LIVELOCK DETECTION IN NETWORKS OF COMMUNICATING FINITE STATE MACHINES

M. G. Gouda, C. H. Chow† and S. S. Lam†

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

TR-84-10 March 1984

†Work supported by National Science Foundation Grant No. ECS83-04734.

ABSTRACT

A livelock arises in a communication protocol if the communicating entities in the protocol keep on exchanging messages while no "useful work" is being done. We investigate this phenomenon in networks of communicating finite state machines. In particular, we show that it is undecidable in general whether the communication of any such network can reach a livelock. We also discuss a number of efficient algorithms to detect livelocks in some special classes of networks. These include: (i) networks of two machines with bounded communication, (ii) networks of two machines where only one channel has bounded communication, (iii) networks of two machines where one machine sends a single message type, and (iv) networks of any number of machines that have closed covers. The basic idea in each of these algorithms is to construct an abstract representation of the reachability graph of a given network; then search it to detect livelocks. Such abstract representations are much smaller than the original reachability graphs. (In fact, they are always finite whereas the corresponding reachability graphs are often infinite.) We apply our algorithms to detect livelocks in several networks that model various real protocols (the negotiation mechanism in virtual terminal protocols, the alternating-bit protocol, a call establishment/clearing protocol, and a version of the CSMA protocol).

Table of Contents

1. Introduction

The model of communicating finite state machines is useful in the specification [4,28,29], analysis [2,5,9,12,13,15,20,21,23,24,26,28,29,30,31,32,33,34], and synthesis [3,6,10,14,23] of communication protocols. The procedure for modeling and analyzing a communication protocol using this model typically proceeds as follows:

- First, the protocol is defined as a network of communicating finite state machines. Each machine in the network has a finite number of states and state transitions (called nodes and edges respectively in this paper). Each state transition of a machine is accompanied by the sending of a message into a channel or receiving of a message from a channel. Channels are assumed to be directional (delivering messages from one machine to another) and FIFO. In general, a network may have an arbitrary topology.

- Second, the network defined is analyzed to ensure that its communication satisfies some nice properties such as boundedness [33], freedom from deadlocks [26,27,31,33] and freedom from unspecified receptions [11].

Examples of some realistic protocols that can be modeled and analyzed using this procedure include: the alternating-bit protocol [1], the Binary Synchronous protocol [22], and the call establishment/clearing procedures in X.21 [24,31] and X.25 [12,25].

One property that should be satisfied by networks of communicating finite state machines modeling real protocols, is freedom from livelocks. A livelock occurs when the machines in a network keep exchanging messages but no "useful work" is being done. The characterization of livelocks in distributed systems and communication protocols was considered by Hajek [17] and Lai [19]. Sherman and Rudin [30] characterized livelocks in networks of communicating finite state machines and pointed out the importance of detecting livelocks in such networks during protocol validation. In this paper we extend the work of Sherman and Rudin by presenting a number of efficient algorithms to detect livelocks in some special classes of networks. (As shown later, the detection of livelocks in a general network of communicating finite state machines is undecidable.)

This paper is organized as follows: Networks of communicating finite state machines are presented formally in Section 2. The concept of livelocks in such networks is defined in Section 3. In Section 4, we present algorithms to detect livelocks in networks belonging to the following classes:

- Networks of two machines with bounded communication.

- Networks of two machines where one of the two channels is bounded.

- Networks of two machines where one machine sends a single message type.

- Networks with any number of machines that have closed covers (as defined in [9].)

Concluding remarks are in Section 5. Proofs of all our theorems are given in the Appendix.

2. Networks of Communicating Finite State Machines

A *communicating finite state machine* M is a labelled directed graph with two types of edges, namely *sending* and *receiving edges*. A sending (or receiving) edge is labelled -g (or +g, respectively) for some *message* g in a finite set G of messages. A node in M whose outgoing edges are all sending (or all receiving) edges is called a *sending* (or *receiving*) *node*. A node in M whose outgoing edges include both sending and receiving edges is called a *mixed node*, and a node in M that has no outgoing edges is called a *final node*. One of the nodes in M is identified as its *initial node*, and each node in M is reachable by a directed path from the initial node.

