BOUNDEDNESS, EMPTY CHANNEL DETECTION AND SYNCHRONIZATION FOR COMMUNICATING FINITE AUTOMATA

Louis E. Rosier and Hsu-Chun Yen

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

TR-84-13 May 1984

ABSTRACT

In this paper, we consider networks of communicating finite state machines (CFSM's), that explicitly allow zero testing (i.e. empty channel detection). In our main result, we show that the boundedness problem is decidable for the class of FIFO networks consisting of two such CFSM's, where one of the two machines is allowed to send only a single type of message to the other. This result, we feel, is somewhat surprising since the zero testing capability is precisely the required extension needed in order to render the problem undecidable for the related class of vector addition systems with states (VASS's) of dimension two. Note that both have the ability to store two nonnegative integers which can be conditionally tested for zero. The reason for the disparity appears to be that such a class of extended VASS's would be capable of more synchronized behavior (since the actions of the two counters can be controlled by a single finite state control). The rest of the paper examines other classes of networks which allow empty channel detection. These results seem to indicate that our main result can not be extended.

Table of Contents

1. INTRODUCTION

Models for distributed communication systems have included Petri nets, or equivalently Vector Addition Systems(VAS's), and more recently networks of communicating finite state machines(CFSM's). Many communication protocols can be modeled as a network of two finite state machines that communicate by exchanging messages over two one directional, FIFO channels[3,4,7,19,23,26]. (Generalizations of this model permit any number of CFSM's, each pair of which communicate as above.) Such models have been shown useful in the detection of many protocol design errors. Design errors considered in the literature include state deadlocks, unspecified receptions, nonexecutable receptions, channel boundedness and channel overflow (c.f. [3,4,7,19,23,26,27]). Petri nets have also been used to model communication protocols. (See e.g. [2,17,24].)

Informally, the communication between two CFSM's machines is said to be bounded iff there is a nonnegative integer k such that in each "reachable state" of the network, the number of messages in each channel is no more than k (i.e the number of distinct reachable network states is finite). Similarly, a VAS is bounded iff each vector position is so bounded. If the channels in any such protocol are bounded, then the protocol can be validated by generating the set of all reachable states and checking this set for any of the aforementioned problems. If the channels, on the other hand, are potentially unbounded, then the network cannot be built. Consequently, a basic problem to consider concerning CFSM's (and VAS's), is whether the communication in a given network is bounded. Unfortunately, this problem is known to be undecidable in general[4]. However, for VAS's and certain restricted classes of CFSM's this problem is decidable[4,6,7,12,18,19,26]. In this paper, we closely examine what features of a communication system, modeled by various extended types of VAS's and networks of CFSM's, contribute to the undecidability of the boundedness problem. We find that the asynchronous behavior of such systems plays an important role in this problem, and our results indicate that in some simple cases asynchronous systems may be easier to analyze than their synchronous counterparts. Before proceeding, however, we give some important historical background on known results concerning various types of VAS's and CFSM's.

In [6], the boundedness problem was studied for the class of networks consisting of an arbitrary number of CFSM's, each pair of which communicate by exchanging a single type of message. Since such networks can be modeled by VAS's[12], decision procedures for this class are well known[12,18]. In particular, each such network can be represented by a VAS, where each channel corresponds to a potentially unbounded vector position and the state of each machine corresponds to a sequence of bounded vector positions. Equivalently, such a network can be modeled by a vector addition system with states (VASS) [9], where the channels are represented as above but where the states are represented in the states of the VASS (i.e. a state of the VASS contains a state for each CFSM in the network). The main intuitive difference, in these models, is that a VAS does not readily illustrate the asynchronous behavior of the CFSM's; and in fact may camouflage it. On the other hand, a VAS may exhibit very synchronous behavior. For this reason, we feel that VAS's (and VASS's) are not good models for asynchronous communication systems as there seems to be no clear way to see that two or more independent entities are executing simultaneously, which can communicate, only by sending and receiving messages. Networks of CFSM's, however, do illustrate this desired asynchronous behavior.

The boundedness problem was examined for networks consisting of two CFSM's, each of which could only send a single type of message to the other machine, in [26], and a more efficient algorithm than the one given in [6], was presented. This result was extended in [19], where the class of communication networks consisting of two CFSM's, in which one of the machines sends only one type of message (the communication in the other direction is not constrained), was shown to have a decidable boundedness problem. In fact, the problem was shown to be nondeterministic logspace complete and thus boundedness can be decided in polynomial time[5]. Both of the preceding results[19,26], were derived by taking advantage of the networks asynchronous properties. As a result, the techniques (and thus the complexity results) do not appear to generalize to the class of VAS's which have no more than two potentially unbounded positions (or equivalently, to the class of VASS's of dimension two).

These models of communication systems have not allowed, with the exception of [20], the communicating entities to realize or act upon any information regarding the channels, with the exception of reading the next available message. For example, no process is allowed to determine if a channel is devoid of messages and move accordingly. Examples illustrating the limitations of the modelling power of VAS's are considered in [1] and [13]. See also [15-17]. In both cases, the limiting factor is precisely the inability of the VAS to test a potentially unbounded position for zero and take action on the outcome of the test. As a result, the literature contains many extensions to the basic model of VAS's (Petri nets). Such extensions use a variety of mechanisms each of which allows zero testing. These include "inhibitor arcs", "constraints", "priorities", "timing constraints", etc. (c.f. [8,17]). Recently, priority networks of CFSM's were introduced, where messages are received based on a fixed, partial-ordered priority relation[7]. (Unrelated messages can be received in any order.) This model is equivalent, in computational power, to certain classes of extended Petri nets, in particular those with priority tokens[8]. However, the CFSM model is more concise (since the channels and their contents are not modeled explicitly), and so is more convenient to use in modelling communication protocols and distributed systems.

The results in [7] focus on the problem of whether the communication of a priority network is bounded. Specifically, it is shown that the problem of detecting boundedness is undecidable even if the machines exchange only two types of messages. Also considered is the case where one of the two channels is known to be bounded, and it is shown that three types of messages can make the problem undecidable in this case. (This problem is decidable in the case of FIFO networks[4].) The same problem becomes decidable if only two types of messages are allowed. Also considered is the case where one of the two machines sends one type of message. The problem is undecidable if the other machine sends three or more types of messages, and is decidable if the other machine sends two or less types of messages. (Both problems are decidable in the case of FIFO networks[4].) However, the latter result can be generalized to the case of three or more messages, if only two message types are mentioned in the priority relation. If the priority relation is the null set (i.e. all messages are received on a random basis), then the boundedness problem is essentially the same as that of a VAS. (Such machines are called Random CFSM's in [7].)

In most of the extended VAS models, where zero testing is allowed, only two potentially unbounded positions are necessary to render the boundedness problem undecidable. This is also the case for the undecidability results concerning Priority CFSM's. The reason is that such extended

VAS's or networks can utilize the potentially unbounded positions (or channels) to store two nonnegative integers and thus can be used to simulate the computation of a two-counter machine [14] with no input. See [7,8,17]. Since the computational power of two-counter machines is equivalent to that of TM's, the result follows.

In this paper, we consider networks of CFSM's (FIFO, Priority and Random), that explicitly allow zero testing (i.e. empty channel detection). In section 3, we consider FIFO networks of two CFSM's, only one of which is restricted to send a single type of message, where each machine can, make the following types of transitions:

1. moves in which a message is sent or received
2. ϵ-moves
3. conditional moves in which the input channel is checked for emptiness
4. conditional moves in which the output channel is checked for emptiness

For this class, we are able to show that the boundedness problem is nondeterministic logspace complete. Such machines are clearly a generalization of those studied in [19], where only moves of the first type were allowed. Recall that the boundedness problem was also shown to be decidable in nondeterministic logspace for this simpler class. The approach taken in [19] was to construct a deterministic one counter automaton (doca)[25], that would simulate, in some sense, the computation of a given network. The results then followed from properties of doca. Here, however, we consider a class of networks that cannot, in general, be simulated by doca; and hence the techniques of [19] do not appear to generalize when moves of type 3-4 are allowed. On the other hand, our approach is similar in that we construct a simulating automaton for each network. However, the automata we consider are more powerful than doca's and are, as far as we know, unique. The difference in the networks considered here, of course, is that the two channels can be conditionally tested for zero (and nonzero) by each machine. Note, that since a machine can test both its input and output channel for emptiness, it can therefore ascertain something about the computation of the other machine (and hence some synchronized behavior may result). This result, we feel, is surprising since the zero testing capability was precisely the required extension that the class of VAS's, with two potentially unbounded positions (or equivalently, the class of VASS's of dimension two), needed in order to render the boundedness problem undecidable. Note that both have the ability to store two nonnegative integers which can be conditionally tested for zero. The reason for the disparity seems to be that such a class of extended VAS's would be capable of more synchronized behavior (since the actions of the two counters can be controlled by a single finite state control). The same synchronous capabilities can be instilled in these CFSM's, however. For example, if we allowed the CFSM's to have the "ability to synchronize" provided by a single "bit" of "shared memory", the problem would become undecidable. (This would not be the case if only moves of the first two types were allowed.)

In section 4, we examine the simplest classes of FIFO, Priority and Random empty-channel-detecting CFSM's for which the boundedness problem remains undecidable. These show that the result in section 3 cannot be extended. In particular, we show that the boundedness problem is undecidable for the following classes:

1. Networks of three empty-channel-detecting FIFO CFSM's. This result holds even if

each machine is restricted to send a single type of message, and only one of the machines is allowed type 3-4 moves. Only type 3 (4) moves are needed, however, if two of the machines are allowed conditional moves. (Recall that the boundedness problem is undecidable for networks of two FIFO CFSM's, which exchange two types of messages, even when no moves of type 2-4 are allowed[4].)

2. Networks of two empty-channel-detecting Priority CFSM's where one machine sends a single type of message and the other machine sends two types of messages. This result holds even if each machine is restricted to only moves of type 1-3 (or type 1-2 and 4).

3. Networks of two empty-channel-detecting Random CFSM's where one machine sends a single type of message and the other machine sends two types of messages. This result holds even if each machine is restricted to only moves of type 1-2 and 4.

Without the empty channel detection capability, each of the three aforementioned problems becomes decidable. See [4,7]. Results 2 and 3 indicate that the FIFO result of section 3 is somewhat of an anomaly in that no corresponding case arises for Priority and/or Random Networks.

In the last section, we consider priority networks where the priority relation is not fixed. For example, consider the case where a (possibly) different priority relation was assigned to each node in the CFSM's. Most of the results given in [7] extend in a straightforward manner to priority networks of this type. However, the results concerning priority relations that mention only two message types do not seem to generalize. (In fact, some are no longer true.) Here, we illustrate the differences resulting when the priority relation is allowed to vary from node to node and we use the techniques of section 3, to handle the case of two machines where one sends a single type of message and the other sends at most two types of messages.

