ELSEVIER

Computer Communications 21 (1998) 460–469

computer communications

Specification, validation, and verification of time-critical systems

Shiuh-Pyng Shieh*, Jun-Nan Chen

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan

Received 19 April 1997; received in revised form 4 August 1997; accepted 4 August 1997

Abstract

In this paper, we propose a new formalism, named the Timed Communicating Finite State Machine (Timed CFSM), for specifying and verifying time-critical systems. Timed CFSM preserves the advantages of CFSM, such as the ability to express communication, synchronization and concurrency in computer systems. A given time-dependent specification can be formalized as a Timed CFSM, from which the reachability graph is constructed to verify the correctness of the specification. To cope with the space explosion problem from which all reachability analysis methods suffer, we propose a space reduction algorithm to meet the space constraint of the verification environment.

Keywords: Time-critical systems; Specification; Validation; Verification; Reachability analysis; The space explosion problem; Path approach

1. Introduction

Computer systems are increasingly affecting nearly every aspect of our lives. They control aircraft, shut down nuclear power reactors in emergencies, keep our telephone systems running, monitor hospital patients, carry out financial transactions, and support multimedia applications [18]. All of these systems exhibit time constraints. These systems with time constraints are called *real-time* systems. Real-time systems are safety-critical especially when they must satisfy stringent time constraints. Otherwise, they will result in catastrophic risks. This type of system is referred to as a *time-critical system* as it emphasizes the potential risks of not meeting time constraints.

Time-critical systems are difficult to design and analyze owing to their dual requirements: both functionality and timeliness. The difficulty arises from the necessity to model the dependence of system behaviors on timing variables. Specifying a time-critical system is meant to express the dependence. Verifying a time-critical system is meant to prove the correctness of the dependence. In previous work, the dual-language model was proposed, which is composed of a state-based language and an assertional language [1,13]. The state-based language is used to specify functional components of time-critical systems, while the assertional language is used to formalize timing requirements. Finite State Machines (FSMs) [2,15] and Petri Nets [6,16,17] are representative examples of state-based languages. To include the ability of specifying timeliness explicitly, state-based languages are extended to the timed versions of original formalisms, for example, the Timed Petri Net. On the other hand, assertional languages are categorized as first-order logic with quantities, for example Hoare Logic [7,8] and RTL_1 [12], and as temporal logic with model operators, for example, TIL [3] and RTTL [15]. Given a specification from the dual-language model, we could verify the correctness of time-critical systems by validating assertions. However, the validation is a theorem-proving process which could be either pen-and-paper work or concerned with reasoning in artificial intelligence. There are two phases in the proof process: first, timing properties is described as "axioms"; second, a desired assertion is proved to be valid or not. The proof process is difficult to automate. Therefore uniform language models [14] are proposed to provide both simplicity and automation. In uniform language models, both functional components and time properties are described in the uniform language. Meanwhile verification of the system is a type of reachability analysis [6,11], rather than a theorem-proving technique. However, reachability analysis suffers from the space explosion problem, as we have discovered in the field of protocol engineering, which

* Corresponding author. Tel.: +886 35 731876; fax: +886 35 724176; e-mail: ssp@csie.nctu.edu.tw

PII S0140-3664(97)00145-X

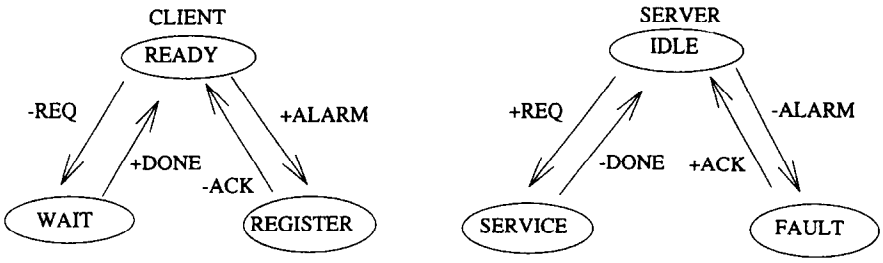

Fig. 1. An example of the CFSM.

means that the space complexity of reachability analysis is exponentially proportional to the number of global states. To date there has been little research about the space explosion problem for the validation of time-critical systems.

We propose a new formalism, named the Timed Communicating Finite State Machine (Timed CFSM) which is able to model time-critical systems. Timed CFSM preserves the advantages of CFSM, for example, the ability to express communication, synchronization and concurrency in computer systems.

Verification of Timed CFSM uses the technique of reachability analysis, which suffers from the space explosion problem. To resolve the space explosion problem, we propose an algorithm based on the path approach which produces only partial graphs, reducing the space requirement. Compared to the probabilistic approach for the verification of the Timed Communicating State Machine (TCSM) proposed in Ref. [16], which could achieve only partial verification for the TCSM, the path approach is able to verify global states exhaustively, while not tampering with the space explosion problem.

Recently, the increase in the number of multimedia applications has overemphasized the verification of timing properties. Previous formalisms have failed to specify multimedia applications, not only in functionality but also in timeliness. New formalisms are proposed to tackle such complicated systems [4,5]. There is a trend to maintain the separation of functional concerns and real-time concerns [4], such that the verification of functionality is not related to the verification of timeliness. Timed CFSM is powerful enough to be used in the verification of timing requirements for multimedia systems. In addition, simulations are adopted to facilitate the verification for the specification of multimedia systems [5]. Timed CFSM can be developed as a simulator, like the verifier proposed in Ref. [11]. As a result, we designed the Timed CFSM to be a powerful formalism for both verification and simulation of real-time and multimedia systems.

This paper is organized as follows. In Section 2, we will introduce Timed CFSM and give example specifications. In Section 3, we will discuss how the reachability graph of the Timed CFSM could be constructed, and give a space-reduction algorithm. Finally, in Section 4, we provide conclusions and future work.

2. Timed Communicating Finite State Machine (Timed CFSM)

In this section, we will first briefly introduce CFSM, which is employed as a basis of our model, and then describe our model, the Timed CFSM.

