Single-Link and Time Communicating Finite State Machines

Wuxu Peng*
Department of Computer Science
Southwest Texas State University
San Marcos, TX 78666

Abstract

We propose two variants of the classical Communicating Finite State Machines (CFSMs) model – Single-Link Communicating Finite State Machines (SLCFSMs) and Time Communicating Finite State Machines (TCFSMs). For SLCFSMs the notion of well-formedness, *which provides a necessary condition for SLCFSMs to be free of some logical errors, is proposed. For TCFSMs, it is argued that they are more suitable for modeling* delay-sensitive *distributed algorithms/communication protocols. Two practical communication protocols – a token ring and a sliding window protocol – are modeled using TCFSMs.*

1 Introduction

A network of *communicating finite state machines* (CFSM) consists of a set of finite state machines which communicate asynchronously with each other over (potentially) unbounded FIFO channels by sending and receiving typed messages. As a concurrency model, CFSMs has been widely used to specify and validate communications protocols [1, 6, 12, 10, 11].

This paper consists of two parts. In the first part, a new concurrency model – *single-link communicating finite state machines* (SLCFSMs), which is a variation of the CFSM model, is proposed. In a network of CFSMs, incoming messages from different machines to any individual machine are coming from different and separate FIFO channels. For SLCFSMs, each machine has only one incoming FIFO link and incoming messages from different machines all come from this single link.

This new model is primarily motivated by our perception that the software component of many communication protocols on individual computers (or systems) can be abstracted by a guarded command enclosed in an infinite loop. Messages from different machines are all passed, in FIFO order, to this component. The single guarded command responds differently to different events, such as receipt of incoming messages from different machines. This view of communications can be very nicely abstracted by the proposed SLCFSM model.

An important characteristics of the CFSMs model is that the time aspects of communications are ignored. Messages sent out by a sending process is assumed to reach the receiving process instantly. This assumption greatly simplifies and is adequate for the specification and verification of some important communication protocols.

Lack of proper mechanism for expressing time in CFSMs, on the other hand, has also severely limited its applicability to many other practical communication protocols. It is now well known that timeout is of fundamental importance to most communication protocols. This fact indicates that most communication protocols are delay sensitive, not delay insensitive.

In the classical CFSMs model timeout is usually modeled by a send transition exiting and entering to the same local state. This modeling is very gross. There is no control as when the send event can occur. As a result, the self-loop style send transition causes the communications to be *unbounded*. In verification there is no easy way to differentiate the unboundedness caused by the above timeout transitions and unboundedness that indicates a design error in the underlying communication protocols.

Modeling/self-stabilizing delay sensitive communication protocols/distributed algorithms such as token rings in CFSMs is very difficult, if not impossible at all. In the second part of this paper another variant of the CFSMs model – *time communicating finite state machines* (TCFSMs) – is proposed. TCFSMs subsumes the classical CFSMs model as a special case. TCFSMs associate two time quantities, which represent time constraints for the execution of the transition, with each transition. Two example applications are used to demonstrate the usefulness of this new model

*Supported in part by faculty research enhancement grant 3-2099 from Southwest Texas State Univ.

0-8186-6685-4/94 $4.00 © 1994 IEEE

for specifying and verifying delay sensitive communication protocols.

We start by introducing SLCFSMs in Section 2. In Section 3, TCFSMs is proposed. Section 4 discusses some interesting properties of the TCFSMs model. Two example applications of TCFSMs are given in Section 5. Section 6 concludes the paper with some remarks.

2 Single-link communicating finite state machines

We define the SLCFSM model in this section. First we define CFSMs. Let $I = \{1, 2, \cdots, n\}$, where $n \geq 2$ is some constant (denoting the number of processes in a network).