2. COMMUNICATING FINITE STATE MACHINES

FIFO Networks

A (empty-channel-detecting) CFSM M is a directed labelled graph with the following five types of edges: *sending edges, receiving edges, empty input channel edges, empty output channel edges,* and ϵ *edges.* A sending (receiving) edge is labelled send(g) (receive(g)) for some message g in a finite set G of messages. An empty input (output) channel edge is labelled E_i (E_o), and may be traversed only when the input (output) channel is empty. An ϵ edge is labelled ϵ, and may be unconditionally traversed. One of the nodes in M is identified as the initial node; each node is reachable by a directed path from the initial node. A *loop* l: $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_n$ is a directed path on M, where $p_1 = p_n$ and $p_i \neq p_j$ for all i and j, $i<j$ where $i \neq 1$ and $j \neq n$. The characteristic value of l, denoted by $C(l)$, is equal to (# of sending edges - # of receiving edges) in l.

Let M and N be two CFSM's with the same set G of messages; the pair (M,N) is called a *FIFO network* of M and N. A *state* of network (M,N) is a four-tuple (v,w,x,y), where v and w are two nodes in M and N respectively, and x and y are two strings over the messages of G. Informally, a state (v,w,x,y) denotes that the executions of M and N have reached nodes v and w respectively, while the input channels of M and N contain the message sequences x and y, respectively.

The *initial state* of a FIFO network (M,N) is (v_0,w_0,E,E) where v_0 and w_0 are the initial nodes in M and N respectively, and E denotes the empty channel.

Let s=(v,w,x,y) be a state of network (M,N); and let e be an outgoing edge of node v or w. A state s' is said to *follow* s *over* e iff one of the following ten conditions are satisfied:

i. e is a sending edge, labelled send(g), from v to v' in M, and s'=(v',w,x,y'), where y'=y.g, ("." is the usual string concatenation operator).

ii. e is a sending edge, labelled send(g), from w to w' in N, and s'=(v,w',x',y), where x'=x.g.

iii. e is a receiving edge, labelled receive(g), from v to v' in M, and s'=(v',w,x',y), where x=g.x'.

iv. e is a receiving edge, labelled receive(g), from w to w' in N, and s'=(v,w',x,y'), where y=g.y'.

v. e is an empty input channel edge, labelled E_i, from v to v' in M, x=E and s'=(v',w,E,y).

vi. e is an empty input channel edge, labelled E_i, from w to w' in N, y=E and s'=(v,w',x,E).

vii. e is an empty output channel edge, labelled E_o, from v to v' in M, y=E and s'=(v',w,x,E).

viii. e is an empty output channel edge, labelled E_o, from w to w' in N, x=E and s'=(v,w',E,y).

ix. e is an ϵ edge, labelled ϵ, from v to v' in M, and s'=(v',w,x,y).

x. e is an ϵ edge, labelled ϵ, from w to w' in N, and s'=(v,w',x,y).

Let s and s' be two states of a FIFO network (M,N); s' *follows* s iff there is a directed edge e in M or N such that s' follows s over e.

Let s and s' be two states of (M,N); s' is *reachable from* s iff s=s' or there exist states $s_1,\ldots,s_r$ such that $s=s_1$, $s'=s_r$ and s_{i+1} follows s_i for i=1,...,r-1.

A state s of a FIFO network (M,N) is said to be *reachable* iff it is reachable from the initial state of (M,N).

A *finite computation* (path) of a network is a sequence of states $s_0,\ldots,s_r$ in which s_0 is the initial state of the network and s_{i+1} follows from s_i, $0\leq i\leq r-1$. An *infinite computation* is such a sequence of infinite length.

The communication of a FIFO network (M,N) is said to be *bounded* iff there exists a non-negative integer k such that for any reachable state (v,w,x,y), $|x|<k$ and $|y|<k$ where |x| is the number of messages in the string x. If there is no such k, then the communication is *unbounded*.

B. Priority Networks:

Let M and N be two CFSM's, as defined earlier, with the same set G of messages, and let $<$ be a partial ordering on G. The triple (M,N,$<$) is called a ***priority network***, where $<$ is called the ***message priority relation*** of the network. If two distinct messages g_1 and g_2, in G, are such that (g_1,g_2) is in $<$, denoted by $g_1<g_2$, then g_2 is said to have a ***higher priority*** than g_1.

A *state* of a priority network (M,N,$<$), over G, is a four tuple (v,w,x,y), where v is a node in M, w is a node in N, and x and y are two multisets of messages in G.

The ***initial state*** of (M,N,$<$) is (v_0,w_0,E,E), where v_0 is the initial node of M, w_0 is the initial node of N, and E denotes the empty multiset.

Let s=(v,w,x,y) be a state of a priority network (M,N,$<$), and let e be an outgoing edge of node v or w. A state s' of (M,N,$<$) is said to ***follow*** s ***over*** e iff one of the following ten conditions are satisfied:

i. e is a sending edge, labelled send(g), from v to v' in M, and s'=(v',w,x,y') where y' is obtained from y by adding one message g.

ii. e is a sending edge, labelled send(g), from w to w' in N, and s'=(v,w',x',y) where x' is obtained from x by adding one message g.

iii. e is a receiving edge,labelled receive(g), from v to v' in M, ***and*** s'=(v',w,x',y) where x contains at least one g, and x' is obtained from x by removing exactly one g, ***and*** if v has an outgoing edge labelled receive(g'), where g$<$g', then x may contain no message g'.

iv. e is a receiving edge,labelled receive(g), from w to w' in N, ***and*** s'=(v,w',x,y') where y contains at least one g, and y' is obtained from y by removing exactly one g, ***and*** if w has an outgoing edge labelled receive(g'), where g$<$g', then y may contain no message g'.

v. - x. are defined the same as for FIFO networks.

The last parts of conditions iii and iv mean that messages are received in accordance with their priorities; the message with the highest priority must be received first; unrelated messages can be received in any order.

The definitions of reachability, computation and bounded communication for a priority network are similar to those, discussed earlier, for a FIFO network.

C. Random Networks:

A *random network* is a priority network whose message priority relation is the empty set.

3. THE MAIN RESULT

In this section, we provide a nondeterministic logspace decision procedure, to determine whether the communication is unbounded, for the following class of communication networks:

C: The class of FIFO networks consisting of two (empty-channel-detecting) CFSM's in

which at least *one* of the two machines sends only one type of message to the other machine (i.e. each edge labelled by a send, in one of the machines, mentions the same message of G).

Since the problem has already been shown to be nondeterministic logspace hard [19], for the subclass of C, in which only sending and receiving edges were allowed, it follows immediately that the boundedness problem, for these classes of networks, is nondeterministic logspace complete. (See [10] for motivations and definitions of nondeterministic logspace hard, nondeterministic logspace complete, etc. See also [11,21,22].)

These results do generalize, to some degree, the earlier results concerning boundedness in [19]. A perhaps more interesting comparison, however, is the related problem for extended VAS (or, equivalently VASS) models, where some sort of zero testing is allowed. In most of the extended VAS (VASS) models, where zero testing is allowed, only two potentially unbounded positions are necessary to make the boundedness problem undecidable. (Note that the problem is decidable[12,18] when zero testing is not allowed, i.e. for VAS's and VASS's.) However, here we show that in some cases the introduction of the zero testing capability for CFSM's does not make the boundedness problem harder. This result, we feel, is surprising since the zero testing capability was precisely the extension that the class of VAS's needed in order to render the boundedness problem undecidable. The difference seems to be that such a class of extended VAS's would be capable of more synchronous behavior (since the actions of the two counters can be controlled by a single finite state control). The same synchronous capabilities can be instilled in these CFSM's, however. For example, if we allowed the CFSM's to have the "ability to synchronize" provided by a single "bit" of "shared memory", the problem would become undecidable. (This would not be the case if only sending, receiving and ϵ edges were allowed.)

Based on above evidence, we find that the asynchronous behavior of these systems plays an important role in the boundedness problem, and thus our results indicate that in some simple cases asynchronous systems may be easier to analyze than their synchronous counterparts.

Let (M,N) be an arbitrary network in C. (See Figure 3.1.) Without loss of generality, we assume that N sends only one type of message to M. A useful technique [19] for dealing with networks of two CFSM's, of this variety, is to run the machine N faster than the other; always insuring that no more than a single message is in C_N (N's input channel) at all times. The resulting computations of the network can then be simulated by a one counter automata, whose counter is bounded in all possible computations iff C_M (M's input channel) is bounded in the network. (The boundedness of C_N is handled separately. See [19].) However, when empty channel edges are allowed, it is not always the case that the restricted computations, where C_N contains at most one message, have the same related boundedness property. For instance, consider the unbounded network (M,N) in figure 3.2. The outgoing edge at node 3 (of M) is an E_0 move. This move can be executed only when C_N is empty, which implies that N has to first transfer all the messages in C_N to C_M. Similarly, the following E_i move on M can be utilized only by first moving all the messages in C_M back to C_N. (In order to do this, M must execute the loop between nodes 4 and 5 (of M) until C_M is empty.) By letting M repeatedly execute the loop containing node 1, in this manner, it can be seen that the channel contents of (M,N) can grow arbitrarily large. In fact, this loop must be executed for the channel contents of (M,N) to grow. It appears, therefore, that a

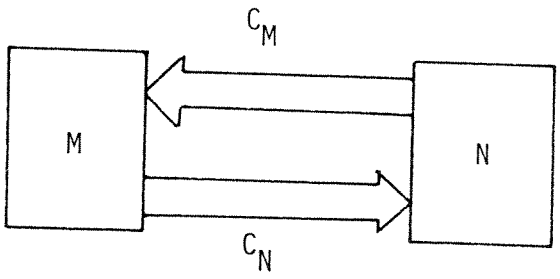

Figure 3.1 A class C newtork (M,N).

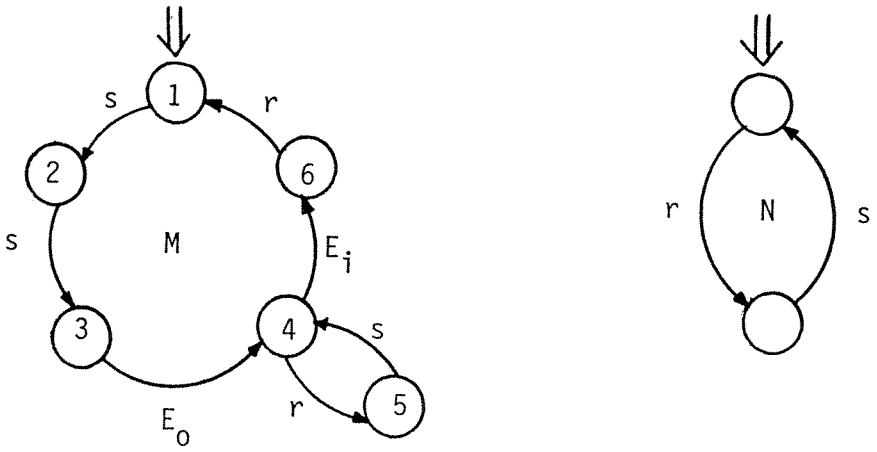

Figure 3.2 An unbounded network (M,N).