2.1. Communicating finite state machine (CFSM)

CFSM is a simple and commonly used representation of communication processes. Each process in CFSMs is a finite-state machine and connects with others via a full-duplex, error-free, FIFO channel. The notations for expressing transmissions and receptions of messages are ' − ' and ' + ', respectively. A ' − ' sign represents the transmission of a message denoted by a capital word, while a ' + ' sign represents its reception. Fig. 1 shows a simple system with two processes specified by CFSM. Initially, the process CLIENT is in state READY, and the process SERVER in state IDLE. The process CLIENT sends a request REQ to the process SERVER, and makes a transition to the state WAIT. After receiving REQ, the process SERVER transits into the state SERVICE, in which service is processing. After the service is done, SERVER sends a message DONE and returns to the state IDLE. CLIENT also returns back to the initial state READY after receiving the message DONE. A similar case exists if SERVER initializes the communication by sending a message ALARM. Other transitions are in the same fashion. The details together with the definition of CFSM can be found in Ref. [10].

2.2. Reachability analysis of CFSM

It has been proved that the reachability analysis is valid for us to understand the execution of the system specified in CFSMs [10]. The reachability analysis involves the exploration of all interactions between processes and the derivation of all reachable global states. A global state is a combination of the states of all processes and the messages queued in all channels. The reachability analysis is a procedure for constructing the reachability graph, in which each node represents a reachable global state and each path corresponds to transitions of processes.

Fig. 2 illustrates the reachability graph for the CFSM shown in Fig. 1. The root of the reachability graph is the

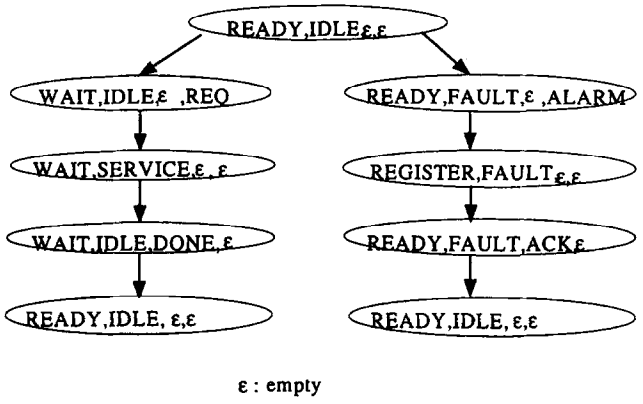

Fig. 2. The reachability graph.

global state, which includes initial states of all processes and the communication queues between processes, which are empty initially. The successors of the root are those global states in which the state of a local process transits and the states of other processes remain the same. All nodes could be expanded by recursion until there are no unexpanded nodes left.

2.3. Time intervals

It is suggested that real time is represented by a real number in Ref. [2]. However, because computer time is discrete, it is reasonable to use a discrete model for representing computer time, in which computer time is represented by a natural number (ct). ct starts from 0 and increases by 1 each time when a new tick is generated. ct never decreases, except when the computer is rebooted and thus ct is reset to 0. In a distributed system, because each host has its own local time ct, we need a time server to synchronize the local ct values. For simplicity, our time model assumes that the time server maintains a global time, gt, and every host in the distributed system is able to access gt and updates its local ct accordingly, without any delay.

Within the time model, a time interval, or bound alternatively, is represented as a 2-tuple $[a,b]$, where both a and b are natural numbers and $a \leq b$. The relative time interval $[a,b]$ is a time interval measured with respect to some reference time instant. An event $E[a,b]$ with time constraint means that the event should be invoked no earlier than time instant a and no later than time instant b.

2.4. Timed CFSM

Based on the CFSM, we propose the Timed CFSM. We bound each transition with a time interval that could be interpreted as a requirement (i.e. the ready time, the deadline, etc.). A relative time interval $[a,b]$ bounded with a transition is measured relative to the time r, at which the process i enters the source state S. On the other hand, $[r+a, r+b]$ is the absolute time interval that the transition must obey. With the time constraint, a communication can succeed only under certain conditions. The following scenario explains these conditions.

Scenario: The process P_i is ready to send a message M in the time interval $[a,b]$ relative to the time T_i when it enters the state S_i. The process P_j is ready to receive the message M in the interval $[c,d]$ relative to T_j in state S_j. The communication possibly occurs in the intersection of the intervals $[T_i + a, T_i + b]$ and $[T_j + c, T_j + d]$ the communication delay is not ignored. Fig. 3 shows the scenario.

In addition, a message type NULL is introduced in Timed CFSM. If a state transition is time-dependent without an occurrence of communication, we associate a time interval $[a,b]$ with the transition only. Based on the aforementioned description, the formal definition of Timed CFSM is**Definition:** A *Timed CFSM* is a quadruple $(\langle S_i\rangle_{i=1,n}; \langle o_{ij}\rangle_{i=1,n}; \langle M_{ij}\rangle_{i,j=1,n}; \text{succ})$ where n is a positive integer, representing the number of processes;

$\langle S_i\rangle_{i=1,n}$ are n disjoint finite sets of states (set S_i includes the states of process i);
$\langle o_i\rangle_{i=1,n}$ is the set of initial states with o_i to be the initial state of process i;
$\langle M_{ij}\rangle_{i,j=1,n}$ are n^2 disjoint finite sets of messages with M_{ii} to be {NULL} for all i, (M_{ij} represents the set of messages sent from process i to process j);
succ is a partial function mapping for each i and j,

$$S_i \times M_{ij} \times [N, \leq] \rightarrow S_i \quad \text{and} \quad S_i \times M_{ji} \times [N, \leq] \rightarrow S_i$$

i.e. $[N, \leq]$ is a partial ordered set where N is the set of natural numbers. Let a,b be any two elements of N such that $a \leq b$. The set $\{x | a \leq x \leq b\}$ is called an interval of N denoted by $[a,b]$. Mapping to our time model, $[N, \leq]$ is exactly the set of time intervals.