2.1 CFSMs

Definition 2.1 (*Communicating Finite State Machines*) *A communicating finite state machine P_i is a four-tuple $(S_i, \Sigma_i^{\pm}, \delta_i, p_{0i})$, where*

- *S_i is the set of local states of machine P_i.*
- *$p_{0i} \in S_i$ is the start state of machine P_i.*
- $$\Sigma_i^{\pm} = \sum_{1 \leq j \leq n} \Sigma_{i,j} \bigcup \sum_{1 \leq j \leq n} \Sigma_{j,i}$$

  where $\Sigma_{i,j}, 1 \leq j \leq n$ is the alphabet of messages that P_i can send to P_j, and $\Sigma_{j,i}, 1 \leq j \leq n$ is the alphabet of messages that P_i can receive from P_j.
- *$\delta_i : S_i \times \pm\Sigma_i \times I \rightarrow 2^{S_i}$ is the transition function. $\delta_i(p, -m, j)$ is the set of states that process P_i could move to from state p after sending a message m to process P_j. $\delta_i(p, +m, j)$ is the set of states that process P_i could move to from state p after receiving a message m sent by process P_j.* ■

A network of CFSMs (NCFSMs) is a tuple $\langle P_1, P_2, \cdots, P_n \rangle$ where each P_i is a CFSM. Detailed definitions of NCFSMs can be found in [1, 6, 10, 12]. We shall take the freedom of using $\vec{c_0}$ to denote an all-empty channel and $[\vec{v_0}, \vec{c_0}]$ the initial global state. A global state $[\vec{v}', \vec{c}']$ in a NCFSMs is *reachable* from another global state $[\vec{v}, \vec{c}]$ if the former is reachable from the latter in zero or more steps. The reachability set, denoted by $RS(N)$, is defined as the least set that contains all the global states reachable from $[\vec{v_0}, \vec{c_0}]$. We shall use δ to denote the the global state transition function [10].

The task of verifying a NCFSMs modeling a communication protocol is to determine if the NCFSMs possesses some undesirable properties that reflects some logic errors in the original protocol. Among the most interested and investigated undesirable properties are *Deadlocks, unspecified receptions*, and *unbounded communications* [1, 6, 10, 11].

The *topology graph*, noted $TG(N)$ for a NCFSMs N, is a directed graph where the nodes are the CFSMs in the network and an edge from P_i to P_j indicates that there is a unidirectional channel from the former to the latter. A global state $[\vec{v}, \vec{c}]$ is said to be *stable* if $\vec{c} = \vec{c_0}$, i.e. all channels are empty. Apparently, a deadlock state is also a stable state, but the reverse is not true.

2.2 SLCFSMs

Syntactically, a SLCFSM is the same as a CFSM. However, the semantics of a network of SLCFSMs is quite different from a NCFSMs. Formally, we define:

Definition 2.2 (*Networks of Single-Link Communicating Finite State Machines*) *A network of single-link communicating finite state machines (NSLCFSMs) is a tuple $\langle P_1, P_2, \cdots, P_n \rangle$ where each P_i is a SLCFSM. Let c_i denote the (potentially) unbounded FIFO buffer that holds messages that process P_i can receive from other machines (It is also assumed that any machine P_i cannot send to and/or receive messages from itself). The semantics of a NSLCFSMs can be captured by the concepts of* global states *and* global state transitions. *A global state is a tuple $[\vec{v}, \vec{c}]$, where $\vec{v} = \langle p_i \rangle_{i \in I}$ and $\vec{c} = \langle c_i \rangle_{i \in I}$ are the n FIFO links, one per machine. The initial global state is $[\langle p_{0i} \rangle_{i \in I}, \langle c_i \rangle_{i \in I}]$, where $c_i = \varepsilon$. A global state $[\langle p'_k \rangle_{k \in I}, \langle c'_k \rangle_{k \in I}]$ is* one-step reachable *from a global state $[\langle p_k \rangle_{k \in I}, \langle c_k \rangle_{k \in I}]$, if $\exists i, j \in I$ such that:*

- *either $p_i \stackrel{(-m,j)}{\longrightarrow} p'_i$ is in P_i, $\forall k \in I (k \neq i \rightarrow p_k = p'_k)$; and $c'_j = c_j.m$, $\forall k \in I (k \neq j \rightarrow c'_k = c_k)$; or*
- *$p_i \stackrel{(+m,j)}{\longrightarrow} p'_i$, $\forall k \in I (k \neq i \rightarrow p_k = p'_k)$; and $m.c'_j = c_j$, $\forall k \in I (k \neq i \rightarrow c'_k = c_k)$.* ■

Fig. 1 illustrates the semantic difference between NCFSMs and NSLCFSMs. In a NCFSMs, incoming messages from a CFSM P_j to a CFSM P_i will go through the FIFO channel $c_{j,i}$, and messages from different machines go through different FIFO channels. In a NSLCFSMs, on the other hand, there is only a single FIFO link leading to any SLCFSM P_i. Messages from different machines will be mixed up in the