Let M and N be two communicating finite state machines with the same set G of messages. Let (M,N) denote the network consisting of machines M and N connected by two FIFO channels in opposite directions.

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 in G. Informally, a state [v,w,x,y] means that the executions of M and N have reached nodes v and w respectively, while the input channels of M and N store the strings x and y respectively.

The *initial state* of 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 string.

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 four conditions is satisfied:

- e is a sending edge, labelled -g, from v to v' in M, and s'=[v',w,x,y.g], where "." is the concatenation operator.
- e is a sending edge, labelled -g, form w to w' in N, and s'=[v,w',x.g,y].
- e is a receiving edge, labelled +g, from v to v' in M, and s'=[v',w,x',y], where x=g.x'.
- e is a receiving edge, labelled +g, from w to w' in N, and s'=[v,w',x,y'], where y=g.y'.

Let s and s' be two states of 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 $s_1,...,s_r$ such that $s=s_1$, $s'=s_r$ and s_{i+1} follows s_i for i=1,...,r-1.

A state s of network (M,N) is said to be *reachable* iff it is reachable from the initial state of (M,N). Next, we use the concept of reachable states to define what it means for the communication of a network (M,N) to be free from deadlocks and unspecified receptions, and to be bounded.

A reachable state [v,w,x,y] of a network (M,N) is a *deadlock state* iff (i) both v and w are receiving nodes, and (ii) x=y=E (the empty string). If no reachable state of network (M,N) is a deadlock state, then the communication of (M,N) is said to be *deadlock-free.*

A reachable state [v,w,x,y] of a network (M,N) is an *unspecified reception state* iff one of the following two conditions is satisfied:

- $x=g_1 \cdot g_2 \cdot \ldots \cdot g_k$ ($k \geq 1$), and v is a receiving node and none of its outgoing edges is labelled $+g_1$.
- $y=g_1 \cdot g_2 \cdot \ldots \cdot g_k$ ($k \geq 1$), and w is a receiving node and none of its outgoing edges is labelled $+g_1$.

If no reachable state of (M,N) is an unspecified reception state, then the communication of (M,N) is said to be *free from unspecified receptions.*

The input channel of machine M (N) in network (M,N) is said to be *bounded by* K iff for every reachable state [v,w,x,y] of (M,N), $|x| \leq K$ ($|y| \leq K$). A channel in (M,N) is said to be *bounded* iff it is *bounded by* K, for some nonnegative integer K. The communication of (M,N) is said to be bounded (by K) iff each of the two channels in (M,N) is bounded (by K).

3. Livelocks

A *marked network* is a triple (M,N,m), where (M,N) is a network of two communicating finite state machines M and N, and m is a function, called the *marking* of the network, that assigns to each edge in M or N either the value "p" or the value "n". Let e be an edge in machine M or N. If m(e)=p then e is called a *progress edge*, otherwise m(e)=n, and e is called a *nonprogress edge.*

Let (M,N,m) be a marked network and let C and D be two directed cycles in M and N respectively. The pair (C,D) is called a *livelock* in (M,N,m), iff the following three conditions are satisfied:

i. All the edges of cycle C ,in M, are nonprogress.

ii. All the edges of cycle D ,in N, are nonprogress.

iii. There exists a sequence $(s_1,\ldots,s_r)$ of reachable states of network (M,N) such that the following two conditions hold:

a. For i=1,...,r-1, state s_{i+1} follows s_i over an edge e_i in M or N. Also state s_1 follows s_r over an edge e_r in M or N.

b. The set of edges $\{e_1,e_2,...,e_r\}$ constitutes the two cycles C and D.

This sequence $(s_1,...,s_r)$ is called a *nonprogress cycle* for the livelock (C,D). (Notice that a livelock may have more than one nonprogress cycle.)

The *livelock detection problem* can be stated as follows: Decide for any given marked network whether it is free from livelocks. Later we will show that this problem is undecidable in general, but can be decided for some special classes of networks. But first we motivate the problem by detecting livelocks in two protocols that are modeled as marked networks.

