_1$, then e is forced to stop and $E_1 = E_1 - \{\langle e, l, u \rangle \mid res\_time(e) = [l, u]\}$.

(2-3) Let E_3 be a set of all events that can be executable after completion of events $e_{p_1}, \ldots, e_{p_r}, \ldots, e_{p_s}$. For each process, timer, or channel, select one event from E_3. Let it be e'_u $(1 \leq u \leq v)$. For each combination of executable events $(e'_1, \ldots, e'_u, \ldots, e'_v)$, make each e'_u $(1 \leq u \leq v)$ simultaneously triggered in a process, a timer, or a channel at $T + t$, and perform substeps (2-4) through (2-5). This case is called *type*1 branching.

(2-4) Let $ge'_w = \{\langle e_p, l'_p, u'_p \rangle \in E_1 \mid 1 \leq p \leq q, p \neq p_1, \ldots, p \neq p_s\} \cup \{\langle e'_u, l'_u, u'_u \rangle \mid 1 \leq u \leq v, time(e'_u) = [l'_u, u'_u]\}$. Generate the next system state ss'_w $(1 \leq w \leq x)$ of ss, where $ss'_w = (gs'_w, ge'_w)$ at $T + t$ and gs'_w is a global state which $\mathcal{P}$ enters from gs.

(2-5) If $ge'_w = \phi$, then stop generating system states following ss'_w. Next, if ss'_w and some system state ss'' that has been already generated satisfy equivalence conditions $E1$ and $E2$ in Definition 6, then stop generating system states following ss'_w. This case is called *type*1 merging. In addition, if ss'_w and ss'' satisfy equivalence conditions $E1$ and $E3$, then stop generating system states following ss'_w. This case is called *type*2 merging. Otherwise, call (2. Induction Step) with system state ss' at the current time $T + t$. □

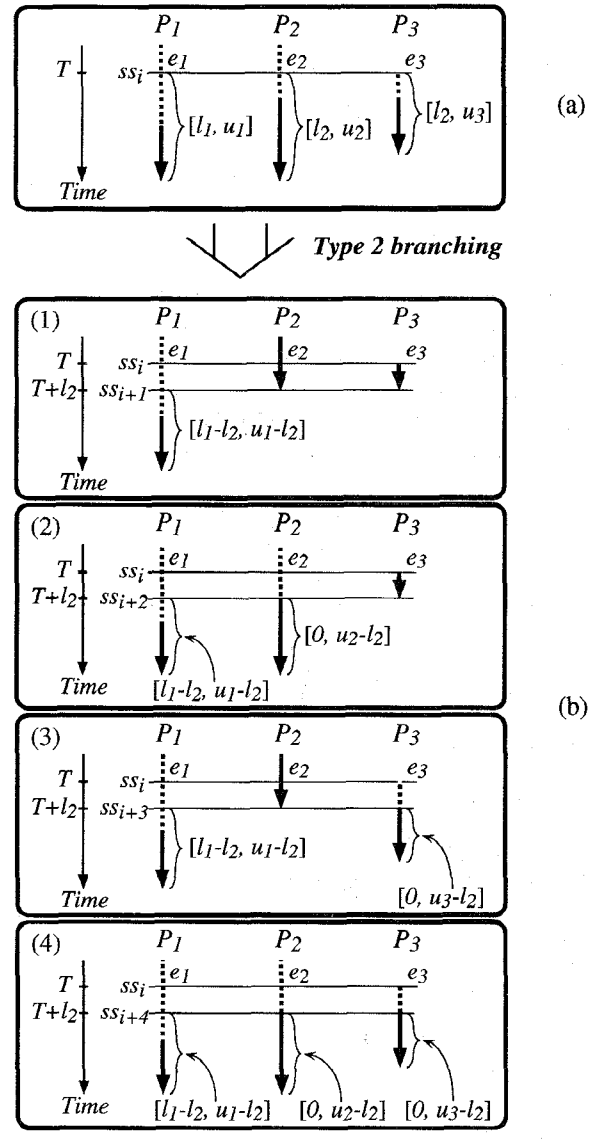


Figure 6. *Type*2 branching

An example of an event sequence chart in Figure 5 briefly explains how the proposed method generates system states. An event sequence chart represents