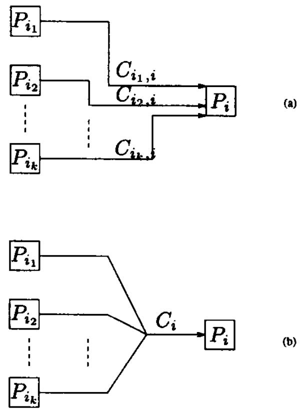

Figure 1: Semantic difference between CFSMs and SLCFSMs: (a) CFSMs; (b) SLCFSMs

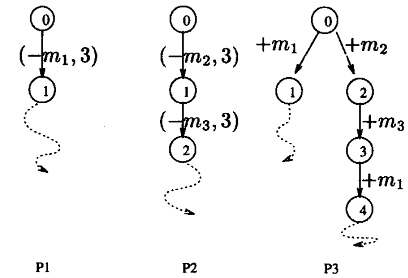

Figure 2: Need of well-formedness

order of arrival in that single link. Notice that messages from different machines sent at the same time to any individual machine P_i will be mixed up nondeterministically in the link for P_i.

From the definition, it can be seen that for any NCFSMs N, if there is at most one incoming channel for any CFSM in N, then N is also a NSLCFSMs. In particular, every *cyclic* NCFSMs is also a NSLCFSMs (a NCFSMs is cyclic if its topology graph is a simple cycle. Cf. [11]). We summarize this fact in the following proposition.

Proposition 2.1 *If there is at most one incoming channel for any CFSM in a NCFSMs N, then N is also a NSLCFSMs.*

Most definitions in NCFSMs, such as deadlocks, unboundedness, reachability, reachability sets, and stable global states, carry over to NSLCFSMs with no need for or with obvious modifications. Therefore we do not repeat them here.

2.3 Well-formed SLCFSMs

In this section we propose the notion of *well-formedness* of SLCFSMs, which provides a necessary condition for SLCFSMs to be free of some potential errors.

In CFSMs send actions from different machines that send messages to a specific machine will be queued in different FIFO channels. In SLCFSMS, however, these messages will be queued in a single FIFO channel. Consider the case shown in Fig. 2. If the send action $0 \xrightarrow{(-m_1,3)} 1$ in machine P_1 is delayed until P_3 reaches state 3, there is no unspecified reception. However, since send actions are non-blocking, the send action $0 \xrightarrow{(-m_1,3)} 1$ can occur at any time. This motivates the notion of *well-formedness*. We first present several supporting definitions.

Let $s_{i,j}$ denote the event of machine P_i sending a message to P_j and $r_{j,i}$ the event of machine P_j receiving a message sent by machine P_i. Let Σ_i^- be the set of send events in machine P_i, i.e. $\Sigma_i^- = \{s_{i,j} | j \in I \;\&\; P_i \to P_j \in TG(N)\}$. Let Σ_i^+ be the set of receive events in machine P_i, i.e. $\Sigma_i^+ = \{r_{i,j} | j \in I \;\&\; P_j \to P_i \in TG(N)\}$. Use the abbreviation $\Sigma_i^\pm = \Sigma_i^- \cup \Sigma_i^+$ to denote the set of all events of machine P_i and $\Sigma^\pm = \cup_{i \in I} \Sigma_i^\pm$ to denote the set of all events of the network.

- For any word $e \in (\Sigma^\pm)^*$, the notation *prefix*(e) denotes the set of prefixes of e, i.e. *prefix*$(e) = \{e' \mid \exists e'' \in (\Sigma^\pm)^* : e'e'' = e\}$.
- A word e in $(\Sigma^\pm)^*$ is called *an event sequence*.
- An event sequence $e \in (\Sigma^\pm)^*$ is *feasible*, if
$$\forall e' \in \mathit{prefix}(e)\ \forall i, j \in I\ |e'|_{r_{i,j}} \leq |e'|_{s_{j,i}}$$
- An event sequence e is *executable*, if $\delta(e)$ is defined, i.e., $\delta(e) \neq \emptyset$.
- An event sequence is *stable* if (a) it is feasible and (b) it contains the same number of send and receive events of all types.
- For any event sequence e and $i \in I$, $e|_i$ denotes the projection of e over machine P_i, i.e. the sequence of actions from machine P_i.