Fig. 4 shows a specification using Timed CFSM, which is an extended case of the example in Fig. 1. Timing constraints are augmented to the system. Initially, gt is set to 0. Meanwhile CLIENT is in state READY, and SERVER in IDLE. It is valid for CLIENT to send a request (message REQ) to SERVER in the interval [0, –] (where ' – ' indicates a 'don't care' constraint which is interpreted as

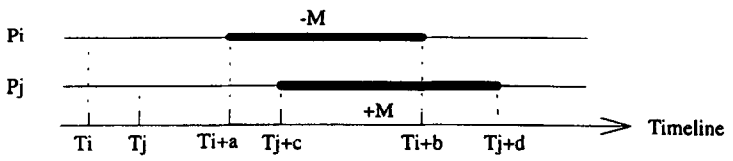

Fig. 3. Communication between P_i and P_j.

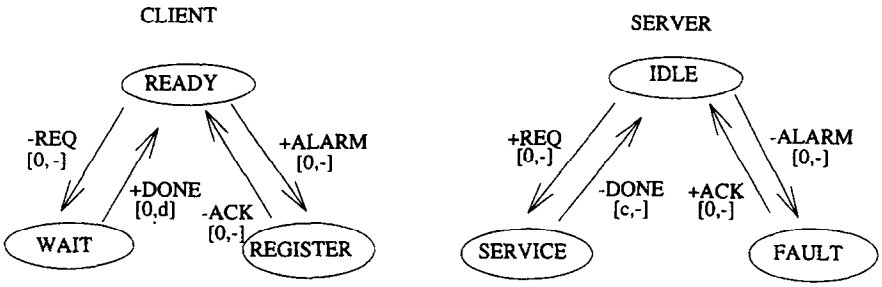

Fig. 4. Timed CFSM.

infinity here). SERVER is free to receive REQ since its bound is also [0, −]. On receiving REQ, SERVER enters state SERVICE. After at least c time units have passed, it completes the service, sends the message DONE to CLIENT, and goes back to state IDLE. Between sending REQ and receiving DONE, CLIENT stays in state WAIT with a bound [0,d]. When receiving DONE, CLIENT checks its time stamp to see whether it is valid or not. An out-of-date message will be of no use at all.

In Fig. 4, it is intuitive to see that CLIENT would violate the timing bound + DONE[0,d] if $d < c$. However, it is not easy to verify timing properties when the number of processes and states increases. Therefore, we need to construct the reachability graphs from Timed CFSM. Before formally presenting the algorithm for reachability graph construction, we introduce the reachability analysis of Timed CFSM, and compare the similarity and the difference between the reachability analysis of CFSM and that of Timed CFSM.

2.5. Reachability analysis of Timed CFSM

As in Section 2.2, a reachability graph for CFSM is composed of nodes (global states) linked by paths (transitions). The reachability graph for Timed CFSM is similar to that for CFSM. We define the global states and the transitions of Timed CFSM as follows.

Definition: a *global state* is a pair ⟨S,C⟩ where S is an n-tuple $(s_1,s_2,\ldots,s_n)$ with s_i to be the current state of process i, and C is an n^2-tuple $(C_{11},\ldots,C_{1n},C_{21},\ldots,C_{nn})$ with C_{ij} to be the sequence of messages from M_{ij}.

Definition: the *initial global state* is the global state $\langle S_0, C_0\rangle$ with S_0 to be $(o_1,o_2,\ldots,o_n)$, and C_0 to be empty.

Definition: a *global transition* is a binary relation → on global states: ⟨S,C⟩⟨S′,C′⟩ iff there exists i, j, and x_{ij} satisfying one of the following conditions:

(i) All the elements of ⟨S,C⟩ and ⟨S′,C′⟩ are the same except

$s_i' = \text{succ}(s_i, -x_{ij})$ and $C_{ij}' = C_{ij}x_{ij}$

$s_j' = \text{succ}(s_j, +x_{ij})$ and $C_{ij} = x_{ij}C_{ij}'$

where + and − represent receiving and sending of x_{ij}.

(ii) All the elements of ⟨S,C⟩ and ⟨S′,C′⟩ are the same except

$s_i' = \text{succ}(s_i,\text{NULL})$

Definition: a global state ⟨S,C⟩ is *reachable* iff $\langle S_0,C_0\rangle \rightarrow {}^*\langle S,C\rangle$ where → * is the reflexive and transitive closure of the global transition → .

In addition, the reachability graph of Timed CFSM differs from that of CFSM in two aspects.

(1) Not all global states in the reachability graph of CFSM should be included in that of Timed CFSM. A global state which is unreachable owing to timing constraints should be deleted from the reachability graph of Timed CFSM.

(2) A global state in the reachability graph of Timed CFSM may be entered more than once and produce different successors owing to different timing relationship.

We will explain how to keep the timing relationship so that we could find both unreachable global states and repeated global states when constructing the reachability graph in the next section. We end this section with an example of Timed CFSM, which will be used to construct the reachability graph in the next section.

2.6. Examples

Fig. 5 shows the railroad crossing control system. The system consists of two machines: MONITOR and

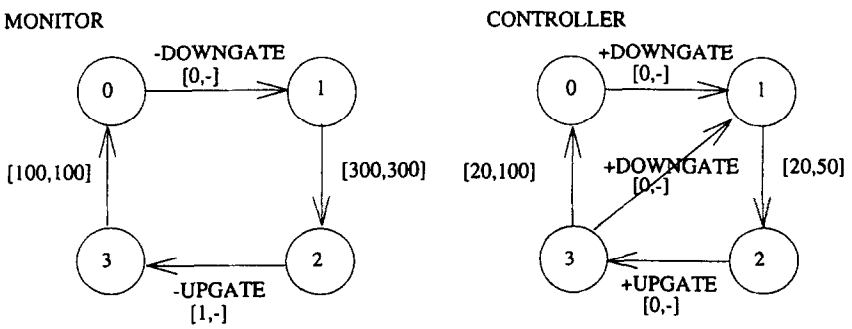

Fig. 5. Timed CFSM for the crossing-road system.