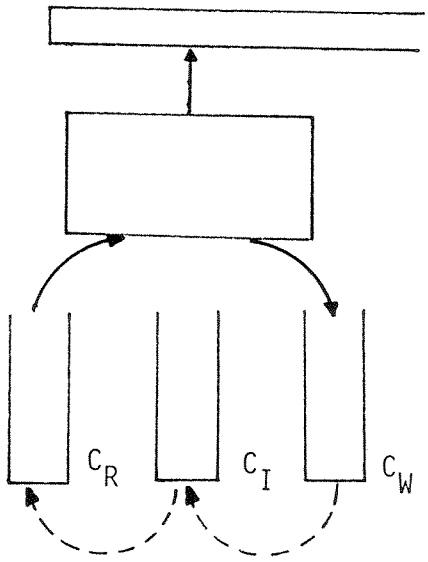

Figure 3.3 A restricted 3-counter machine.

simulation by one counter automata will not work in the case of C networks, because all the messages cannot be stored in a single channel. To overcome this difficulty, we introduce a new class of automata, which we call *restricted 3-counter machines*, that are better able to simulate C networks. Later, we will show that the boundedness problem for the class of simulating machines is decidable in nondeterministic logspace. As a result, we will have that the boundedness problem for C networks is nondeterministic logspace complete.

As shown in figure 3.3, a *restricted 3-counter machine* W is a device with a finite state control, a one-way read-only input tape, and three counters (C_R, C_I, and C_W), each of which is capable of storing a nonnegative integer. (Actually, we refer to them as storing a nonnegative number of *tokens*.) Roughly speaking, the actions of W are to read an input and then depending on the current state and the contents of the counters, W can change it's state, alter the contents of counters, (either add one to C_W or subtract one from C_R) and move its input head 0 or 1 positions to the right. (More will be said about the operation of W shortly.) Independent from W, tokens may arbitrarily be moved from C_W to C_I and/or from C_I to C_R. We ascribe this action with what we call the *daemon*. It's actions are to be considered totally asynchronous with those of W. Note that W can only subtract tokens from C_R or add tokens to C_W; and that the daemon can only move tokens from C_W to C_I or from C_I to C_R. No other movement of the tokens is allowed.

Formally, the machine W is a 7-tuple $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, where Q is the set of states, Σ is the (finite) input alphabet, $\Gamma=\{Z_0,B\}$ is the stack alphabet (note that the bottom-of-stack symbol Z_0 can never be erased), δ is the transition function which will be described in detail later, q_0 is the initial state, Z_0 is the bottom-of-stack symbol, and F is the set of accepting states. A configuration c, of W, is described by a 4-tuple (q,x,y,z), where $q\epsilon Q$ represents the current state, x,y,and z are nonnegative integers representing the contents of C_R, C_I, and C_W, respectively. $|c|=x+y+z$. $c_0=(q_0,0,0,0)$ is the initial configuration. A move $c\rightarrow c'$ is a transition (specified by the transition function) that leads from configuration c to c'. A *computation* $c-w\rightarrow c'$ is a sequence of transitions, beginning in configuration c and ending in c', which causes the machine W to read the input string w. A configuration c is said to be *reachable* iff there exists a computation $c_0-w\rightarrow c$ for some input string w. The machine W is said to be *unbounded* iff for every integer $k>0$, there exists a reachable configuration (q,x,y,z), such that $(x+y+z)>k$. We say that W accepts an input w iff $c_0-w\rightarrow q_f$, for some q_f in F. A *loop* is a sequence of moves l: $c_1\rightarrow c_2\rightarrow\ldots\rightarrow c_n$ such that c_1 and c_n are in the same state, and no other intermediate configurations have this property. We define the *characteristic value* of l, denoted by $C(l)$, to be (# of w moves - # of r moves) in l, where w and r moves stand for "write to C_W" and "read from C_R", respectively.

Now we define the moves or transitions of W. Let W be represented as a directed graph whose set of vertices is Q and whose edges are the transitions . All edges, in this graph, can be given a *label* which describe two things—a precondition on the counter values necessary in order for this transition to execute and the subsequent change in the counter values due to the transition being executed. (The associated state change is, of course, implicit.) The possible labels and their semantics is given as follows:

	Label	Counter Precondition	Subsequent Counter Action
1.	C_{Rj} $(0 \leq j < \|M\|)$	$C_R = j$	none
2.	C_{Ij} $(0 \leq j < \|M\|)$	$C_I = j$	none
3.	C'_{R0}	$C_R > 0$	none
4.	C'_{I0}	$C_I > 0$	none
5.	r (read)	$C_R > 0$	$C_R := C_R - 1$
6.	w (write)	none	$C_W := C_W + 1$
7.	ϵ	none	none

We now describe the restricted 3-counter machine W that will simulate the network (M,N). The finite state control of W will contain the following entities:

V: contains the current state of M. Its initial value is v_0, where v_0 is the initial node of M.

W: contains the current state of N. Its initial value is w_0, where w_0 is the initial node of N.

BUFF: a buffer, of size one, used to contain a message in G, or the value E (empty). BUFF contains the contents of C_N (if any) which will, in the simulation, be no more than a single message. The initial value of BUFF is E.

P_flag: 0/1 valued; P_flag is zero iff both C_I and C_R were zero since the last E_o move on N was simulated. Initially, P_flag equals 0.

SBUF: a bounded counter whose value may range from 0 to $|M|^2 * |N|^2$, which is used to indicate the number of consecutive E_o moves simulated on N, since the last send move was simulated on M.

d_1: a variable in which to store a value between 0 and |M|.

d_2: a variable in which to store a value between 0 and |M|.

A state of W, contains a value for all the entities described above. However, in most cases, we are only interested in the values of V, W, and P_flag. Therefore, in most cases, we will simply use $(v,w)_{P\_flag}$ to indicate a state of W instead of listing the value of each entity.

The operation of W can be expressed as the following algorithm. (Although we never explicitly describe the labelled graph describing W, it is implicit in what follows, and is used in many of the subsequent proofs.) Let $l=(|M|^2+|N|^2)*(|G|+5)$. Notice that the total number of possible moves that can be made from any network state is no more than l. Hence, the possible moves of the network can, for any network state, be indexed by 1,...,l. In other words, we let $\Sigma=\{1,...,l\}$.

A. The code for W

```
initialize variables;
halt:=0;
while halt=0 do
   begin
      read an input symbol a in Σ;
      simulate the move indicated by a;
      phase check;
   end
```

B. Simulation of network moves

a) *receive on M*: v$-$r$\rightarrow$v'

```
if C_R≠0 ∧ BUFF=E
then
   C_R:=C_R-1;
   V := v';
else halt:=1;
endif
```

b) *send(g) on M*: v$-$s(g)$\rightarrow$v'

```
if BUFF≠E
then halt:=1;
else SBUF:=|M|^2*|N|^2; BUFF:=g; V:=v';
endif
```

c) *receive(g) on N*: w$-$r(g)$\rightarrow$w'

```
if BUFF≠g
then halt:=1;
else BUFF:=E; W:=w';
endif
```

d) *send on N*: w$-$s$\rightarrow$w'

```
if BUFF=E
then C_W:=C_W+1; W:=w';
else halt:=1;
endif
```

e) *ϵ-move on either machine*: v$-\epsilon\rightarrow$v' (w$-\epsilon\rightarrow$w')

```
if BUFF=E
then V:=v'; (W:=w';)
else halt:=1;
endif
```

f) *E_i move on M*: v$-E_i\rightarrow$v'

```
if  C_R=0 ∧ BUFF=E
then V:=v';
else halt:=1;
endif
```

g) *E_i move on N*: w$-E_i\rightarrow$w'

```
if P_flag=0∧C_W=0∧C_I=0∧BUFF=E
then W:=w';
else halt:=1;
endif
```

h) *E_o move on M*: v$-E_o\rightarrow$v'

```
if P_flag=0∧C_W=0∧C_I=0∧BUFF=E
then V:=v';
else halt:=1;
endif
```

i) *E_o move on N*: w$-E_o\rightarrow$w'

```
if SBUF≠0 ∧ C_W=0 ∧ BUFF=E
```

```
then
   P_flag:=1,
   W:=w';
else halt:=1;
endif
```

C. **Subroutine Consume(V,d_1,d_2)**

\each of the conditions to be checked in Consume are to be evaluated nondeterministically\

```
Subroutine Consume(V,d1,d2)
   if
     There is a path to a reachable loop,
     from v in M, with both receiving and
     sending (but no Eo) edges where at
     least d1 receiving edges occur before
     the first Ei move.
   then return "0";
   if
     M can consume d1+d2 messages by
     executing some path (but no loop)
     from v to some v' on M where at
     least d1 receiving edges occur before
     the first Ei move.
   then return "1";
   if
     There is a path to a reachable loop,
     from v in M, with receiving (but no
     sending or Eo) edges where at least d1
     receiving edges occur before the first
     Ei move.
   then return "2";
   else return "3";
   endif
```

(Note that in what follows, we sometimes use "yes" to indicate that the subroutine returns "0", "1" or "2"; "no" to represent "3")

D. **Phase Check**

\to handle the existence of a send loop on M\

```
   determine if a send loop, a loop with no
   receiving or Eo edges, has been executed
   on M. (This condition can always be
   checked nondeterministically by checking
   to see if the current value of V has been
   repeated without the intermediate
   simulation of either a receive or an Eo
   move on M.)
if so
then
   loop
   CW:=CW+1;
   forever
endif
```

\C_I should always be empty when P_flag=0\

```
if P_flag=0 ∧ CI≠0
then halt:=1;
endif
```

\to detect the end of a phase change\

```
if P_flag≠0
then
   if CR=0 ∧ CI=0
   then P_flag:=0;
   endif
endif
```

\to detect M's ability of consuming those messages in C_R and C_I\

```
if P_flag≠0
then
      if d1<min{|M|,CR}
      then halt:=1;
      else d1:=min{|M|,CR};
      endif

      if d1<|M|
      then d2:=min{|M|-d1,CI};
```

```
        else d2:=0;
        endif

        t:=consume(V,d1,d2);

        if input="Eo on N" ∧ t>0
        then SBUF:=SBUF-1;
        endif

        if t=3 then halt:=1; endif
    endif
```

Thus far, little has been mentioned concerning the operation of the daemon, other than that its moves occur asynchronously from those of W. In the construction above and the subsequent proofs, however, we implicitly assume that the daemon does not move any tokens during the period in which W is executing a pass of its *while* loop; thus the daemon is assumed to only be active before and after W executes an iteration of its loop. This assumption is not necessary, but it simplifies our constructions and many of the ensuing arguments.

For the sake of brevity, W has been described via an algorithm. However, many of the ensuing proofs utilize certain facts regarding the labelled state graph of W. We now list these facts.

1. C_I is always zero, in any configuration of W, where P_flag=0, that occurs during a pass of the loop, unless halt=1.

2. C_{Ij} and C_{Rj} moves, where $0<j<|M|$, can only be made by W when Pflag=1. Furthermore, a C_{Ij} move is made only when the sum of tokens in C_R and C_I is less than $|M|$.