Definition 2.3 *Two event sequences e_1 and e_2 are* equivalent, *denoted as $e_1 \simeq e_2$, iff for every $i \in I$, $e_1|_i = e_2|_i$.*

Definition 2.4 (*Well-formed SLCFSMs*) *A NSLCFSMs N is* well-formed *if for every executable event sequence e, every feasible event sequence e' equivalent to e is also executable.*

The importance of the concept of *well-formed* SLCFSMs is that it provides a necessary condition for SLCFSMs to be free from deadlocks and unspecified reception errors.

Theorem 2.1 *If a NSLCFSMs N is not* well-formed, *it has deadlocks or unspecified receptions.*

Proof: Let N be a non-well-formed NSLCFSMs. Let e be an executable event sequence in N and $e_i = e|_i$ be the projection of e over P_i. It is easy to show that:

> The set of event sequences resulted from all possible concurrent executions of these e_i's is equal to the set of all feasible event sequences equivalent to e.

Because N is not well-formed, one of these concurrent execution is actually *not* executable. Hence a deadlock or unspecified reception will occur. ∎

From the definition, it is not difficult to see that every two-machine NSLCFSMs is well-formed.

3 Time communicating finite state machines

In this section the new TCFSMs model is defined. As for CFSMs, TCSFMs assume asynchronous semantics of nonblocking sending and blocking receiving, except that both now have to conform to the additional time constraints.

Let $\mathcal{N}$ denote the set of natural numbers, including ∞, and let $\mathcal{N}_\perp = \mathcal{N} \cup \{\perp\}$.

Definition 3.1 (*Time Communicating Finite State Machines*) *A time communicating finite state machine P_i is a four-tuple $(S_i, \Sigma_i^\pm, \delta_i, p_{0i})$, where*

- *S_i is the set of local states of machine P_i.*
- *$p_{0i} \in S_i$ is the start state of machine P_i.*
- $$\Sigma_i^\pm = \sum_{1\leq j\leq n} \Sigma_{i,j} \bigcup \sum_{1\leq j\leq n} \Sigma_{j,i}$$

  where $\Sigma_{i,j}, 1 \leq j \leq n$ is the alphabet of messages that P_i can send to P_j, and $\Sigma_{j,i}, 1 \leq j \leq n$ is the alphabet of messages that P_i can receive from P_j.
- *$\delta_i : S_i \times (\pm\Sigma_i \times I \times \mathcal{N}_\perp \times \mathcal{N}) \rightarrow 2^{S_i}$ is the transition function defined as:*
  - *$\delta_i(p, a, j, t, \sigma)$, where a is either of the form $-m$ or $+m$ for some message type m, and $t \neq \perp$, is the set of states that process P_i could move to from state p after sending a message m to (or receiving a message m from) process P_j. However, this action is allowed only when P_i has been in the local state p for at least t amount of time, and can only be performed before (including) the time $t + \sigma$.*
  - *$\delta_i(p, +m, j, \perp, \sigma)$ is the set of states that process P_i could move to from state p after receiving message m sent by process P_j. However, this action is allowed only when P_i has been in state p for less than or equal to σ amount of time, and must be executed immediately once the desired message m is available in the channel during the time interval between 0 and σ.* ∎

For any local state p in a TCFSM, the two time quantities t and σ associated with each transition originated from p regulate a time span during which the transition can execute. The collection of all these pairs of values define a time span during which some of these transitions can execute. To better understand the above definition, one can imagine that whenever a TCFSM enter a local state p, it will start a clock initialized to zero. A transition $\delta_i(p, a, j, t, \sigma)$ can only execute during the time interval $[t, t + \sigma]$. We say that this transition is *valid* in the interval $[t, t+\sigma]$. Because send transition is nonblocking, the only constraint to a send transition is that it can only execute within the regulated time interval. For a receive transition, besides the time restriction, the desired message must be in the channel during that time interval before it can execute.

The special receive transition $\delta_i(p, +m, j, \perp, \sigma)$ must be executed once the message m appears in the channel during the interval $[0, \sigma]$. This type of transitions can be very useful for modeling signal interruptions that should be handled immediately.

A network of time communicating finite state machines (NTCFSMs) is defined similarly as for a network of communicating finite state machines (NCFSMs). The notions of global states and reachability in NTCFSMs are defined similarly as in NCFSMs.