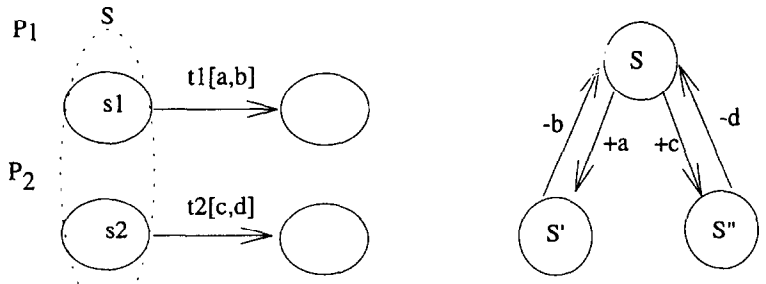

Fig. 6. The scenario behind the pruning process.

CONTROLLER. MONITOR watches the railroad for the train approaching. When the train is less than half a mile from the crossing, MONITOR sends the signal DOWN-GATE to CONTROLLER. While in state 0, CONTROLLER goes to state 1 on receipt of DOWN-GATE, and before the deadline it makes a transition to state 2. On the other hand, the train continues processing and enters state 2 after a delay of 300 time units. The situation is shown in Fig. 5 by the transition with the time interval [300,300] which is a transition of a message type NULL. Later, when the train has passed, MONITOR sends the message UP-GATE to CONTROLLER and transits to state 3. Upon receiving UP-GATE, CONTROLLER transits to state 3 immediately. While in state 3, CONTROLLER either transits to the initial state 0, or transits back to state 1 if a message DOWN-GATE is received.

3. The construction of reachability graphs

We will present the algorithm for constructing the reachability graph in a backward way. In Section 3.1, we introduce two processes used to modify the reachability graph derived from the non-Timed CFSM, which is the CFSM abstracting timing constraints away from a Timed CFSM. The pruning process deletes those nodes violating timing constraints in the reachability graph of the non-Timed CFSM. On the other hand, the growing process expands the offspring for nodes under certain conditions. The resulting graph, after executing these two processes, is the reachability graph of a Timed CFSM.

3.1. The weighted reachability graph

Before looking into the pruning and growing processes, we introduce the weighted reachability graph which is derived from the reachability graph of the non-Timed CFSM by adding weighted edges. *The weight of an edge* between two global states is assigned either a positive value or a negative value. A positive weight of the edge from global state S to global state S′ represents the minimum delay of the system in state S before transiting to state S′. On the other hand, a negative weight of the edge from S′ to S represents the deadline for the system to transit from S to S′. In addition, if the value of the weight is 0, it means that S is synchronized with S′. With the definition of weights, we present the algorithm for construction of a weighted reachability graph below:

(1) Construct the reachability graph of the non-Timed CFSM.

(2) Traverse the reachability graph width-first. During the traversal, add weighted edges by the following rules: (i.e. we will refer a node in the reachability graph as a global state to avoid confusion).

Assume the presently traversed node is S, and S′ is a successor of S resulting from a transition t[*a*,*b*] from local state s to local state s′.

- If t is a sending or a NULL operation, add an edge with weight $+a$ from S to S′ indicating the delay between S and S′, and an edge with weight $-b$ from S′ to S indicating the deadline of the transition.
- If t is a receiving operation, add an edge with $+0$ from S to S′ and an edge -0 from S′ to S. The weights with value 0 indicate the synchronization for S and S′. Push S in a temporary stack. Backtrack along the path; find the global state S″ which includes s as one of its local states, whereas the predecessor of S″ does not. S″ is called *the trigger node* of S′ corresponding to s, which triggers the transition to s. Add an edge with weight $+a$ from S″ to S′ and an edge with $-b$ from S′ to S″. Pop up S for continuing traversals.

(3) Continue the traversal until all nodes are traversed.

In the following sections, we will apply the pruning and growing processes to the weighted reachability graph. Before we continue, we introduce some terms about the weighted reachability graph: the weight of a path P from S to S′ is the sum of weights of consecutive edges composing P, and the weight of the shortest path from S to S′, denoted as w(S,S′), is defined as the minimum path weight from S to S′. If $w(S,S') > 0$, S happens before S′. If $w(S,S') = 0$, S is synchronized with S′. On the other hand, if $w(S,S') < 0$ and $w(S',S) < 0$, S and S′ are concurrent.

3.1.1. The pruning process

The pruning process is used to discard global state violating timing constraints. Fig. 6 shows the scenario involving two processes, P1 and P2. The left part of Fig. 6 shows

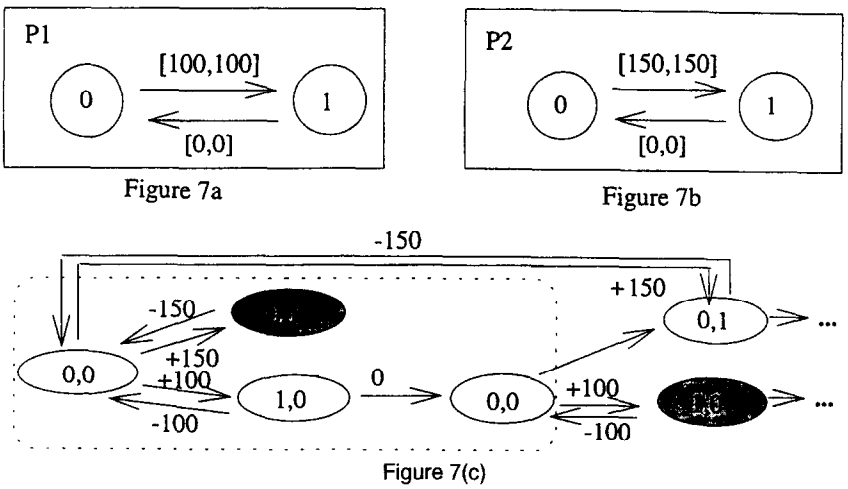

Figure 7a

Figure 7b

Figure 7(c)