Example 1 (A Symmetric Negotiation Mechanism in a VTP): The negotiation mechanism in a virtual terminal protocol (VTP) is provided to allow different machines (terminals or host computers) with different capabilities to agree on a common set of capabilities (e.g. line length, page size, etc.) The negotiation mechanism is either asymmetric or symmetric [7]. In an asymmetric mechanism, negotiation can be initiated only by the host computer side. In a symmetric mechanism, negotiation can be initiated by either side.

Consider the two communicating finite state machines M and N in Figure 1; they model two virtual terminals in a symmetric negotiation mechanism. (Notice that the two machines have the same structure.) The exchanged messages have the following meanings:

CI_x denotes a "capability indication" message sent by machine x.

CS_y denotes a "capability selected" message. The subscripts are used in this case to distinguish between different lists of selected capalilities. For simplicity, we model the situation where there are only two possible common sets of capabilities, i.e. y=1,2. (It is straightforward to extend the model to a situation with n possible common sets of capabilities.)

Starting from node 1, machines M and N exchange CI messages that indicate the lists of capabilities they can support. After receiving the CI message of the other machine, each machine analyzes both lists of capabilities and selects a common set of capabilities. They then exchange the selected capability lists via CS messages. If the two selected capability lists are identical, the agreement is reached and each machine goes to node 6. Otherwise, each of them has to return to node 3 and selects another common set of capabilities.

One natural marking m for this network is as follows: In each machine, only the two edges from nodes 4 and 5 to node 6 are marked progress; all other edges are marked

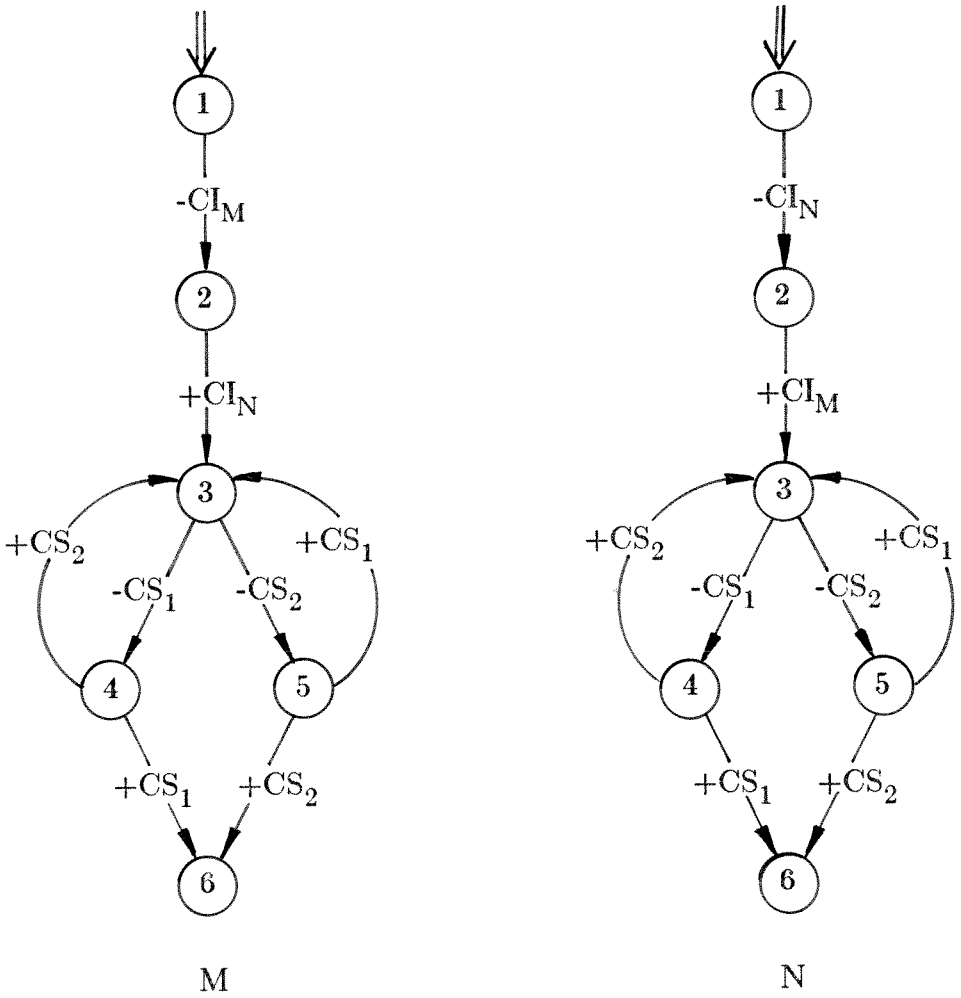


Figure 1. A symmetric negotiation mechanism for a VTP.

nonprogress.