3. If a C_{Rj} move is made by W for some $0\leq j<|M|$, then no C_{Rl} move, for $l>j$, can be made by W unless in an intermediate configuration P_flag was zero. Thus, the value of C_R does not grow while P_flag=1 once it becomes less than $|M|$. (Actually, it is possible that C_R may grow by a single token just previous to a loop iteration that is to simulate a receive move on M. Note, however, that no subsequent C_{Rj} move is made until after the additional token has been removed from C_R.)

The truth of these facts should be reasonably easy to see from a straightforward encoding of the algorithm.

The simulation of the network (M,N) by W can be viewed as oscillating between phases; those where P_flag is zero which correspond to portions of the network's computation where C_N is small (or potentially empty), and those where P_flag is one which correspond to portions of the network's computation where C_M is small. Any portion of the network's computation where C_N is empty can be simulated directly, as long as P_flag is zero; since then E_i moves on N and E_o moves on M can be simulated whenever both C_I and C_W are zero. (Recall that C_I is always zero when P_flag is zero.) Otherwise, W always runs (or simulates) N fast (whenever C_N is to contain more than a single message) in order to keep the size of BUFF (C_N) small. So that W can

simulate portions of the network's computation where C_M is empty, E_i moves on M are allowed to be simulated whenever C_R is zero and E_o moves on N are allowed to be simulated whenever C_W is zero. (Certain additional checks are made by W, however, to insure that N does not run too fast and proceed beyond a point where M would have blocked its movement.) Thus, the portions of the network's computation, where C_M is empty, are simulated by running the moves of N early and storing the output temporarily in C_W or C_I. After the simulation of an E_o move on N, P_flag is set to one, indicating that both C_R and C_I must later become zero before an E_o move on M or E_i move on N can next be simulated. Furthermore, the feasibility of this happening is rechecked (by Phase Check and Consume) after each subsequent simulated move, until P_flag is reset to zero. This insures that for each E_o move on N simulated, that M will later (on W) receive enough messages, before its next E_o move, for C_M to have actually been empty. Notice, as W oscillates between phases: That while P_flag is zero C_I is also zero; and then while P_flag is one, first C_R goes to zero and then C_I goes to zero ultimately causing P_flag to be reset to zero. The reader should recall this cyclic behavior of W as it will be analyzed in detail in the subsequent discussion.

In what follows, we show that the network (M,N) is unbounded iff the corresponding machine W (as described above) is unbounded. To prove this, we require the following lemmas :

Lemma 3.1: The state (v,w,x,E) is reachable in (M,N) iff the configuration $((v,w)_0,x,0,0)$ is reachable in W.

Proof :

If Part : Let $c=((v,w)_0,x,0,0)$ be a reachable configuration of W. Suppose c is reachable via the computation path P. We prove that the state (v,w,x,E) is reachable in (M,N) by induction on the number of E_o moves on M or E_i moves on N simulated in P.

Induction Base : If there is no such move in P then M could, in the network, run fast enough, compared to N, so that C_M would always be empty unless M's next move were to be a receive. This strategy allows the E_i moves on M and E_o moves on N to always be executable. Hence, by executing the same sequence of moves on M and N, but interleaved differently, the network could reach (v,w,x,E).

Induction Hypothesis : Assume that the assertion is true for n such moves.

Induction Step : Consider the case in which there are n+1 of these moves. Let $c_1=((v_1,w_1)_0,x_1,0,0)$, $c_2=((v_2,w_2)_0,x_2,0,0)$ be the configurations just after the n-th and (n+1)st such moves in P. Clearly, no E_o move on M or E_i move on N is simulated from c_1 to c_2 unless c_1 and c_2 are adjacent on P. By the induction hypothesis, the state (v_1,w_1,x_1,E) is reachable. Now, consider the path from c_1 to c_2. Since there is no E_o move on M or E_i move on N, M can always run fast, so that the empty channel moves can be progressed without being blocked. Hence, the state (v_2,w_2,x_2,E) is reachable. Moreover, since there is no E_o move on M or E_i move on N simulated from c_2 to c, the state (v,w,x,E) is then reachable.

Only If Part : For a path P that leads to the network state c=(v,w,x,E), we prove that the configuration $((v,w)_0,x,0,0)$ is reachable by doing induction on the number of intermediate states in which N's input channel is empty.

Induction Base : For the initial state (v_0,w_0,E,E), it is obvious that the configuration $((v_0,w_0)_0,0,0,0)$ is reachable.

Induction Hypothesis : Assume that the assertion is true for any reachable state of the form (v,w,x,E), in which, only n intermediate states have C_N empty.

Induction Step : Consider a reachable state c=(v,w,x,E), via path P, such that there are n previous states with C_N empty. Let $c_1=(v_1,w_1,x_1,E)$ be the last state before c such that C_N is empty. According to the induction hypothesis, the configuration $((v_1,w_1)_0,x_1,0,0)$ is reachable. During the period from c_1 to c, no E_o move on M or E_i move on N is executed (unless c_1 and c are adjacent on P). Therefore, in the simulation from c_1 to c, the messages output by N can be stored in C_I, so that E_i moves on M will not be blocked. The only case in which this might fail is when the simulation gets blocked by a zero SBUF. Without loss of generality, we assume that no state repeats in P, and each time the subroutine Consume is called, it will return "yes" as long as M can indeed consume those tokens in C_I and C_R. Moreover, we assume the subroutine will return "0" if there exists some reachable loop on M with both sending and receiving edges. In other words, the subroutine returns "1" or "2" only when no such loop exists. Now, let $d_i=(v_i,w_i,E,y_i)$ (i=1,...,k) be the configurations just after the E_o moves on N between c_1 and c. Since the original computation from c_1 to c can be proceeded, it must be the case that the subroutine will always return "yes" for those d_i's. Based on this fact, the simulation can be blocked only if during some period, d_s to $d_{s+|M|^2*|N|^2}$, the subroutine will always return "1" or "2", and therefore, SBUF becomes 0. Let $((v_s,w_s')_f,x_s,y_s,0)$ be the configuration after simulating the E_o move at d_s. Consider the path on M from w_s' to w; M can send at most |M| segments of messages. Now, consider the period from d_s to $d_{s+|M|^2*|N|^2}$; M can send at most |M| segments of messages and N will not receive any message. Without loss of generality, we also assume that the same E_o move on N is not executed twice without an intermediate send or receive move. Thus for every |N| E_o moves there must be at least one intermediate send move on N executed; and also a read move on M executed. Between at most |M| of these, M can send a segment or sequence of messages. Thus, there must exist a sequence of at least |M|*|N| of the d_i's for which no intervening sends on M occur. Therefore, there must be some p and q, such that $d_p=d_q$, which implies that states repeat in P, which, of course, contradicts our assumption. Hence, during the simulation, SBUF will not become zero. It is then clear that the configuration $((v,w)_0,x,0,0)$ is reachable. □

Lemma 3.2: If the configuration $c=((v,w)_1,x,y,z)$ is reachable in W, then one of the following must be true: Let t=x+y+z.

1. There is a reachable send loop on N.

2. There exists a reachable state (v',w',x',y') in (M,N), such that $y'>(t-|M|^{4*}|N|^3-|M|)/|M|$.

Proof : Let P: $c_o \rightarrow c$ be a computation where Consume returns zero whenever there is a reachable loop on M containing both receiving and sending edges. In other words, the subroutine returns "1", "2" or "3" only if no such loop exists. Let $c_1=((v_1,w_1)_o, x_1,0,0)$ be the configuration just after the last E_o move on M (or E_i move on N) in P. Let $c_2=((v_2, w_2)_1, x_2, y_2, z_2)$, after c_1, be the configuration just after the last time that Consume returns "0". Now, consider the following cases:

(Case 1:) $c_2 \neq c$ and $(x_2+y_2+z_2) \leq t-|M|^{4*}|N|^3$.
Then from c_2 to c, N must add at least $|M|^{4*}|N|^3$ tokens to C_W. Since Consume cannot yield a zero response between c_2 and c, we have that no more than |M| sends on M (and hence |M| reads on N) can occur during this period unless a send loop was executed on M. In the latter case (2) is true, otherwise no more than $|M|^{2*}|N|^2$ E_o moves may occur between any two of these sends, since SBUF would be decremented for each occurrence. However, for one of these |M| periods at least $|M|^{3*}|N|^3$ tokens are added to C_W. Consequently, during this period a send loop must have been executed on N. So (1) is true.

(Case 2:) $c_2 \neq c$ and $(x_2+y_2+z_2)>t-|M|^{4*}|N|^3$.
Let v_1 and v_2 be the states of M in the configurations c_1 and c_2, respectively. Since Consume$(v_2,d_1,d_2)=0$ at c_2, there exists a loop *l* on M with both sending and receiving edges. Now, by executing machine M fast, we can reach the state v_2. Then, M can enter the loop *l* while consuming at most |M| messages. After that, M repeatedly executes the loop *l*. In this way a state (v',w',x',y') with $y'>(t-|M|^{4*}|N|^3-|M|)/|M|$, is reachable since *l* must send at least one message for every |M| messages received). So (3) is true.

(Case 3:) $c_2=c$.
Similar to Case 2. □

It follows directly from lemmas 3.1 and 3.2 that the network (M,N) is unbounded if the machine W is unbounded. The next lemma provides the converse.

Lemma 3.3: If the network (M,N) is unbounded, then the machine W is unbounded.

Proof :

(Case 1:) Suppose that the channel C_M is unbounded. Consider a computation path *l* such that C_M gets arbitrarily large along this path. Consider an arbitrary state c=(v,w,x,y) on the path *l*, such that $x>|M|^*|N|$. Let $c_1=(v_1,w_1,x_1,E)$ be the last configuration before c in which C_N is empty. Then according to lemma 3.1, $((v_1,w_1)_0,x_1,0,0)$ is reachable in W. Now, consider the computation P: $c_1 \rightarrow c$. Since there is no E_o move on M (or E_i move on N) in P, W can, basically, run

N fast by using C_I to store those messages output by N (and hence get at least x tokens in the counters). The only case in which this might fail is when this simulation gets blocked by a zero SBUF. By the same argument as was used in lemma 3.1, SBUF cannot become zero, and hence, a configuration $((v",w")_f,x",y",z")$ with $x"+y"+z" \geq x$ is reachable.

(Case 2:) Suppose that the channel C_M is bounded by k.

Consider a computation path l of the network, such that C_N gets arbitrarily large in which no state repeats. Then, there can be only a finite number of E_o moves on M or E_i moves on N in l (otherwise, a state would repeat). Let $c_1=(v_1,w_1,x_1,E)$ be the configuration just after the last of these moves. Furthermore, we assume that there is no send loop on M in l; otherwise, this send loop would be detected during phase check and, it would follow immediately that W is unbounded. Therefore, M must have an infinite number of receiving moves in l. Consequently, N must have an infinite number of sending moves in l. After c_1, by using C_I to store those messages output by N, it is then clear that W is unbounded. □

From lemmas 3.1-3.3, we obtain:

Theorem 3.1: The network (M,N) is unbounded iff the corresponding machine W is unbounded.