Fig. 7. Scenario involving processes P1 and P2.

the Timed CFSMs for P1 and P2, respectively, and the right shows the weighted reachability graph. Let S represent the global state (s_1,s_2), $t_1[a,b]$ is a transition from s1, and $t_2[c,d]$ is from s_2 When constructing the next global states, if both P1 and P2 are executable (i.e. a global state is executable if it contains a transition of type sending or type NULL, or type receiving with desired message available) and the next global states are S′ and S″, respectively. There will be three possible relations between time intervals $[a,b]$ and $[c,d]$ as follows:

1. If $b < c$, $[a,b]$ is before $[c,d]$. Under this condition, $w(S',S'')$ which is equal to $-b+c$ will be positive. Thus, S′ happens before S″, so S″ is unreachable from S due to S′.
2. If $d < a$, $[c,d]$ is before $[a,b]$. Under this condition, $w(S'',S')$ which is equal to $-d+a$ will be positive. Thus, S″ happens before S′, so S′ is unreachable from S due to S″.
3. If $a \leq c \leq b$ or $c \leq b \leq d$, $[a,b]$ overlaps with $[c,d]$. Under this condition, both w(S′,S″) and w(S″,S′) ≤ 0, therefore S′ and S″ are concurrent. and are reachable from S.

In addition, there is another type of unreachable state, namely the unsatisfied receiving, which happens under the condition when a message with an invalid timestamp is received. If S′ is a successor of S and the transition t from S to S′ is a receiving, we have to check whether the receiving message arrives in the valid period. To sum up, the algorithm for the pruning process is shown below:

1. Traverse the reachability graph width-first.
2. During the traversal, assume S is the currently traversed node, and SUCC is the set of successors of S. For each S_i in SUCC, if there exists S_j in SUCC, such that $w(S_j,S_i)$ is positive, discard S_i and its offspring.
3. For those S_i with a transition of the type receiving, find the trigger node St_i for S_i corresponding to the receiving, and calculate the weight of the cycle S_i,S,...,St_i,S_i. If the weight is positive, that is, the message arrives too early, the global state S together with its offspring should be discarded. On the other hand, if the aggregated weight for the cycle S_i,St_i,...,S,S_i is positive, that is, the message arrives too late (the deadline of the reception is violated), the global state S together with its offspring should be discarded.
4. Repeat step (2) through (3) until all nodes are traversed.

3.1.2. The growing process

The growing process is used to expand the pruned graph, and thus derive the reachability graph. The pruned graph is not the reachability graph for the Timed CFSM yet, because two global states containing the same local states and the same channel messages could generate different successors under different timing sequences. The growing process identifies the condition and expands the weighted reachability graph when necessary. Fig. 7 shows the scenario involving two processes, P1 and P2, which run with periods 100 and 150, respectively. Fig. 7(a) and (b) show the Timed CFSMs of P1 and P2. Fig. 7(c) shows the partial reachability graph with unreachable states marked gray. The dash-line block in Fig. 7(c) contains the weighted reachability graph derived from the non-timed CFSM, and the other part of Fig. 7(c) is what is newly expanded. The execution of the system is described as follows. The initial global state is (0,0). After 100 time units, P1 goes to state 1 and back to state 0 instantaneously. Again the global state is (0,0). However this time the state (0,0) is different from the initial global state, though both have the same local states. We observe that P2 is able to make a transition to state 1 after 50 time units, which is not enabled in the initial global state. In fact, P1 and P2 will run periodically every 300 time units. Therefore, after 300 time units, the system is in a state equivalent to the initial global state. We state the algorithm for the growing process as follows:

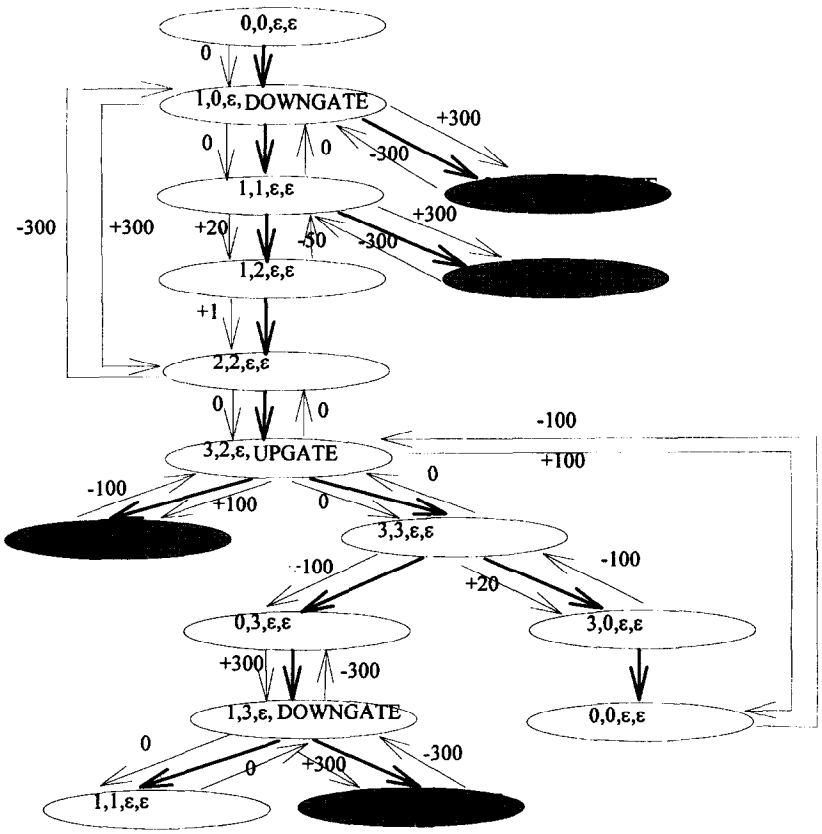

Fig. 8. The weighted reachability graph.