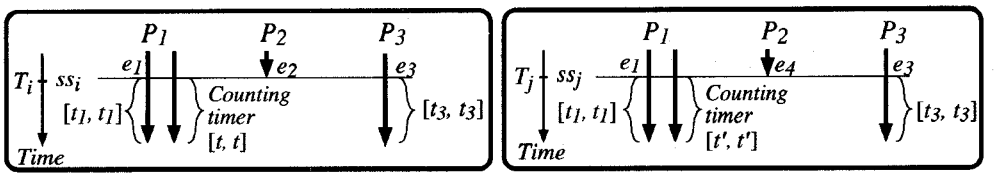


Figure 7. Illustration for *type*2 merging

graphically an event sequence. In Figure 5 (a), the processes and channels and the system states are drawn on the horizontal axis and the vertical axis, respectively, and system states corresponding exactly to horizontal lines are enumerated.

Now assume that a system state at the current time T is $ss_i = (gs_i, ge_i)$ where $ge_i = \{\langle e_1, l_1, l_1\rangle, \langle e_2, l_2, l_2\rangle, \langle e_3, l_3, l_3\rangle\}$ as shown in Figure 5 (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 l_2 is the smallest of three lower bounds l_1, l_2 and l_3, and that only event e_4 is executable after completion of e_2. After l_2 time units elapse, e_4 becomes triggered and a new system state ss_{i+1} is generated at time $T + l_2$. Then, ss_{i+1} is defined by $ss_{i+1} = (gs_{i+1}, ge_{i+1})$ where $ge_{i+1} = \{\langle e_1, l_1 - l_2, l_1 - l_2\rangle, \langle e_4, l_4, l_4\rangle, \langle e_3, l_3 - l_2, l_3 - l_2\rangle\}$ as shown in Figure 5 (b). Thus, the next system state is generated based on residual time of each event that is under execution at the current time.

4.2. Branching

The branching from an event sequence chart to event sequence charts is classified into two cases : *type*1 and *type*2. *Type*2 branching is newly introduced in this paper, since in class $C2$ we must consider for each event all possible completions between its lower and upper bounds. Please note that although *type*1 branching can be performed for classes $C1$ and $C2$, *type*2 branching can be performed only for class $C2$.

First, in *type*1 branching, if execution of more than one event in a process can start at the same time, then we must consider the execution order of their events. Please refer to [6] for the details.

Second, in *type*2 branching, we must consider for each event all possible completions between its lower and upper bounds. Furthermore, if more than one event is under execution at the current time, then we must also consider all combinations of their events. Consider an example sequence chart as shown in Figure 6 (a). Let $e_i (i = 1, 2, 3)$ be an event under execution in process P_i at time T. Assume that the lower bound of residual times for e_2 and e_3 are the same, and that no execution of event starts after completion of e_2 and e_3. In this case, the following four new systems states are generated at time $T + l_2$: (1) $ss_{i+1} = (gs_{i+1}, ge_{i+1})$, where $ge_{i+1} = \{\langle e_1, l_1 - l_2, u_1 - l_2\rangle\}$, (2) $ss_{i+2} = (gs_{i+2}, ge_{i+2})$, where $ge_{i+2} = \{\langle e_1, l_1 - l_2, u_1 - l_2\rangle, \langle e_2, 0, u_2 - l_2\rangle\}$, (3) $ss_{i+3} = (gs_{i+3}, ge_{i+3})$, where $ge_{i+3} = \{\langle e_1, l_1 - l_2, u_1 - l_2\rangle, \langle e_3, 0, u_3 - l_2\rangle\}$, and (4) $ss_{i+4} = (gs_{i+4}, ge_{i+4})$, where $ge_{i+4} = \{\langle e_1, l_1 - l_2, u_1 - l_2\rangle, \langle e_2, 0, u_2 - l_2\rangle, \langle e_3, 0, u_3 - l_2\rangle\}$. Thus, the event sequence chart is branching out to four event sequence charts as shown in Figure 6 (b).

4.3. Merging

The merging of two event sequence charts is classified into two cases : *type*1 and *type*2. Equivalence conditions for *type*1 merging are $E1$ and $E2$ in Definition 6, and equivalence conditions for *type*2 merging are $E1$ and $E3$. Condition $E2$ requires that for each event difference between two lower bounds, and difference between two upper bounds must be the same. Thus, *type*1 merging is not much frequently performed for class $C2$. In this paper, we introduce *type*2 merging with equivalence condition $E3$ which relaxes condition $E2$. So, the number of system states generated by the proposed method can be dramatically decreased. Please note that both *type*1 and *type*2 mergings can be performed for classes $C1$ and $C2$. Please refer to [6] for details of *type*1 merging.