Let C_1 (D_1) be the directed cycle in machine M (N) that starts at node 3, goes to node 4, and then returns to node 3. Also let C_2 (D_2) be the directed cycle in M (N) that starts at node 3, goes to node 5, then returns to node 3. It is straightforward to show that (C_1,D_1) and (C_2,D_2) are two livelocks for the marked network (M,N,m). Each of these livelocks can cause the two machines to go through an infinite cycling of proposal and counterproposal without effective progress.

There are two proposed solutions to prevent these livelocks. One solution [8] requires a single, network-wide algorithm which the two machines execute to compute the list of common capabilities from the list of possibilities. In another solution [18], the "capability selected" message carries an eight bit random number. If a disagreement occurs, the selected capabilities in the message with larger random number prevails.

□

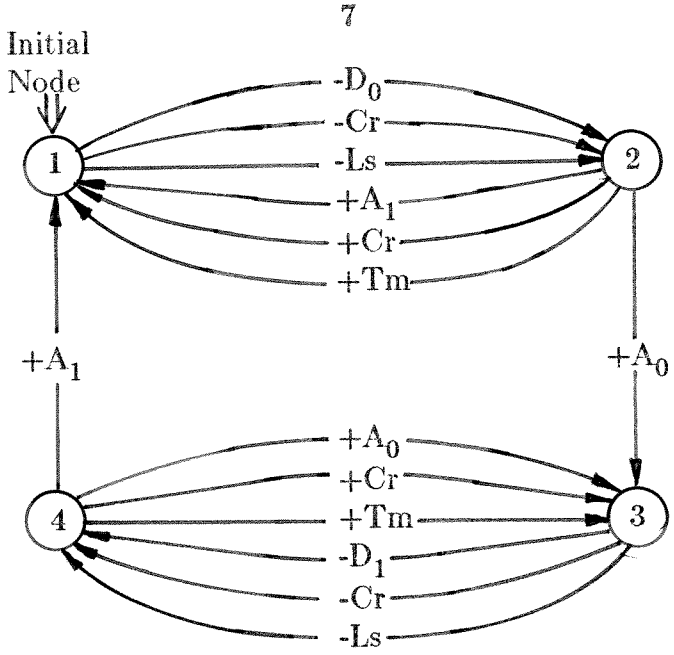
Example 2 (The Alternating-Bit Protocol): The alternating-bit protocol was proposed by Bartlett et al [1] to ensure reliable transmission of data messages from a sender to a receiver over a communication medium that can corrupt or lose transmitted messages. If a data message is corrupted or lost during transmission, or if its positive acknowledgement is corrupted or lost, then the data message is retransmitted. The retransmission can be triggered by any of the following:

- The sender receives a negative acknowledgement.
- The sender receives a corrupted message.
- The sender has waited for a specified time period, called a timeout period, to receive a response but no response has been received (indicating that either the original message or its response is lost.)