1. Traverse the graph width-first. Start from the root; mark it as expanded.
2. If the currently traversed global state S contains the same local states and channel messages with some expanded state S′, we use the following rules to identify their equivalence. Let S be $(s_1,s_2,\ldots\ldots s_n)$, and the set of successors SUCC be $\{S_1,S_2,\ldots,S_k\}$, resulting from transitions $t_1,t_2,\ldots,t_k$, respectively. Let SUCC′ be the set of successors of S′, which is equal to SUCC. For each state s_i in S, find the trigger node for S corresponding to s_i, say St_i. At the same time, for s_i in S′, find the trigger node, say St_i. For each successor S_j from S, and the corresponding successor S_j' from S′.

- If the conditions $w(St_i,S_j) = w(St_i',S_j')$ and $w(S_j,St_i) = w(S_j',St_i')$ hold, then S is equivalent to S′; the subgraph generated from S′ is isomorphic to the subgraph generated from S. Under such condition, discard S′ and go to step (4). Or if the following two conditions hold, S is also equivalent to S′.
- If l is the largest lower bound for all t_i values then the conditions $w(St_i,S_j) \geq l$ and $w(St_i', S_j') \geq l$ hold.
- If u is the smallest upper bound for all t_i values, then the conditions $w(St_i,S_j) < u$ and $w(St_i', S_j') < u$ hold.When these two conditions hold, all transitions from S and S′ to their successors can be enabled and all deadlines can be met. In this case, all successors can be reached, and therefore S is equivalent to S′.

3. Otherwise, S is different from S′. We should expand the subgraph from S by copying the subgraph from S′.
4. Repeat (2) through (3) until all nodes are traversed.

After applying the pruning and growing processes, the resulting graph omitting weighted edges is the reachability graph of the Timed CFSM. Fig. 8 shows the weighted reachability graph, in which the gray nodes represent unreachable global states.

3.2. *The proposed algorithm for reachability graph construction*

With the pruning and growing processes, we have developed an algorithm for constructing the reachability graph from a Timed CFSM. The algorithm is explained as follows:

1. Construct S_0 the initial global state. Designate S_0 as an unexpanded node.
2. Choose an unexpanded node S. Mark it as expanded. Let SUCC be the set of successors $\{S_1,S_2,\ldots,S_k\}$ Add the weighted edges. Use the pruning process to discard unreachable successors in SUCC.
3. For each S_i remaining in SUCC, if it is equivalent to an expanded global state S′, add an edge from S to S′; else add an edge from S to S_i, mark S_i unexpanded.
4. Repeat step (2) through (3) until there are no unexpanded nodes left.

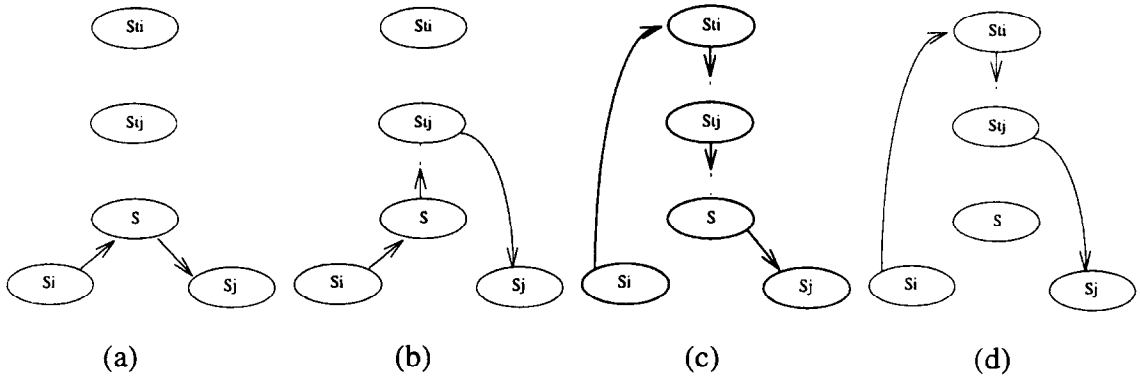

Fig. 9. The four candidates of the shortest path from S_i to S_j.

The reachability graph constructed from the algorithm is proved to be finite. The proof is as follows:**Proof:** In the growing process, we have identified the conditions for two global states to be equivalent. All the equivalent global states could be classified into an equivalent class. If we prove the number of equivalent classes is finite, we can assert that the construction of the reachability graph can be terminated. We state the proof in three steps:

1. Partition the nodes of the reachability graph by their local states and channel messages. The nodes of a partition have the same local states and channel messages.
2. For each node S, refine the partition as follows:

   (A) If the deadlines for all transitions of S's local states are infinite, then we can divide the nodes into classes 0 through r, where r is the largest ready time among all transitions of S's local states. We classify all S's in the partition according to $w(S',S)$, where S' is the trigger node for S. A node S belongs to class i if $w(S',S)$ is equal to i and less than r. Otherwise, S belongs to class r. Thus, there are $r+1$ classes in total.

   (B) if the smallest deadline for S's local state transitions is finite, say d, then for each successor S', classify the nodes into class 0 through $d+1$, according to $w(S,S')$. The partition is further refined according to $w(S',S)$. If the deadline for the transition to S' is infinite, then $w(S',S)$ is negative infinite. Alternatively, if the deadline is finite, say d', then $w(S',S)$ is bounded to the sum of the two deadlines $d+d'$. Therefore, there are a finite number of equivalent classes.
3. Each node of the reachability graph must belong to a certain equivalent class, as described in step (2). Thus, the construction of the reachability graph can be terminated.

In the next section, we will discuss the space requirement of the algorithm.