4 Some properties of TCFSMs

From any given NTCFSMs N, a NCFSMs N' can be constructed by ignoring the time constraint enforced on each transition (or equivalent by replacing each transition (p, a, j, t, σ) in every machine P_i in N with a new transition $(p, a, j, 0, 0)$). Let us call this N' the NCFSMs *underlying* the NTCFSMs N.

4.1 Deadlocks and dead at a local state

The definition of deadlocks for CFSMs is not directly applicable to TCFSMs. There are many subtle communication scenarios that have to be carefully defined and clarified.

4.1.1 Late arriving or early arrived messages

Consider portion of a NTCFSMs N_1 depicted in Fig. 3. Assume that machine P_1 and Q_1 arrive in local states p and q respectively at the same time. There are three possible interactions between P_1 and P_2:

(a) $t_1 + \sigma_1 < t_2$. In this case, message m will arrive too early to be received by Q_1.

(b) $t_2 + \sigma_2 < t_1$. In this case, message m will arrive too late to be received by Q_1.

(c) The intervals $[t_1, t_1 + \sigma_1]$ and $[t_2, t_2 + \sigma_2]$ overlap.

Both cases (a) and (b) indicate communication errors. Machine Q_1 is said *dead at state q* in both cases. The validity of case (c) depends upon the semantics used, which will be discussed in the next subsection. It is interesting to note that in TCFSMs a single machine can be *dead* at a local state, while a deadlock in CFSMs always involve more than one machine.

Transitions of the form $\delta(p, -m, j, t, \infty)$, where $t \neq \infty$, present another problem. The corresponding receiving events must be able to wait for arbitrarily long for communications to take place. This could be a communication problem in many cases. On the other hand, this type of transitions are useful to model situations such as sending a shutdown message by an operating system.

4.1.2 Forced execution vs. unforced execution

Consider a local state p with only two receive transitions $(p, +m_1, j_1, t_1, \sigma_1)$ and $(p, +m_2, j_2, t_2, \sigma_2)$ in a TCFSM. There are at least three ways that these two transitions can interact with the rest of the network:

(a) At least one of the two messages m_1/m_2 is available in the time interval $[t_i, t_i + \sigma_i]$, and the corresponding receive transition is executed.

(b) None of the two messages m_1/m_2 is available in the time interval $[t_i, t_i + \sigma_i]$, and hence the TCFSM is *dead* in state p.

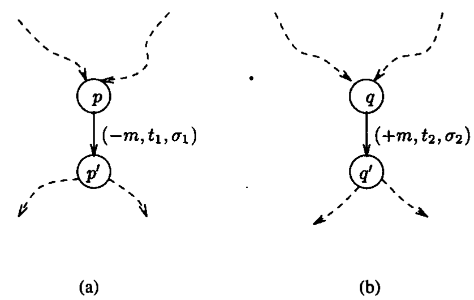

Figure 3: A possible communication problem: (a) Machine P_1; (b) Machine Q_1

(c) At least one of the two messages m_1/m_2 is available in the time interval $[t_i, t_i + \sigma_i]$, but none of the two corresponding receive transitions is being elected to execute during its valid interval. The TCFSM is again *dead* in state p.

We call the semantics stated in case (c) above *unforced semantics*:

No transitions from any local state can be forced to be executed.

An alternative is *forced semantics*:

When execution reaches any given local state p, if there ever exists at least one valid transition out of p, then one valid transition will be executed.

The forced semantics seems more reasonably and is adopted in this paper. With this semantics, case (c) described above is never possible, and case (b) is the same as case (a) and (b) in last subsection (the machine is dead at a local state).

4.2 Relationships between TCFSMs and CFSMs

With the force execution semantics and from the definition, it can be easily seen that any NCFSMs N' is equivalent to a NTCFSMs N that is obtained by replacing each transition in every machine $\delta_i(p, a, j)$ in N by a transition $\delta_i(p, a, j, 0, 0)$.

It is easy to see that, due to the time constraints, a reachable global state in its underlying NCFSMs may no longer be reachable in the NTCFSMs itself. In addition, it is also apparent that any reachable global state in N is also a reachable global state in N'. This is because each transition $(p, a, j, 0, 0)$ in N' can simulate all possible timings of the corresponding transition (p, a, j, t, σ). Hence we have the following theorem.