From the following series of lemmas, we are able to ascertain a simple method by which we can determine whether the simulating automaton W is bounded. The general idea here is to show that the number of tokens stored by W can only grow by a fixed amount unless either W executes a simple "pumpable" loop l (with $C(l)>0$), of the type considered in Lemma 3.5 or W continually oscillates between states where P_flag=0 and states where P_flag=1. Lemmas 3.6-3.8 analyze this cyclic or oscillating behavior. Lemmas 3.9-3.10 show that certain "complex" loops, where the value of P_flag oscillates (and the characteristic value is greater than zero) are pumpable, while lemmas 3.11-3.12 show that if the number of tokens stored by W grows by so much then such a complex pumpable loop must have been executed, if a simple pumpable loop was not.

Lemma 3.4: Let c=(q,x,y,z) and c'=(q',x',y',z') be reachable configurations of W such that l: $c-w\rightarrow c'$ (for some input w), where

1. $x'+y' \geq |M|$ and Consume(v',x',y')="yes" (v' is the state of M in c'),
2. no E_o move on M in l,

then the C_{Ij} $(0<j<|M|)$ moves in l can be replaced by C'_{I0}, without affecting the computation of l.

Proof : Let l' be the new computation in which all C_{Ij} moves are replaced by C'_{I0}. The only possible difference would be having l' fail at some intermediate configuration c"=(q",x",y",z"). In other words, Consume(v",x",y")="yes" but Consume(v",x",|M|)="no". However, since (1) is true, there must exist a loop (with receiving edges), on M, reachable from c'. Moreover, (2) implies that the loop must be also reachable from c" without traversing an E_o move. Therefore, it

must be the case that Consume(v'',x'',|M|)="yes". As a consequence, l' will lead to the same configuration c'. □

Lemma 3.5: Let c=(q,x,y,z) and c'=(q,x',y',z') be reachable configurations of W such that x'+y'+z'>x+y+z , and where l: c$\rightarrow$w$\rightarrow$c' is a computation (for some input w) that satisfies at least one of the following conditions:

1. There is no C_{W0} move in l.
2. There is no C_{I0} move in l.

Then W is unbounded.

Proof : The proof is by cases.

(Case 1:) Suppose that there is no C_{W0} move in l. Let l: $c_0 \rightarrow c_1 \rightarrow ... \rightarrow c_n$ where c_0=c and c_n=c'. We construct a new computation l': $c'_0 \rightarrow c'_1 \rightarrow ... \rightarrow c'_n$ such that c'_0=c and c'_n=(q,x,y,z+h) where h=(x'+y'+z')-(x+y+z). Clearly, such a computation could be "pumped" and hence W would be unbounded. The computation l' will execute exactly the same sequence of moves (of W) that are executed in l. The only difference will be in the action of the daemon. After each step, of the computation l', the daemon moves the fewest number of tokens possible in order to allow the next step of the computation of W to proceed. Clearly, then if c'_{n-1}=(q'',x'',y'',z'') then x''$\leq$z and y''$\leq$y. At this point the daemon moves just enough tokens to allow the last move to result in c'_n.

(Case 2:) Suppose there is no C_{I0} move in l. If there are no C_{Ij} moves (or no C_{W0} moves) in l, the proof is similar to that of case 1. We assume therefore, without loss of generality, that l contains both C_{Ij} and C_{W0} moves. Since a C_{Ij} move is made only when the value of C_R+C_I is less than |M|, C_R can not increase in l. This is true since P_flag must always be 1; otherwise a C_{I0} move will be made. The value of C_R also can not decrease (since it ends up in the same state, d_1 must be the same), hence the value of C_R is identical in all c_i in l. Since P_flag always is 1 over l, any C_{W0} move must be the result of simulating an E_o move on N. Consider the sequence of moves in l of M.

case 1: no send moves.

Then from c to c', there is no receiving move on N. Moreover, since the states repeat in c and c', and SBUF will not be reset during the path from c to c' (because there is no send move on M), it must be true that the subroutine will always return "0"; otherwise, SBUF would have been altered. Therefore, the computation on N, from c to c', can be executed an infinite number of times, hence, W is unbounded.

case 2: only send moves.

This send loop will be detected during the phase check, hence W is unbounded.

case 3: E_i and send moves.

Note that after the first E_i move, C_R becomes zero. Furthermore, since P_flag equals 1 throughout *l*, the daemon cannot move any token to C_R. Therefore, W is unbounded.

case 4: receive and send moves (or E_i, receive and send moves).

The C_{Ij} or C'_{I0} move makes no difference now, since the "subroutine" can response "yes" for any number of tokens to be consumed (i.e. can replace all C_{Ij} moves in *l* by C'_{I0}). □

In what follows, let $L=2*|Q|^3$ and $U=2*|Q|^3+2*|Q|^2$, where $|Q|$ is the number of states in W.

Lemma 3.6: Let c=(q,x,0,0) (or (q,0,0,z)) and c'=(q',x',0,z') be two reachable configurations, such that P: c−w→c' is a computation. If for each c''=(q'',x'',y'',z'') in P, we have:

1. $L<|c''|<U$,
2. P_flag of c'' is zero,

then one of the following must be true:

A. W is unbounded.

B. if $x'\leq(L/2)$, then (q,x+h,0,0) (or (q,0,0,z+h))−w'→(q',x',0,z'+h) is a computation for some input w'.

C. if $x'>(L/2)$ then (q,x+h,0,0) (or (q,0,0,z+h))−w'→(q',x'+h,0,z') is a computation for some input w'.

Proof : (By induction on the number of C_{R0} moves in P)

Since P_flag equals zero throughout the entire path P, no E_o move on N will be simulated in P. So, we may assume that $C_I=0$ for all c'' in P.

Induction Base : Assume that there are no C_{R0} moves in P. We consider the case where c=(q,x,0,0) first. Let $c_1=(q_1,x_1,0,z_1)$ be configuration just before the last C_{W0} move in P. Clearly, the next configuration can, without loss of generality, be $c'_1=(q'_1,x_1+z_1,0,0)$ (since C_I is always empty) where, of course, $x_1+z_1>L$. Now, if we start with (q,x+h,0,0), since there is no C_{R0} move from c to c_1', we can certainly reach $(q'_1,x_1+z_1+h,0,0)$ by following the same computation path . Then, if $x'>(L/2)$, the configuration (q',x'+h,0,z') can be reached by executing the same computation from c_1' to c'. Therefore, (C) is true. On the other hand, if $x'\leq(L/2)$, then it must be the case that, from c_1' to c', at least L/2 of the tokens in C_R were consumed. During the same period, some tokens were added to C_W. Therefore, there must exist a loop, with

no C_{W0} or C_{R0} moves, whose traversal caused a transfer of tokens from C_R to C_W. One of the following three cases must therefore occur:

1. There exists such a loop l_i such that $c(l_i)>0$.
2. There exists such a loop l_j such that $c(l_j)=0$.
3. All such loops have characteristic value less than zero.

Case 1: Lemma 3.5 implies that W is unbounded.

Case 2: Then the loop l_j can be iterated in order to transfer the h additional tokens from C_R to C_W. As a consequence, (q',x',0,z'+h) is reachable.

Case 3: Then to consume those tokens in C_R, at least $2*|Q|^2$ loops should be executed (each such loop can have at most $|Q|/2$ w moves). However, each loop will cause the net loss of at least one token. Consequently, there must be a net loss of at least $2*|Q|^2$ tokens in the computation from c'_1 to c'. This, of course, contradicts assumption 1.

Now, consider the case where c=(q,0,0,z). If there are no C_{W0} moves in P, (q',x',0,z'+h) can be reached from (q,0,0,z+h), by following exactly the same path. Otherwise, let $c_2=(q_2,x_2,0,0)$ be the configuration just after the first C_{W0} move in P. By having the daemon move those h additional tokens to C_R just before the C_{W0} move, $(q_2,x_2+h,0,0)$ is reachable. Hence, it follows directly from the previous argument that either A, B, or C must be true.

Induction Hypothesis : Assume that the assertion is true for K C_{R0} moves.

Induction Step : If there exist K+1 C_{R0} moves in P, we want to show that the assertion is still true. We consider the case where c=(q,x,0,0), first. We divide the computation $c \rightarrow c'$ into two subcomputations, $c \rightarrow c_1$ and $c_1 \rightarrow c'$, where $c_1=(q_1,0,0,z_1)$ is the configuration just after the first C_{R0} move in P. Therefore, in $c_1 \rightarrow c'$, there are at most K C_{R0} moves. According to the base case, if we start with (q,x+h,0,0), $(q_1,0,0\ z_1+h)$ is reachable. Then by hypothesis, one of the three cases A, B, or C is true. So, the assertion is true for K+1 C_{R0} moves.

The same argument can be used to deal with the case where c=(q,0,0,z). □

Lemma 3.7: Let c=(q,x,y,0) and c'=(q',0,y',z') be reachable configurations, such that P: $c \rightarrow w \rightarrow c'$ is a computation, and if for each c'' =(q'',x'',y'',z'') in P, $L<|c''|<U$, P_flag=1, and $x''>0$ (unless c''=c'), then one of the following is true:

A. W is unbounded.

B. if $x \leq (L/2)$, then either $y'>(L/2)$ and $(q,x,y+h,0) \rightarrow w' \rightarrow (q',0,y'+h,z')$ or $(q,x,y+h,0) \rightarrow w' \rightarrow (q',0,y',z'+h)$.

C. if $x>(L/2)$, then either $y'>(L/2)$ and $(q,x+h,y,0) \rightarrow w' \rightarrow (q',0,y'+h,z')$ or $(q,x+h,y,0) \rightarrow w' \rightarrow (q',0,y',z'+h)$.

Proof : Let $c_1=(q_1,x_1,y_1,0)$ be the configuration after the last C_{W0} move. Let c_2 be the last time before c' that $C_R+C_I>|M|$. According to lemma 3.4, all C_{Ij} moves from c to c_2 can be replaced by C'_{I0}. Let c_3 be the configuration just before the first C_{Rj} move.

If $x\leq(L/2)$, it follows from the assumptions that $y>(L/2)$. If c_3 occurs in between c and c_1, then $y_1>L/2$ (since C_R cannot increase after a C_{Rj} move). So, by following exactly the same computation from c to c', we have $(q,x,y+h,0)-w\rightarrow(q',0,y'+h,z')$, and $y'>L/2$. If c_3 occurs after c_1, then we have the following two cases:

case 1: $y_1 > L/2$.
Using the same argument as above, (q',0,y'+h,z') with $y'>L/2$ is reachable.

case 2: $y_1 \leq L/2$.
Then we must have $x_1>L/2$. By allowing the daemon to move h tokens from C_I to C_R, we can reach $(q_1,x_1+h,y_1,0)$ with $x_1>L/2$. Now, consider the path from c_1 to c_3. M must have traversed some loop, without C_{Rj} or C_{W0} moves, in order to transfer tokens from C_R to C_W, whose characteristic value is zero; otherwise, either W is unbounded or we have a contradiction (same reasoning as in the proof of lemma 3.6). By using that loop, we are able to reach the configuration (q',0,y',z'+h).