3.3. Time and space requirements of the algorithm

Two basic steps of our algorithm are the assignment of weighted edges and the calculation of weights of paths. We will discuss the time requirement of two basic steps. Assume the currently traversed node is S, the executable transitions from S are $t_1,t_2,\ldots,t_k$ with corresponding source local states $s_1,s_2\ldots,s_k$ and resulting successors $S_1,S_2\ldots,S_k$. To assign weighted edges we have to backtrack the graph to find the trigger nodes St_i for S corresponding to s_i. As a result, the time complexity of the backtracking is linearly proportional to the length of the path from St_i to S_i. On the other hand, to calculate $w(S_i,S_j)$, we have to find the shortest path from S_i to S_j. There are four candidates for the shortest path shown in Fig. 9: (a) S_i,S,S_j, (b) $S_i,S,\ldots,St_j,S_j$, (c) $S_i,St_i,\ldots,S,S_j$ and (d) $S_i,St_i,\ldots,St_j,S_j$. The time complexity of the calculation of $w(S_i,S_j)$ is linearly proportional to the length of the path from S_i to S_j. In the worst case, if the trigger node for S corresponding to S_i is the root R, we have to backtrack all nodes along the path from S to R. Again, in the worst case, the path length can be equal to the state space. However, in general, the trigger nodes for S are not far from S.

In the aspect of space complexity, the analysis of reachability suffers from the state space explosion problem. Assume that there are, n processes in the Timed CFSM, each global state will have n states and n^2 communication channels, and the average number of states of each process is l, then the space requirement is $(n\cdot s + n^2\cdot m)\cdot l^n$ where s and m are space requirements for keeping a state and a channel, respectively. In addition, the growing process will speed up the explosion of space. As a result, we have to reduce the state space during the construction of the reachability graph. A possible solution is to construct only partial graphs when we validate the Timed CFSM. In the next section, we will discuss our solution based on the path approach.

3.4. Space complexity reduction

Before we present our solution for space reduction, we will describe the tree representation of a Timed CFSM in Section 3.4.1 and then explain the path approach in Section 3.4.2.

3.4.1. Tree representation of timed CFSMs

The tree representation of a Timed CFSM is a collection

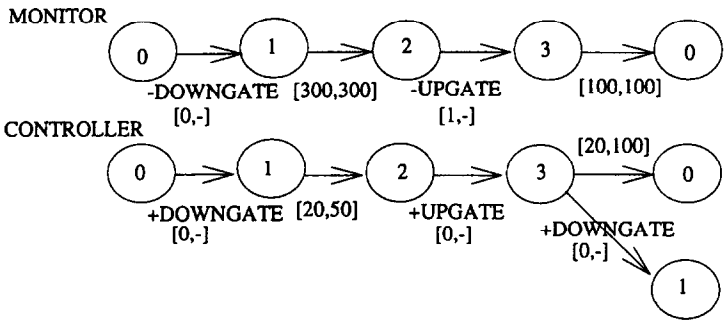

Fig. 10. Tree representation of Timed CFSM.

of trees each of which represents a process in the Timed CFSM. The root of a tree is the initial state of a process P_i and we expand the tree recursively with the following algorithm.

1. Start from the initial state o_i. Designate o_i as the root of the tree.
2. Choose a branch node. Add all transitions from the corresponding state as edges, and all entering states as nodes. If the newly expanded node has a terminal state or is the same as some branch node on the path from root, then it is a leaf; else it is a branch node.
3. Repeat (2) until all branch nodes expanded.

Fig. 10 shows the tree representation of the railroad crossing control system in Section 2.5. A *path* for a process P_i is defined as a path starting from the root and ending with a leaf in the tree representation. For example, the only path in the process MONITOR is 0–1–2–3–0 and two paths in the process CONTROLLER are 0–1–2–3–0 and 0–1–2–3–1, respectively.

3.4.2. The path approach

The path approach is to validate the Timed CFSM, denoted as M, by constructing only partial graphs of the reachability graph. A concurrent path set is a set whose elements are paths $p_1,p_2,\ldots,p_n$ in processes $P_1,P_2,\ldots,P_n$ respectively. A concurrent path set could be viewed as a new Timed CFSM, M′, in which each path represents a process. Therefore we could apply the algorithm in Section 3.2 to construct the reachability graph of M′ which is a partial graph of the reachability graph of M. During the construction of the reachability graph, if M′ enters a state that has no successors, the concurrent path set is illegal. A state does not have successors under the following conditions: (1) a process in the concurrent path set sends a message while the receiving processes is not in the concurrent path set. Under this condition, the concurrent path set is invalid; (2) a deadlock occurs within the concurrent path set. With this idea in mind, we revise our algorithm as follows. (Assume the tree representation has been constructed.)

1. Choose a concurrent path set, start from the initial global state S_0 Mark it as unexpanded.
2. Choose an unexpanded node S. If an unexpanded node is not found, store the reachability graph in the secondary storage and go to step (5). Let SUCC be the set of successors $\{S_1,S_2,\ldots,S_k\}$. If SUCC is null, go to step (4). Add the weighted edges and discard unreachable successors in SUCC.
3. For each S_i remaining in SUCC, if it is equivalent to an expanded global state S′, add an edge from S to S′. Otherwise, add an edge from S to S_i, mark S_i unexpanded, and mark S expanded. Go to step (2).
4. If there is any message left in channels, the concurrent set is invalid. On the other hand, if there is no message left in channels, we conclude that a deadlock has occurred.