Theorem 4.1 *Let N be a NTCFSMs, and N' be the NCFSMs underlying N. Then*

$$\mathrm{RS}(N) \subseteq \mathrm{RS}(N')$$

The concept of *unspecified receptions* in TCFSMs is similar to that in CFSMs with obvious modifications. The following theorem can be directly obtained from the relevant definitions and the above theorem.

Theorem 4.2 *Let N be a NTCFSMs, and N' be the NCFSMs underlying N. The following statements are true with respect to N and N'.*

(1) If N has unspecified receptions or unbounded communications, then N' also has unspecified receptions or unbounded communications. The reverse, however, is not true.

(2) N' can be free of deadlocks while N has dead states, and vice versa.

5 Modeling with TCFSMs

In this section we show through two practical examples how to specify and verify communication protocols using TCFSMs. The concept of self-stabilization used here is based on reachability sets [3]. The notation *LRS*(N) denotes the set of *all legal global states*. Informally, a system N self-stabilizes if it either stays in the set *LRS*(N) if there is no perturbation, or returns to one of states in *LRS*(N) within a finite number of steps if it strays from *LRS*(N).

5.1 Example 1: a token ring

Fig. 4 shows a simplified token ring modeled by CFSMs. It is simplified because neither the case of loss of the token nor the case of multiple tokens are handled. In any practical token ring, these two problems can disrupt the entire operation.

The NTCFSMs N_3 shown in Fig. 5 models a more realistic version of the token ring shown in Fig. 4. Here it is assumed that tokens can get lost or duplicated during communications. To model loss or duplication of tokens, following the practices in CFSMs, n TCFSMs C_i modeling the unreliable communication media are created. In that figure, $\alpha = (n-1)\sigma$, $t_1' = \Sigma_{i=2}^{n} t_i$, $t_1'' = t_1' + \alpha + \epsilon_1$, and $t_1''' = t_1' - \epsilon_2$, where $\epsilon_1, \epsilon_2 > 0$ are two constants. t_i represents the *transmission and propagation* delay from station P_i to P_{i+1}.

The network N_3 implements a token ring together with a simple ring management algorithm described as follows. Station P_1 is chosen as the ring coordinator. It is assumed that the minimum and maximum delays between two neighbor stations are constants (the value t_i). The algorithm also assumes that P_1 knows the total delay from station P_2 to station P_1 (through stations $P_3, P_4, \cdots, P_n$). In addition, P_1 also knows the maximum time interval for which each station can keep the token (the value σ). After sending out the token to its successor station, station P_1 will set two timers. The first timer is intended to detect loss of the token and is set to a value slightly larger than the worst case token circulation time. The second timer is intended to check for existence of multiple tokens and is set to a value slightly smaller than the shortest token circulation time.

We now show that N_3 correctly implements the above ring management algorithm and is in fact self-stabilizing.

Theorem 5.1 *The token ring depicted in Fig. 5 is self-stabilizing.*

Proof: The set of legal states *LRS*(N_3) contains all the global states satisfying one of the following two conditions:

- Every machine is in its local state 0, there is a message m in a channel $P_i \to C_i$ or $C_i \to C_{i+1}$, and every other channel is empty;
- There is a machine P_i or C_i that is in its local state 1, every other machine is in its local state 0, and every channel is empty.

It is interesting to notice that the initial global state, where every machine is in its local state 0 and every channel is empty, is not in *LRS*(N_3). This is where the power of self-stabilization shows up. After t_1'' amount of time, the transition $0 \xrightarrow{(-m,t_1'',0)} 0$ in machine P_1 will execute and a message m is sent to machine C_1, leaving N in a legal state. It is easy to show (by induction) that as long as every transition $0 \xrightarrow{(-m,0,0)} 0$ or $0 \xrightarrow{(+m,\perp,\infty)} 0$ in every machine C_i does not execute, N_3 will remain in the set of legal states.

Now assume that the transition $0 \xrightarrow{(+m,\perp,\infty)} 0$ in some machine C_i executes at some point, which generates one more message m. When P_1 sent out the message m last time, it returned to its local state 0. There are now two messages m circulating around the ring. These two messages have to be received by P_1 one by one. As a result, when the second message is received, it must arrive in less than t_1' time. The transition $0 \xrightarrow{(+m,\perp,t_1''')} 0$ in machine P_1 will then take away the second message m, bringing N_3 back to a legal global state.