Now, consider the case where $x>(L/2)$. We want to show that if we start with x+h tokens in C_R, we can move those h tokens to either C_I or C_W. Consider the following two cases:

case 1: $x_1>(L/2)$.

In this case, there must exist a loop l between c_1 and c_3 such that $C(l)=0$, and there is no C_{W0} move in l. We then can use that loop to transfer those h additional tokens from C_R to C_W, hence, (q',0,y',z'+h) is reachable by following the same path from c_3 to c'.

case 2: $x_1\leq(L/2)$.

In this case, if the loop has no C_{W0} move, we have the same result as in case 1. Otherwise, since $x_1\leq(L/2)$, it must be the case that $y_1>(L/2)$. By using the loop (possibly with C_{W0} moves, but no C_{Ij} moves), clearly those h additional tokens can be moved to C_I. Then by following the same path from c_3 to c', the configuration (q',0,y'+h,z') ($y'>(L/2)$) is reachable. □

Lemma 3.8: Let c=(q,0,y,z) and c'=(q',0,0,z') be two reachable configurations, such that P: $c-w\rightarrow c'$ is a computation. For each c''=(q'',x'',y'',z'') in P, if c'' satisfies the following three conditions:

1. $L<|c''|<U$,

2. x''=0,

3. P_flag of c'' is always one (except at c'),

then one of the following must be true:

A. W is unbounded.

B. if y>(L/2), then (q,0,y+h,z)$-w'\rightarrow$(q',0,0,z'+h) is a computation.

C. if y$\leq$(L/2), then (q,0,y,z+h)$-w'\rightarrow$(q',0,0,z'+h) is a computation.

Proof :

(Case 1:) y>(L/2).

Let c_1 be the last configuration of P in which C_I=L/2. Let c_2 be the last configuration of P in which C_I>|M|. Clearly, between c and c_2, all C_{Ij} moves can be replaced by C'_{I0}. Between c_1 and c_2, there are no C_{W0} moves; otherwise, C_I would then be greater than L/2. Moreover, there must exist a loop *l*, between c_1 and c_2, with C(*l*)=0 which traverses no C_{W0} edges. The rest is similar to the proof of the lemma 3.7.

(Case 2:) y$\leq$(L/2).

If there is no C_{W0} move in P, then (q',0,0,z'+h) can certainly be reached from (q,0,y,z+h). Assume that there are some C_{W0} moves in P. After the last C_{W0} move, the configuration can be (q'',0,y''+h,0) and y''>(L/2). By using the same argument as in case 1, (C) is then true. □

Lemma 3.9: Let c=(q,x,0,0) ((q,0,0,z)) and c'=(q',x'+h,0,0) ((q',0,0,z'+h)) be two reachable configurations, such that P: c$-w\rightarrow$c' is a computation. If for each c'' in P, L<|c''|<U, and P_flag=0 for states q and q', then (q,x+h,0,0) ((q,0,0,z+h))$-w'\rightarrow$(q',x'+2*h,0,0) ((q',0,0,z'+2*h)) is a computation.

Proof : To prove this, we divide the computation from c to c' into phases. Each phase starts and ends with a change of the P_flag. Hence, there are two kinds of phases, namely, 0-1 phases and 1-0 phases. A 0-1 phase is the period which begins when P_flag changes from 1 to 0, and ends when it next changes from 0 to 1. The 1-0 phase is defined similarly. In the following, we want to show that the additional h tokens can be moved across each phase.

We consider the case where c=(q,0,0,z) and c'=(q',0,0,z'+h) first. The proof is by induction on the number of phase changes. A phase change is the combination of two consecutive phases, 0-1 and 1-0.

Induction Base : no phase change.

It is given that c and c' have P_flag=0. Furthermore, since there is no phase change C_I is

always zero. Then by an induction on the number of C_{W0} moves (and also by induction on the number of C_{R0} moves in between two consecutive C_{W0} moves) in P, we can show that $(q,0,0,z+h)-w'\rightarrow(q',0,0,z'+2*h)$ is a computation. So, the base case is true.

Induction Hypothesis : Assume that the assertion is true for K phase changes.

Induction Step : Consider the case that there are K+1 phase changes in P. Let $c_1=(q_1,x_1,0,z_1)$ and $c'_1=(q_1,x_1,z_1,0)$ be the final two configurations at the end of the first 0-1 phase . Let $c_2=(q_2,0,0,z_2)$ be the one that appeared after the first phase change. Let $c_t=(q_t,0,y_t,z_t)$ be the first configuration (after c'_1) in which C_R is zero. Now, if we start with (q,0,0,z+h), then according to lemma 3.6, we are able to reach

1. $(q_1,x_1,z_1+h,0)$, if $x_1\leq(L/2)$
2. $(q_1,x_1+h,z_1,0)$, if $x_1>(L/2)$

Then, from lemma 3.7, we can reach

1. $(q_t,0,y_t+h,z_t)$, if $y_t>(L/2)$
2. $(q_t,0,y_t,z_t+h)$, if $y_t\leq(L/2)$.

Now, following the result of lemma 3.8, we can therefore reach $(q_2,0,0,z_2+h)$. By using induction hypothesis, it follows immediately that (q',0,0,z'+2*h) is reachable. □

Lemma 3.10: If there is a reachable computation P as described in lemma 3.9, and c'=(q,x+h,0,0) ((q,0,0,z+h)) (i.e. c and c' have the same state), then W is unbounded.

Proof : For any integer k>0, let d=⌈k/h⌉. From lemma 3.9, by executing the computation from c to c' d times, we are able to reach the configuration c'=(q',x+h*(⌈k/h⌉),0,0) (or (q',0,0,z+h*(⌈k/h⌉))), and clearly, |c'|>k. Therefore, W is unbounded. □

Lemma 3.11: Let *l*: $c-w\rightarrow c'$ be a reachable computation, where c=(q,x,y,z) and c'=(q,x',y',z'). and satisfies the following conditions:

1. x'+y'+z'>x+y+z.
2. for each c'' in *l*, |c''|>2*|M|

Then one of the following three things must be true:

1. W is unbounded,
2. there exists a configuration (q'',x'',0,0) in *l*,
3. there exists a configuration (q'',0,0,z'') in *l*.

Proof : There are three cases for *l*:

(Case 1) P_flag =0 throughout l.

Clearly, no E_o move on N can occur in l. If there is a C_{W0} move in l, it must be for simulating an E_o move on M (or an E_i move on N). Consequently, a configuration of the form (q'',x'',0,0) is in l. On the other hand, if there is no C_{W0} move in l, it follows directly from lemma 3.5, that W is unbounded.

(Case 2) P_flag changes in l.

It is clear that a configuration of the form (q'',0,0,z'') is in l.

(Case 3) P_flag = 1 throughout l.

If $C_R > |M|$ throughout l, then no C_{Ij} move need be in l (Lemma 3.4). Therefore, lemma 3.5 implies that W is unbounded. If a C_{Rj} move occurs in l, then it must be the case that C_R=j throughout l (since the state repeats in c and c', and C_R is nonincreasing). Moreover, C_{Ij} and C_{W0} moves must occur in l; otherwise, according to lemma 3.5, W is unbounded. After a C_{W0} is executed, C_I will be greater than |M|. Therefore, all C_{Ij} moves in l can be replaced by C'_{I0}. Hence, lemma 3.4 implies that W is unbounded. □

Lemma 3.12: The machine W is unbounded iff a configuration c, with $|c| > U$, is reachable.

Proof : Clearly we need only concern ourselves with the if part. Without loss of generality, assume that b is the first configuration such that $|b|=2^*|Q|^3+2^*|Q|^2$. Consider the computation from c_0 to b, where c_0 is the initial configuration. Let a be the last configuration before b such that, $|a|=2^*|Q|^3$. Clearly, each configuration between a and b contains between $2^*|Q|^3$ and $2^*|Q|^3+2^*|Q|^2$ tokens.

Define intervals (a_i, b_i), $0 \le i \le 2^*|Q|-1$, where a_i= the last time before b that $|a_i|=2^*|Q|^3+i^*|Q|$, b_i= the first time after a_i that $|b_i|=2^*|Q|^3+(i+1)^*|Q|$. Now between time a_i and b_i, we must have executed some loop, say l_i, in which there is a gain of the counters. This must be the case, since during the interval (a_i, b_i), the machine W gains |Q| tokens and at least |Q| steps were executed. See Figure 3.4.

Now consider the loops l_i $(0 \le i \le 2^*|Q|-1)$. According to lemma 3.11, there must exist a configuration of the form d_i=(q'',x'',0,0) or e_i=(q'',0,0,z'') in loop l_i, otherwise, W is unbounded immediately. For the configuration d_i (or e_i) $(0 \le i \le 2^*|Q|-1)$, there must be two configurations d_s and d_t (or e_s and e_t) such that they are in the same state of the machine W, and $|d_t| > |d_s|$ (or $|e_t| > |e_s|$). Therefore, according to lemma 3.10, W is unbounded. □

Theorem 3.2: The boundedness problem for W can be decided in nondeterministic logspace.

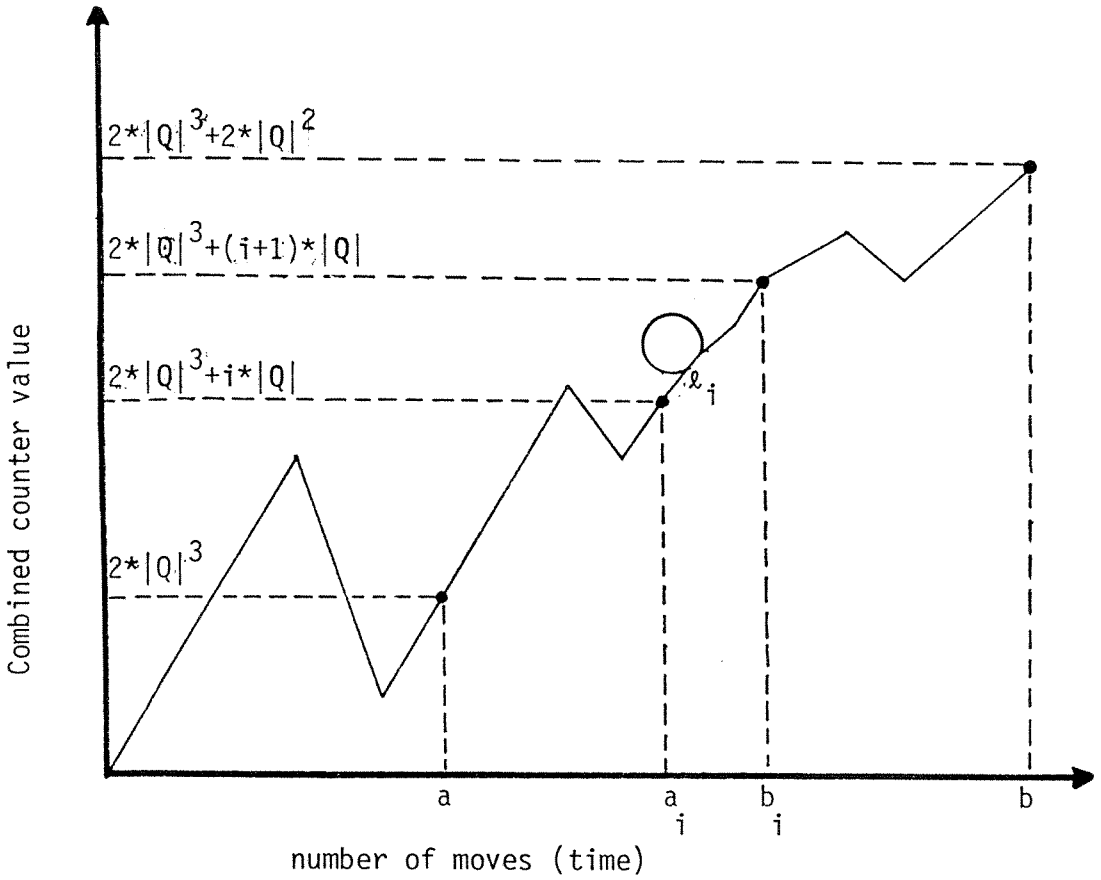

Figure 3.4 An unbounded computation of W.