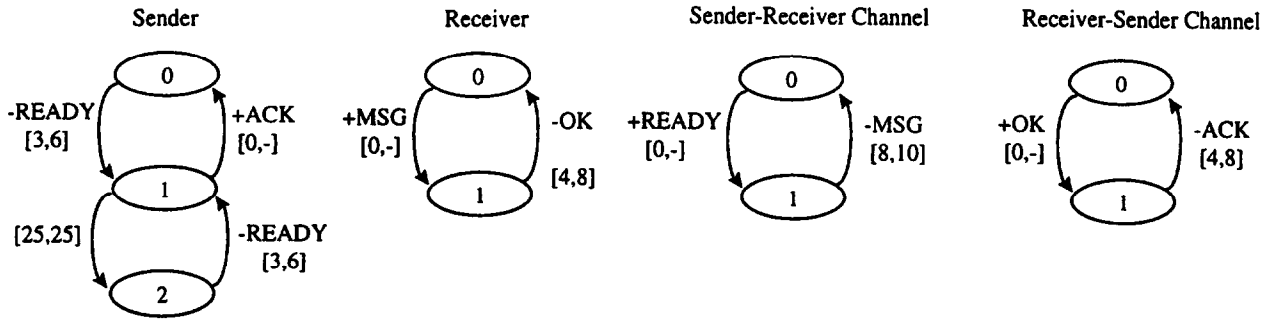

Fig. 11. Timed CFSM for stop-and-wait protocol.

5. Repeat (1) through (3) until all concurrent path sets are verified.

The space requirement of the revised algorithm would be less than that of the original algorithm. Assume the average length of each path is l', which is shorter than l. Each time that a concurrent path set is validated, the number of possible global states is $(l')^n$. Furthermore, the capacity of communication channels needs to be one unit only. The memory requirement of the algorithm is $(n \cdot s + n^2) \cdot (l')^n$, where n is the number of the processes, and s is the space requirement for keeping a state. Although the space requirement of the revised algorithm is also exponential, the base component is smaller than that in the previous algorithm. On the other hand, the time complexity would be Πn_i times compared to the original algorithm. If the removal of illegal concurrent path sets is performed in advance, the time complexity can be further reduced.

3.5. Applications of timed CFSM

Many communication protocols, in which timing constraints are required to be bundled with original functionality, can be modeled by Timed CFSM. For example, the stop-and-wait protocol presented in Ref. [9] can be modeled as the Timed CFSM in Fig. 11. There are four entities in the Timed CFSM: a sender, a receiver, a sender–receiver channel, and a receiver–sender channel. We can identify both unreachable and equivalent global states during the construction of the reachability graph. Other flow control protocols, such as X-on/X-off, Ping-Pong, and Window protocols, can also apply reachability analysis to verify their timing property in a similar way.

4. Conclusions

In this paper we propose the Timed CFSM, which can serve as a model for specifying and verifying time-critical systems. Transitions in Timed CFSM are bounded by a time interval, defined by [min,max], where min is the ready time and max is the deadline. Given a specification formally defined by a Timed CFSM, we are able to verify its timing properties by constructing the reachability graph for the Timed CFSM. The construction of the reachability graph from Timed CFSM is achieved by the pruning and growing processes. To cope with the space explosion problem, we propose a space complexity reduction algorithm to meet the space constraint of the verification environment.

Acknowledgements

This work was supported in part by the National Science Council of Taiwan. under contract NSC 85-2213-E-009-032.

References

[1] A.C. Shaw, Reasoning about time in higher-level language software, IEEE Trans. Software Eng. 15 (1989) 875–889.
[2] A. Gabrielian, M.K. Franklin, Multilevel specification of real-time systems, Comm. of ACM, May 1991, pp. 51–60.
[3] A.C. Shaw, Communicating real-time state machines, IEEE Trans. Software Eng. 18 (Sept.) (1992) 805–816.
[4] A.F. Ates, M. Bilgic, S. Saito, B. Sarikaya, Using timed CSP for specification, verification and simulation of multimedia synchronization, IEEE J. Selected Areas in Communications 14 (Jan.) (1996) 126–137.
[5] A. Lakas, G.S. Blair, A. Chetwynd, Specification and verification of real-time properties using LOTOS and SQTL, in: Proceedings of IWSSD-8, IEEE, 1996.
[6] B. Berthomieu, M. Diaz, Modeling and verification of time dependent systems using Time Petri Nets, IEEE Trans. Software Eng. 17 (Mar.) (1991) 259–273y.
[7] C. Hoare, An axiomatic basis for computer programming, Com ACM 10 (12 Oct.) (1969) 576–580.
[8] C. Hoare, Communicating sequential processes, Comm. ACM 8 (2 Aug.) (1978) 666–677.
[9] C.M. Huang, S.W. Lee, J.M. Hsu, Probabilistic timed protocol verification for the extended state transition model, Proceedings of International Conference on Parallel and Distributed Systems (1994), Hsinchu Taiwan, pp. 432–437.
[10] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. ACM 30 (April) (1983) 323–342.
[11] D. Stuart, Implementing a verifier for real-time systems, in: Proceedings of the 11th Real-Time Systems Symposium, Lake Buena Vista, Florida, (1990), pp. 62–71.
[12] F. Jahanian, A. Mok, Safety analysis of timing properties in real-time system, IEEE Trans. Software Eng. SE-12 (Sept.) (1986) 890–904.
[13] F. Jahanian, D. Stuart, Method for verifying properties of modechart specifications, in: Proceedings of the Ninth Real-Time Systems Symposium, Huntsville, Alabama, (1988), pp. 12–21.
[14] C. Ghezzi, D. Mandrioli, S. Morasca, M. Pezz'e, Unified high-level Petri net formalism for time-critical systems, IEEE Trans. Software Eng. 17 (Feb.) (1991) 160–172.
[15] J.S. Ostroff, W.M. Wonham, Modeling, specifying, and verifying real-time embedded computer systems, in: Proceedings of the Eighth Real-Time Systems Symposium, San Jose, California (1987), pp. 124–132.
[16] M. Felder, D. Mandrioli, A. Morzenti, Proving properties of real-time systems through logical specifications and Petri net models, IEEE Trans. Software Eng. 20 (Feb.) (1994) 127–141.
[17] N.G. Leveson, J.L. Stolzy, Safety analysis using Petri nets, IEEE Trans. Software Eng. SE-13 (Mar.) (1987) 386–397.
[18] T. Litter, A. Ghafoor, Synchronization and storage models for multimedia objects, IEEE J. Selected Areas in Communications 8 (Apr.) (1990) 413–427.