Next, *type*2 merging is explained using an example. Consider two system states ss_i at time T_i and ss_j at time T_j in two different event sequence charts as shown in Figure 7. One is $ss_i = (gs_i, ge_i)$ where $ge_i = \{\langle e_1, t_1, t_1\rangle, \langle e_3, t_3, t_3\rangle, \langle counting\ timer, t, t\rangle\}$, and no event is triggered after completion of event e_2. The other is $ss_j = (gs_j, ge_j)$ where $ge_j = \{\langle e_1, t_1, t_1\rangle, \langle e_3, t_3, t_3\rangle, \langle counting\ timer, t', t'\rangle\}$, and no event is triggered after completion of event e_4. Assume that after ss_i is generated, ss_j is newly generated, and that relations $t < t'$ and $gs_i = gs_j$ hold. Assume also that in the event sequence following ss_i, *counting timer* is forced to stop by completion of *resetting timer* in ss_{i+3} at $T_{i+3} = T_i + t''$ $(t'' < t)$. Although the residual times t and t' of *counting timers* are different, ge_i is equal to ge_j except for the lower

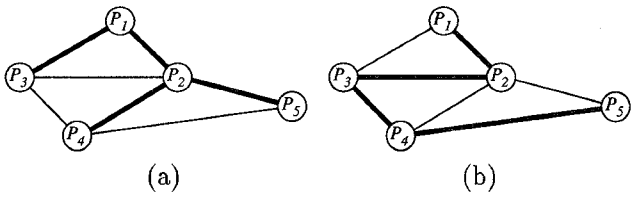


Figure 8. Example of tree

and upper bounds t and t' of *counting timer*. On the other hand, since $t < t'$ holds, in the event sequence following ss_j *counting timer* is forced to stop at time $T_{j+3} = T_j + t''$ and two event sequences following ss_{i+3} and ss_{j+3} become the same. Then, two event sequences are merged at ss_i by stopping further expansion of event sequence following ss_j.

5. Experimental Evaluation

We have developed a verification tool (about 6500 lines) with C language on SONY NEWS-5000 with 64 M bytes memories. The following two kinds of data are the input for the tool: (1) Text files which describe a communication protocol to be verified, and (2) A method selected out of the proposed method and the previous method. This tool can provide all states generated by the proposed method and all states generated by the previous method. Note that in this tool we have not yet implemented equivalence condition $E3$ in Definition 6.

5.1. Experiments

In this subsection, we apply the tool to a broadcasting protocol. A broadcasting protocol is a protocol which transmits the same data from the root process in a tree on the network to all other processes [8]. The number of states and transitions in each process are 10 and 13, respectively.

Experiment 1 We compare the total number of states generated by the proposed method with that by the previous method. We assume that data are transmitted from root process P_1 along with the spanning tree on the network. We also assume that execution times of all events except for "counting timer" are one time unit for the proposed method.

We apply the tool to four kinds of networks: (1) three processes, (2) five processes, (3) seven processes, and (4) nine processes. Assume that a spanning tree on each of these networks is a binary tree. Table 1 shows the result of experiment 1. In Table 1, M_{prop} and M_{prev} denote the proposed method and the previous method, respectively.

Table 1. Result of Experiment 1

# of processes		3	5	7	9
# of states	M_{prop}	34	58	61	82
	M_{prev}	144	1584	16707	–

Table 2. Result of Experiment 2

# of processes		3	5	7	9
# of states	M_{prop}	56	110	171	225
	M_{prev}	91	337	1177	4568

Experiment 2 Through applying each of the proposed method and the previous method to two kinds of networks, we examine relation between parallelism of processes in a network and the total number of generated states. Figures 8 (a) and (b) show spanning trees on the network to be used in this experiment. We assume that in Figure 8 (b) data are transmitted from process P_1 along with channels which are represented by bold lines. Each process P_i $(i > 1)$ in Figure 8 (b) is dependent on the behavior of process P_{i-1}. On the other hand, processes P_2 and P_3 in Figure 8 (a) are dependent on that of process P_1, and processes P_4 and P_5 are dependent on that of P_2. However, P_2 and P_3, and P_4 and P_5 can concurrently execute, respectively. Therefore, we can conclude that the degree of parallelism of processes in Figure 8 (a) is higher than that in Figure 8 (b).