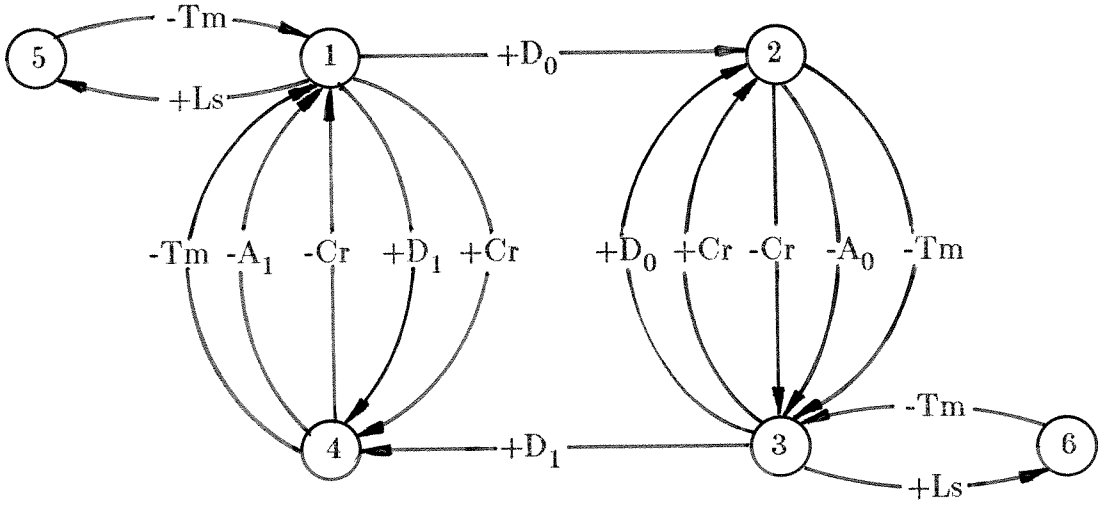
When the receiver has accepted a data message, it should be able to detect whether it has received an identical copy of this message earlier. For this reason, the value of some bit in the sender is attached to each data message sent. Whenever a data message is being retransmitted, the value of this bit remains the same; but whenever a new data message is about to be sent, the value of this bit is flipped (hence, the name "alternating-bit protocol").

Figure 2 shows a model of the alternating-bit protocol. Machines M and N model the sender and the receiver respectively. Instead of modeling the medium as a separate machine, the medium's effects are modeled as follows:

- Whenever a machine (M_1 or N_1) sends a data message g, it either sends g, or sends a special message Cr that denotes a corrupted message, or sends a special message Ls that denotes a lost message.



(a) Sender M



(b) Receiver N

Figure 2. An alternating-bit protocol.

- Whenever N sends a response message g, it either sends g, or sends a corrupted message Cr, or sends a timeout message Tm to simulate the loss of g and force M to resend the previously sent data message. Whenever N receives the message Ls, it sends a timeout message Tm to force M to resend the previously sent data message.

The other exchanged messages between M_1 and N_1 have the following meanings:

D_i (i=0,1) denotes a data message with a bit of value "i" attached to it.

A_i (i=0,1) denotes a positive acknowledgement message for D_i.

One natural marking m for network (M,N) is as follows: Only the receiving edges labelled $+A_0$ and $+A_1$ are marked progress; all other edges are marked nonprogress.

Let C (D) be the directed cycle in M (N) that starts at node 2 (2), goes through the edge labelled +Cr (-Cr) and reaches node 1 (3), then goes through the edge labelled $-D_0$ ($+D_0$) and returns to node 2 (2). It is straightforward to show that (C,D) is a livelock in the marked network (M,N,m).

This livelock represents the situation where M keeps on retransmiting the data message D_0 but the corresponding positive acknowledgements sent by N are repeatedly corrupted. One solution to prevent this livelock is to implement a counter in machine M to keep track of the number of successive message corruptions or message losses, and to report the situation to its user after a fixed number of retries.

□

The above two examples demonstrate the importance of the livelock detection problem in analyzing communication protocols. Unfortunately, the next theorem states that this problem is undecidable in general. (Its proof is in [16].)

Theorem 1: It is undecidable whether any arbitrary marked network (M,N,m) can reach a livelock.

□

Theorem 1 states that there is no algorithm to solve the livelock detection problem in general. Next we identify some special (yet interesting) classes of marked networks for which the problem is decidable.

4. Detecting Livelocks in Special Classes of Networks

In this section, we present efficient algorithms to detect livelocks in four special classes of networks. The basic idea in each of these algorithms is to construct an abstract representation of the reachability graph of the network under consideration, and then search it to detect livelocks. The efficiency of these algorithms results from the fact

that these abstract representations are much smaller than the original reachability graphs. In fact, the abstractions are always finite whereas the corresponding reachability graphs are often infinite.