Proof : This follows immediately from lemma 3.12. □

4. RELATED UNDECIDABILITY RESULTS

In this section, we examine the simplest classes of FIFO, Priority and Random empty-channel-detecting CFSM's (CFSM's whose moves may include, in addition to those already allowed, conditional moves in which *any* input and/or output channel is checked for emptiness) for which the boundedness problem remains undecidable. These show that the result in the previous section cannot be extended. The undecidability proofs given in this section are similar in technique and style to those of [7]; and hence in each case we only provide an outline of the actual proof.

Theorem 4.1: The boundedness problem is undecidable for networks of three empty-channel-detecting FIFO CFSM's.

Proof: For an arbitrary 2-counter machine T [10,14], we construct a network (M_1,M_2,M_3) that will be bounded iff the 2-counter machine halts. Since the halting problem for 2-counter machines is undecidable, the boundedness problem for this class of networks is undecidable.

Roughly speaking, the machine M_1 is used to simulate the finite control of T; the machine M_2 acts as an "echoer" that transmits the contents of its input channel to its output channel and the machine M_3, acts as a "synchronizer" that coordinates the actions between M_1 and M_2. At certain instances, the number of messages in the channel from M_2 to M_1 will equal $2^i * 3^j$, where i and j are the two integers currently stored in the counters of T. The simulation proceeds in phases. First M_1 will process the number of messages from M_2 sending the output back to M_2. After which, in the next phase, M_2 will send them back again so that M_1 can continue with its next phase. In each case the end of a phase takes place when the active machine's input channel becomes empty. The following phase does not begin, however, until the currently active machine signals the other with a message via M_3. The remaining details are left to the reader. □

This result holds even if each machine is restricted to send a single type of message, and only one of the machines is allowed type 3-4 moves. Only type 3 (4) moves are needed, however, if two of the machines are allowed conditional moves. The reader should recall the definition of type 1-4 moves from section 1. (Recall also that the boundedness problem is undecidable for networks of two FIFO CFSM's, which exchange two types of messages, even when no moves of type 2-4 are allowed[4].)

Our last two results concern Priority and Random CFSM's [7]. Recall that for these types of networks the channel contents, at any given time, are represented by a multiset of messages (over a finite number of distinct message types). Such a channel is considered to be empty iff the channel is devoid of each type of message. The last two theorems also indicate that the FIFO result of section 3 is somewhat of an anomaly in that no corresponding case arises for Priority and/or Random networks.

Theorem 4.2.: The boundedness problem is undecidable for networks of two empty-channel-detecting Priority CFSM's where one machine sends a single type of message and the other machine sends two types of messages. This result holds even if each machine is restricted to only moves of type 1-3 (or type 1-2 and 4).

Proof: Here we construct a Priority network (M,N) to simulate the 2-counter machine T. Let M be the "simulator" and N be the "echoer". We choose N to be the machine which can send two types of messages. In other words, M's input channel contains two types of messages -- a lower priority message (g_1) and a higher priority message (g_2). The number of g_1 messages is used to represent the value $2^i * 3^j$; the message g_2 is used to activate a new phase on M. In this simulation N determines the end of its phase when its input channel is empty and signals M that it may begin the next phase with a g_2 message. M determines the end of its phase in the same manner, but has no way to signal N that it may begin the next phase. Thus, N nondeterministically guesses the beginning of its next phase and sends a high priority message to M that signals this decision. M will then block if N has begun its next phase before M has finished the current one. Again, the details are left to the reader. □

Theorem 4.3.: The boundedness problem is undecidable for networks of two empty-channel-detecting Random CFSM's where one machine sends a single type of message and the other machine sends two types of messages. This result holds even if each machine is restricted to only moves of type 1-2 and 4.

Proof: Here we construct a Random network (M,N) to simulate the two counter machine T. Let M be the "simulator" and N be the "echoer". We choose M to be able to send two types of messages, "red" and "green", to N. Furthermore, each machine is allowed to execute only type 1-2 and 4 moves. Intuitively speaking, in order to avoid the mixture of messages sent back by N when M is still progressing, M will change the color of its sending messages from phase to phase. Figure 4.1 shows two consecutive phases of M and N. Notice that each phase of M starts with an E_o move, which indicates that the opposite machine N has finished its operations and therefore M can proceed. After finishing, M sends a message (either "red" or "green", depending on the current phase) to activate machine N (which must be waiting at node 8 or 4). Since the color of the messages sent from M change from phase to phase, any mismatch of the speed between the phases of M and N will block the entire simulation. Thus, for example, while N is at node 5 (or 6), M cannot traverse its edge from node 5 to node 6. This insures the proper synchronization between M and N. The rest of the details are left to the reader. □

We do not know, at this time, whether Theorem 4.3 holds when only moves of type 1-3 are allowed. The reader should note that, in each of the above theorems, a two counter machine can be simulated because the CFSM's were able to synchronize their actions before and after each phase of the simulation. This capability was precisely what was absent in the previous section. Lastly, we note that without the empty channel detection capability, each of the three aforementioned problems becomes decidable. See [4,7].

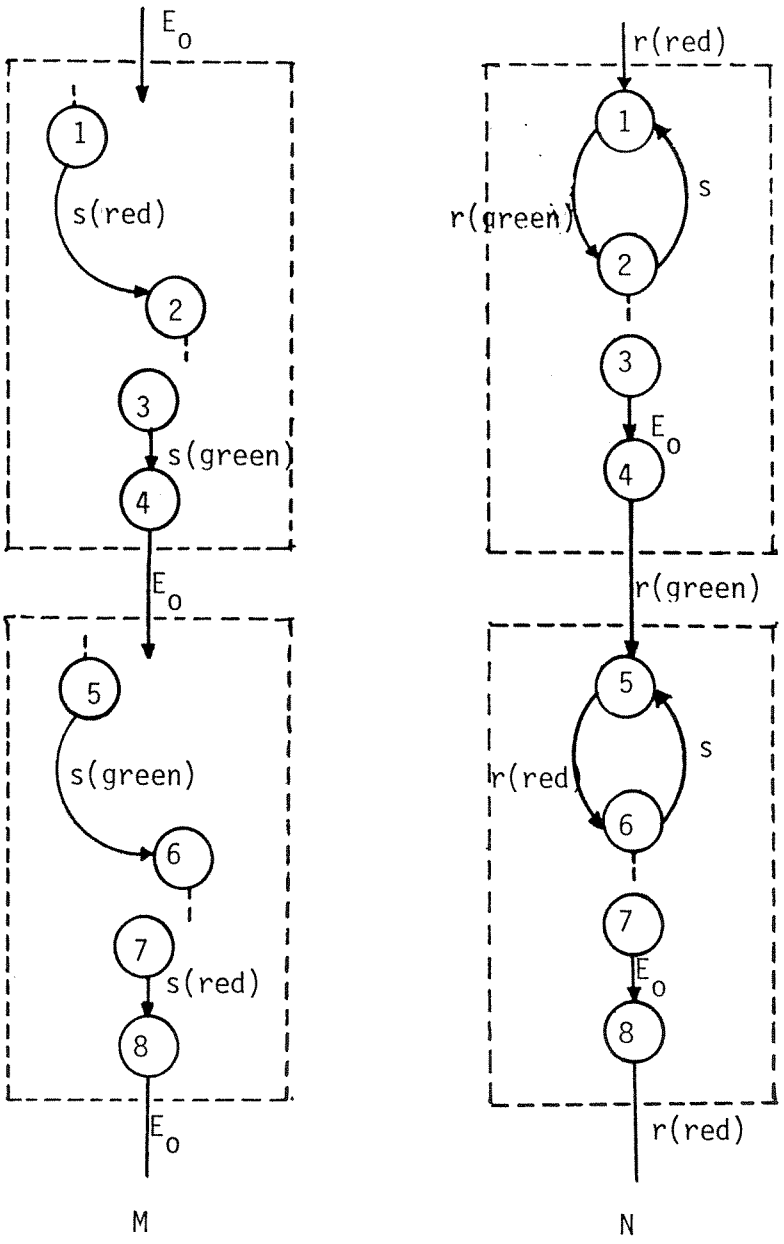

Figure 4.1 Two consecutive phases of M and N.

5. PRIORITY NETWORKS WHERE THE PRIORITY RELATION IS NOT FIXED

Recently, priority networks of CFSM's were introduced, where messages are received based on a fixed, partial-ordered priority relation[7]. (Unrelated messages can be received in any order.) This model is equivalent, in computational power, to certain classes of extended Petri nets, in particular those with priority tokens[8]. However, the CFSM model is more concise (since the channels and their contents are not modeled explicitly), and so is more convenient to use in modelling communication protocols and distributed systems. The results presented in [7] focus on the problem of whether the communication of a priority network is bounded. Specifically, it is shown that the problem of detecting boundedness is undecidable even if the machines exchange only two types of messages. Also considered is the case where one of the two channels is known to be bounded, and it is shown that three types of messages can make the problem undecidable in this case. (This problem is decidable in the case of FIFO networks[4].) The same problem becomes decidable if only two types of messages are allowed. Also considered is the case where one of the two machines sends one type of message. The problem is undecidable if the other machine sends three or more types of messages, and is decidable if the other machine sends two or less types of messages. (Both problems are decidable in the case of FIFO networks[4].) However, the latter result can be generalized to the case of three or more messages, if only two message types are mentioned in the priority relation.

In this section, we consider priority networks where the priority relation is not fixed. To be precise, we consider the case where a (possibly) different priority relation were assigned to each node in a CFSM. The results given in [7] extend in a straightforward manner to priority networks of this type, with two notable exceptions. Specifically, the results concerning priority relations that mention only two or three message types do not seem to generalize (at least the proofs given in [7] do not seem to generalize). In fact, some are no longer true. For example, consider the case of two machines where one sends a single type of message and the other sends at most three types of messages. For fixed priority relations that mention only two types of messages, the boundedness problem is decidable. However, if the priority relation is allowed to vary from node to node the problem becomes undecidable. This can be seen directly from the proof of Theorem 2 of [7], where the fixed priority constructed mentions three message types, but only two need be mentioned at any one node.