In this experiment, we also assume that execution times of all events except for "counting timer" are one time unit for the proposed method also. Table 2 shows the result of experiment 2.

5.2. Evaluation for Class $C1$

In this subsection, we evaluate efficiency of the proposed method for class $C1$, according to experimental results shown in Tables 1 and 2.

(1) The number of states generated by the proposed method becomes much less than that by the previous method. While the previous method must consider all possible sequential temporal orders of event execution, the proposed method considers all possible parallel execution of events. This is the reason why the number of generated states can be dramatically decreased.

(2) The difference between the number of states generated by the proposed method and that by the previous method in Figure 8 (a) is much larger than their difference in Figure 8 (b). This result implies that the proposed method is efficient for the network where the degree of parallelism of processes

is relatively high like Figure 8 (a). Furthermore, the number of states generated by the proposed method becomes small for both networks.

5.3. Evaluation for Class $C2$

Generally speaking, if for each event the lower and upper bounds of its execution time may be different, then the total number of states generated by the proposed method may increase. In addition, equivalence condition of system states is strict (see condition $E2$ in Definition 6). If at least one of their bounds is not the same, then two system states cannot be determined to be equivalent. This case is often seen especially for "counting timer". Thus, the total number of generated system states may increase.

In order to resolve this problem, we have defined a new equivalence condition $E3$ in Definition 6. In fact, we have applied condition $E3$ to the broadcasting protocol which consists of three processes and four channels. Note that we have manually generated system states since we have not yet implemented condition $E3$. We assume that for each process-type event except for "counting timer" the lower and upper bounds of its execution time are one time unit, and that for each channel-type event the lower and upper bounds of its execution time are 20 time units and 22 time units, respectively. Then, 435 system states are generated without condition $E3$. However, when we apply the proposed method with condition $E3$, the number of system states is decreased to 193. Thus, we may conclude that by introducing condition $E3$, the proposed method is efficient for class $C2$ of communication protocols, and the number of generated system states can be decreased.

6. Conclusion

In this paper, we have proposed a timed reachability analysis method for class $C2$ of communication protocols, where for each event the lower and upper bounds of its execution time may be different. The experimental results show that the total number of states generated by the proposed method is much less than that by the previous method, and that the proposed method is much more efficient for a communication protocol with high degree of parallelism in process executions.

In future work, we are planning to introduce compression techniques for preserving data of each state to the verification tool. Then, we believe that the tool based on the proposed method is applicable to more large-scaled communication protocols. We are also planning to verify real-time properties, such as timing constraints of event executions, of practical communication protocols based on the proposed method.

References

[1] B. Berthomieu and M. Diaz: "Modeling and verification of time dependent systems using time Petri Nets," IEEE Trans. Softw. Eng., Vol 17, No.13, pp.259-273, 1991.

[2] P. G. Godefroid and G. J. Holzmann: "On the verification of temporal properties," Proc. 13th Symp. Protocol Specification, Testing and Verification, pp.109-124, 1993.

[3] Y. Kakuda, T. Kikuno and K. Kawashima: "Automated verification of responsive protocols modeled by extended finite state machines," Real Time Systems Journal, Vol.7, No.3, pp.275-289, 1994.

[4] H. Kopetz and Y. Kakuda(eds.): "Responsive Computer Systems," Dependable Computing and Fault-Tolerant Systems, Vol.7, Springer-Verlag, pp.17-26, 1993.

[5] M. T. Liu: "Protocol Engineering," Advances in Computers, Vol.29, pp.79-195, 1989.

[6] S. Nagano, Y. Kakuda and T. Kikuno: "Timed reachability analysis method for communication protocols modeled by extended finite state machines," Trans. IPSJ, Vol.37, No.5, pp.698-710, 1996.

[7] J. S. Ostroff: "Verification of safety critical systems using TTM/RTTL," LNCS 600 Real-time: Theory in Practice, pp.573-602, 1991.

[8] A. Segall: "Distributed network protocols," IEEE Trans. Information Theory, IT-29, 1, pp.23-35, 1983.