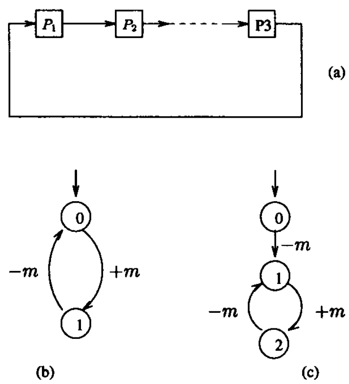

Figure 4: A token ring net modeled by a NCFSMs N_2: (a) Topology graph; (b) Machines P_i, $i = 2, 3, \cdots, n$; (c) Machines P_1

The case where the transition $0 \xrightarrow{(-m,0,0)} 0$ in some C_i executes can be proved similarly as for the initial global state. This completes the proof. ∎

The ring net in Fig. 5 is asymmetric. This is in agreement with the conclusions in [4]. In particular, it is not difficult to see that there is no symmetric version of self-stabilizing extension of the ring net (it is impossible to construct the transition from state 0 to state 1 in each machine P_i, $1 \leq i \leq n$ because they are interdependent).

5.2 Example 2: a sliding window protocol

Fig. 6 shows a NCFSMs N_4 modeling a sliding window protocol with both send and receive windows equal to 1. The communication media are modeled by the two machines C_1 and C_2. For simplicity of presentation, it is assumed that messages can get lost, but cannot get duplicated during communications. It is easy to see that N_4 is actually self-stabilizing. The only problem is the that existence of the two self-loop send transitions $1 \xrightarrow{-m_0} 1$ and $3 \xrightarrow{-m_1} 3$ (which model timeout) causes the communication unbounded.

In Fig. 7, the same sliding window protocol is modeled by a NTCFSMs, where t_1 and t_2 are the (transmission and propagation) delays between the two TCFSMs respectively. Only machines P_5 and Q_5 are redrawn. Assume that the four self-loop transitions in both C_1 and C_2 can only execute a predefined number of times during a given unit of time (say one second), N_5 is both self-stabilizing and bounded in communications. The proof is simple and hence omitted.

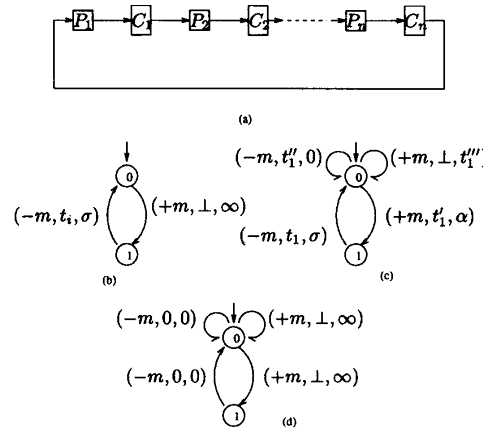

Figure 5: A self-stabilizing token ring modeled by a NTCFSMs N_3: (a) Topology graph; (b) Machines P_i, $i = 2, 3, \cdots, n$; (c) Machine P_1; (d) The channel machines C_i, $i = 1, 2, \cdots, n$

6 Concluding remarks

We have proposed two variants of the classical CFSMs model. To the author's best knowledge, both the SLCFSMs and TCFSMs models are innovative. As variants of the classical CFSM model, we believe that they can be used to formally specify many practical communication protocols.

The relative modeling power of the CFSM model and the SLCFSM model is interesting. As it is known that the class of two-machine NCFSM has the full power of a Turing machine [1], and any two-machine NCFSM is also a NSLCFSM, we conjecture that these two models have the equal modeling power.

Whether the well-formedness property of SLCFSMs is decidable is still an open problem, although we conjecture that it is undecidable.

The proposed TCFSMs model retains all the advantages of the classical CFSMs model. Its added ability of expressing time constraints makes it a very promising candidate for modeling delay-sensitive systems, as demonstrated by its clean and concise modeling of the two example protocols to be presented later. In fact, the TCFSMs model can be viewed as an advanced form of the classical CFSMs model. Existing specification and verification methods for CFSMs can be used at initial stages of modeling using TCFSMs. New methodologies for specifying and verifying TCFSMs are needed. The crucial problem is *timing analysis*, which should reveal the inter- and intra-dependencies of time constraints of transitions.