Lastly, we consider the case where only two message types are mentioned in the priority relation. In what follows, we use techniques similar to those of Section 3, to show that the boundedness problem is decidable for these cases of two machines where one sends a single type of message and the other sends at most two types of messages. Without loss of generality, we consider an arbitrary (M,N) where N sends a single type of message and M sends at most two types of messages. Let $\{g_1,g_2\}$ be the set of messages sent by M. For any state s=(v,w,x,y), let $\#_1(s)$ ($\#_2(s)$) denote the number of g_1 (g_2) messages in y. Now the priority relation at each node of N is either $\{g_1<g_2\}$, $\{g_2<g_1\}$ or ϕ. (Since M can receive only one type of message a priority relation need not be assigned to the nodes of M.)

To show that the boundedness problem is decidable, we require the following lemmas:

Lemma 5.1: For a network (M,N), the input channel of M is unbounded iff one of the following two conditions is satisfied:

A. There are two reachable states s=(v,w,x,y) and s'=(v,w,x',y') such that the following three conditions hold:

i. s' is reachable from s via a path P.

ii. For every state s" in P, $\#_1(s")>0$ and $\#_2(s)=\#_2(s')$ (or $\#_2(s")>0$ and $\#_1(s)=\#_1(s')$)

iii. $|x'| > |x|$ and $|y'| \geq |y|$.

B. There are two reachable states s=(v,w,x,y) and s'=(v,w,x',y') such that the following three conditions hold:

i. s' is reachable from s via a path P.

ii. $\#_1(s)=\#_1(s')$ and $\#_2(s)=\#_2(s')=0$ (or $\#_2(s)=\#_2(s')$ and $\#_1(s)=\#_1(s')=0$)

iii. $|x'| > |x|$

Proof:
(If Part:) Clearly, if either (A) or (B) holds, the computation P can be "pumped"; hence, M's input channel is unbounded.

(Only If Part:) Since M's input channel is unbounded, there must exist a path *l* such that, along *l*, the number of messages in M's input channel gets arbitrarily large. Consider any such computation *l* in (M,N); then the moves in *l* can be rearranged into a computation *l'* such that at any intermediate state s in *l'*, N will be executed next as long as $\#_1(s)>0$ and $\#_2(s)>0$. (Note, that the moves of *l'* on M may be a finite prefix of the moves of *l* on M if after some point on *l*, N stops reading. However, in either case the moves of N on *l* and *l'* are the same.) In other words, we can always proceed N fast as long as N's input channel contains both types of messages. Without loss of generality let *l* be such a path. Along this path let $E=\{s|s \text{ is in } l, \#_1(s)\leq 1 \text{ and } \#_2(s)=0\}$, $F=\{s|s \text{ is in } l, \#_2(s)\leq 1 \text{ and } \#_1(s)=0\}$. Then we have the following two cases:

Case 1. Both E and F are finite.
If this is the case, then after a time, every state s along *l* has $\#_1(s)>0$ (or $\#_2(s)>0$). Since M's input channel is unbounded, there is an infinite sequence of reachable distinct states $s_1=(v,x,x_1,y_1)$, $s_2=(v,w,x_2,y_2)$,... of (M,N) such that the following conditions hold:

i. For i=1,..., s_{i+1} is reachable from s_i.

ii. For i=1,..., $\#_2(s_i)>0$ (or $\#_1(s_i)>0$).

iii. There exist i and j (i<j), such that $\#_1(s_i)=\#_1(s_j)$ (or $\#_2(s_i)=\#_2(s_j)$), $|x_j|>|x_i|$ and $|y_j|\geq|y_i|$.

Hence, (A) is true.

Case 2. Either E or F is infinite.
Without loss of generality, assume that E is infinite. Then there exists an infinite sequence of

reachable distinct states $s_1=(v,w,x_1,y_1)$, $s_2=(v,w,x_2,y_2)$,... of (M,N) such that the following conditions hold:

i. For i=1,..., s_{i+1} is reachable from s_i.

ii. There exist i and j (i<j), such that $\#_1(s_i)=\#_1(s_j)$, $\#_2(s_i)=\#_2(s_j)=0$ and $|x_j|>|x_i|$.

Hence, (B) is true. □

Lemma 5.2: Assume that M's input channel is bounded. Then the network (M,N) is unbounded iff there exist two reachable states s=(v,w,x,y) and s'=(v,w,x,y') such that s' is reachable from s via some path *l*, and one of the following conditions is satisfied:

A. For every s" in *l*, either ($\#_2(s")>0$, $\#_1(s')=\#_1(s)$ and $\#_2(s')>\#_2(s)>0$) or ($\#_1(s")>0$, $\#_2(s')=\#_2(s)$ and $\#_1(s')>\#_1(s)>0$).

B. There is a reachable send loop on M.

Proof: Assume that M's input channel is bounded by K.

(If Part:) Clearly, the path *l* can be pumped, hence, N's input channel is unbounded.

(Only If Part:) Since N's input channel is unbounded, there exists a sequence of configurations $s_0,s_1,...,s_t$ where $s_t=(v,w,x,y)$ and $|y| > 2 * K * |M| * |N| + K * |M|$.

One can construct from this computation another computation $s'_0,...,s'_t$ by interleaving the identical sequence of moves made by M and N in such a way as to always execute the next move of N, as long as N's input channel contains both types of messages. Clearly, $s_t=s'_t$. Also, consider the state $s_i=(v',w',x',y')$ immediately after the last move of N. It must be the case that $|x'| \leq K$. Without loss of generality, we assume that $\#_1(s_i)\leq 1$. Now, consider the following two cases:

Case 1: $\#_2(s_i) \leq 2 * K * |M| * |N|$.
Then from s_i to s_t, M will send more than $K * |M|$ messages to N, and therefore M must have traversed a send loop; (B) is true.

Case 2: $\#_2(s_i) > 2 * K * |M| * |N|$.
Let s_j be the last state before s_i in which $\#_2(s_j)=0$. Then on the path from s_j to s_i, (A) must be true. □

From lemmas 5.1 and 5.2, we have the following theorem.

Theorem 5.1: Consider the class of priority networks where a different priority relation (that mentions at most two messages) is assigned to each node in a CFSM. Then the boundedness problem is decidable for such networks of two CFSM's where one sends a single type of message and the other sends at most two types of messages.

6. REFERENCES

[1] Agerwala, T. and Flynn, M., Comments on Capabilities, Limitations, and 'Correctness' of Petri Nets, *Proceedings of the First Annual Symposium on Computer Architectures*, New York: ACM, 1973, pp. 81-86.

[2] Berthelot, G. and Terrat, R., Petri Net Theory for the Correctness of Protocols, *IEEE Trans. on Comm.*, Vol. COM-30, No. 12, December 1982, pp. 2497-2505.

[3] Bochmann, G., Finite State Description of Communication Protocols, *Computer* Networks, Vol. 2, 1978, pp.361-371.

[4] Brand, D. and Zafiropulo, P., On Communicating Finite-State Machines, *J. ACM*, Vol. 30, No. 2, April 1983, pp. 323-342.

[5] Cook, S., Characterizations of Pushdown Machines in Terms of Time Bounded Computers, *J. ACM*, Vol. 18, 1971, pp. 4-18.

[6] Cunha, P. and Maibaum, T., A Synchronization Calculus for Message-Oriented Programming, *Proc. 2nd International Conf. on Distributed Computing Systems*, April 1981, pp. 433-445.

[7] Gouda, M. and Rosier, L., Priority Networks of Communicating Finite State Machines, accepted for publication in the *SIAM Journal on Computing*.

[8] Hack, M., Decidability Questions for Petri Nets, Ph.D. dissertation, Department of Electrical Engineering, MIT, 1975.

[9] Hopcroft, J. and Pansiot, J. On the Reachability Problem for 5-Dimensional Vector Addition Systems, *Theor. Computer Science,* Vol. 8, 1979, pp. 135-159.

[10] Hopcroft, J. and Ullman, J., "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley, Reading, Mass., 1979.

[11] Jones, N., Lien, Y. and Laaser, W., New Problems Complete for Nondeterministic Logspace, *Math. Systems Theory*, Vol. 10, 1976, pp. 1-17.

[12] Karp, R. and Miller, R., Parallel Program Schemata, *J. of Computer and System Sciences*, Vol. 3, No.2, 1969, pp.147-195.

[13] Kosaraju, S., Limitations of Dijkstra's Semaphore Primitives and Petri Nets, *Operating Systems Review*, Vol. 7, No. 4, Oct. 1973, pp. 122-126.

[14] Minsky, M., "Computation: Finite and Infinite Machines", Prentice Hall, Englewood Cliffs, NJ, 1967.

[15] Parnas, D., On a Solution to the Cigarette Smokers' Problem (Without Conditional Statements), *Communications of the ACM*, Vol. 18, No. 3, March 1975, pp. 181-183.

[16] Patil, S., Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes, Group Memo 57, Project MAC, MIT, February 1971, 18 pages.

[17] Peterson, J., "Petri Net Theory and the Modeling of Systems", Prentice Hall, Englewood Cliffs, NJ, 1981.

[18] Rackoff, C., The Covering and Boundedness Problems for Vector Addition Systems, *Theor. Comput. Sci.*, Vol. 6, 1978, pp. 223-231.

[19] Rosier, L. and Gouda, M., On Deciding Progress for a Class of Communication Protocols, in the *Proceedings of the Eighteen Annual Conference on Information Sciences and Systems*, Princeton Univ., 1984.

[20] Raeuchle, T. and Toueg, S., Exposure to Deadlock for Communicating Processes is Hard to Detect, Tech. Rep. No. 83-555, Cornell University, Department of Computer Science.

[21] Savitch, W., Relationships between Nondeterministic and Deterministic Tape Complexities, *J. of Computer and System Sciences*, Vol. 4, No. 2, 1970, pp. 177-192.

[22] Sudborough, I., On Tape-Bounded Complexity Classes and Multihead Finite Automata, *J. Computer and Systems Sciences*, Vol. 10, No. 1, 1975, pp. 62-76.

[23] Sunshine, C., Formal Modeling of Communication Protocols, USC/Inform. Sc. Institute, Research Report 81-89, March 1981.

[24] Tannenbaum, A., "Computer Networks", Prentice Hall, Englewood Cliffs, NJ, 1981.

[25] Valiant, L. and Paterson, M., Deterministic One-Counter Automata, *J. of Computer and System Sciences*, Vol. 10, 1975, pp. 340-350.

[26] Yu. Y. and Gouda, M., Unboundedness Detection for a Class of Communicating Finite State Machines, *Information Processing Letters*, 17, December 1983, pp. 235-240.

[27] Zafiropulo, P., et. al., Towards Analyzing and Synthesizing Protocols, *IEEE Trans. on Comm.*, Vol. COM-28, No. 4, April 1980, pp. 651-661.