As a new alternative design tool, many questions

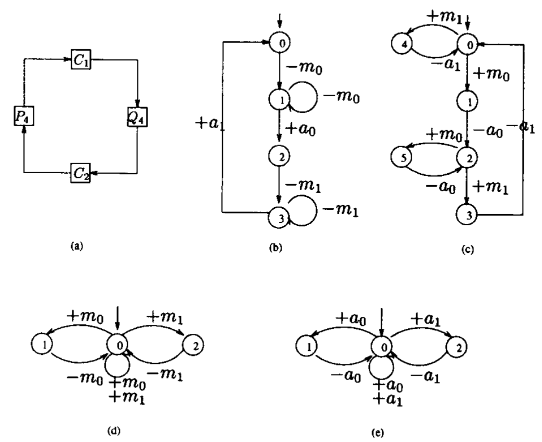

Figure 6: A NCFSMs N_4 modeling a sliding window protocol with both send and receive window equal 1: (a) Topology graph; (b) Machine P_4, the sender; (c) Machine Q_4, the receiver; (d) Machine C_1; (e) Machine C_2

about TCFSMs still remain to be investigated. The semantics still need to be polished. Even the current method of associating time constraints to CFSMs is debatable. However, we feel our proposal is one step further toward a right direction.

Finally it should be noted here that the notion of associating time constraints to transitions is from Merlin's paper about Time Petri nets [8].

References

[1] D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, **30**(2):323-342, 1983.

[2] G.M. Brown, M.G. Gouda, and C. Wu. Token systems that self-stabilize. *IEEE Trans. on Software Engi.*, **14**(6), June. 1989, pp.845-852.

[3] L. Cherkasova, R. Howell, and L. Rosier. Bounded self-stabilizing Petri nets. *Proc. of the 12th Annual Petri Net Conference*, Dec. 1991.

[4] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of Association for Computing Machinery*, **17**(11), Nov. 1974, pp.643-644.

[5] S. Ghosh. Stabilizing Petri nets. *Proc. of 1991 IEEE International Conference on Parallel and Distributed Systems*, Dec. 1991.

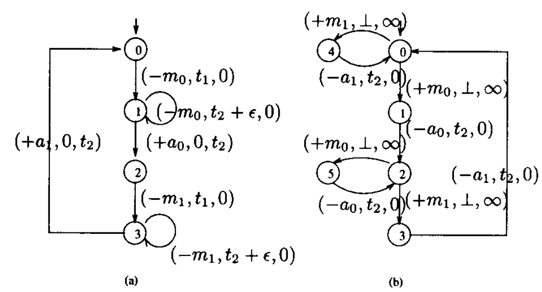

Figure 7: A NTCFSMs N_5 modeling a sliding window protocol with quantified time-out intervals: (a) Machine P_5, the sender; (b) Machine Q_5, the receiver

[6] M. Gouda, E. Manning, and Y. T. Yu. On the Progress of Communication between Two Finite State Machines. *Information and Control*, **63**(3), 1984, pp.308-320.

[7] M.G. Gouda and N.J. Multari. Stabilizing communication protocols. *Technical Report, TR-90-20*, Dept. of Comp. Sci, The Univ. of Texas at Austin, June 1990.

[8] P.M. Merlin and D.J. Farber, Recoverability of communication protocols - implecations of a theoretical study. *IEEE Trans. on Comm.*, Vol.COM-24, Sept. 1976, pp. 1036-1043.

[9] W. Peng. Petri nets and self-stabilization of communication protocols. *Proc. of 1993 International Conference of Network Protocols*, San Francisco, Oct. 1993, pp. 166-174.

[10] W. Peng and S. Purushothaman. A Unified Approach to Deadlock Detection Problem in Networks of Communicating Finite State Machines. *Lecture Notes in Computer Science*, No. 531, Springer, 1991, pp. 243-252.

[11] W. Peng and S. Purushothaman. Analysis of a class of communicating finite state machines. *Acta Informatica*, **29**, pp.499-522, 1992.

[12] Y. T. Yu and M. G. Gouda. Deadlock detection for a class of communicating finite-state machines. *IEEE Transaction on Communications*, COM-**30**(12):2514-2518, Dec. 1982.