4.1. Two-machine networks with bounded communication

Let (M,N,m) be a marked network whose communication is bounded; i.e. the number of its reachable states is finite. The (finite) *reachability graph* of this network is a directed graph whose vertices represent reachable states, and whose arcs represent the "follow over" relation defined in Section 2. It is possible to examine this graph to decide whether the given network is free from livelocks; however since the number of vertices and arcs in this graph are usually large, a straightforward state exploration algorithm is not efficient.

A more efficient algorithm is to examine the the fair reachability graph (to be defined later) of the network. This is because fair reachability graphs usually have smaller numbers of vertices and arcs than reachability graphs. In fact, it is shown earlier that using fair reachability graphs instead of reachability graphs yields efficient algorithms to detect deadlocks [27,32], unspecified receptions [27], and unboundedness [33]. In this section, we discuss an efficient algorithm that uses fair reachability graphs to detect livelocks, but first we state some definitions:

A state [v,w,x,y] of network (M,N) is *fair* iff |x|=|y|, where |x| is the number of messages in string x. Obviously, the initial state of (M,N) is fair.

Let s and s' be two fair states of (M,N); and let e and f be two edges in M and N respectively. s' *fairly follows* s *over* e and f iff there exists a state s" such that either (s" follows s over e and s' follows s" over f) or (s" follows s over f and s' follows s" over e).

Let s and s' be two fair states of network (M,N); and let P be a directed path of edges $e_1,...,e_r$ in M, and Q be a directed path of edges $f_1,...,f_r$ in N. s' is *fairly reachable from* s *over* the edges of P and Q iff there exist fair states $s_0,s_1,...,s_r$ such that $s=s_0$,$s'=s_r$, and s_{i+1} fairly follows s_i over e_{i+1} and f_{i+1}, i=0,...,r-1. s' is *fairly reachable from* s iff s' is fairly reachable from s over the edges of some two directed paths P and Q in M and N respectively. s' is *fairly reachable* iff it is fair and is fairly reachable from the initial state of (M,N).

The following algorithm, which is based on the notion of fair reachability, can be used to decide whether any given bounded network (M,N,m) is free from livelocks.

Algorithm 1:

i. Construct the *fair reachability graph* G for the given network (M,N,m) as follows:

a. For each fairly reachable state s of the network, add one vertex, also called s for convenience, to G. (G has "vertices" and "arcs" so that they are not confused with the nodes and directed edges of the machines.)

b. For any two fairly reachable states s and s', if s' fairly follows s over some edges e and f in M and N respectively, then add an arc, labelled {e,f}, from node s to node s' in G.

ii. **If** G has a directed cycle where each arc is labelled with a set {e,f} such that m(e)=m(f)=n (such a cycle is called a *nonprogress cycle* in the fair reachability graph G)
then (M,N,m) can reach a livelock
else (M,N,m) is free from livelocks.

□

Theorem 2 (Correctness of Algorithm 1): The communication in a network (M,N,m) is free from livelocks iff there is no nonprogress cycle in the fair reachability graph of (M,N,m).

□

Theorem 2 states that fair reachability graphs can be used to detect livelocks in marked networks. This is advantageous since the fair reachability graph of a marked network is far smaller than its reachability graph, and so it requires less execution time and memory space to construct and to examine. As an example, the marked network (M,N,m) in Example 2 is bounded. Its reachability graph has 34 vertices and 50 arcs, but its fair reachability graph has 10 vertices and 26 arcs. Therefore using Algorithm 1 to detect livelocks yields a saving factor of about 2 in both time and storage.

As stated earlier, Algorithm 1 can be used when the communication of the given marked network is bounded. However, this algorithm can be also used with unbounded networks provided that their fair reachability graphs are finite. This is illustrated by the following example.

Example 3 (A Connection Establishment/Clearing Protocol): Consider the two communicating finite state machines M and N in Figure 3a; they model two machines in a simple connection establishment/clearing protocol. The exchanged messages have the following meanings:

RQST denotes a request to establish a connection.
ACPT denotes an acceptance of the request.
DATA denotes a data message to be sent over the connection.
CLEAR denotes a "clear the connection" message.

Starting from node 1, machine M can send a request RQST message to N and waits at node 2. There are two possibilities:

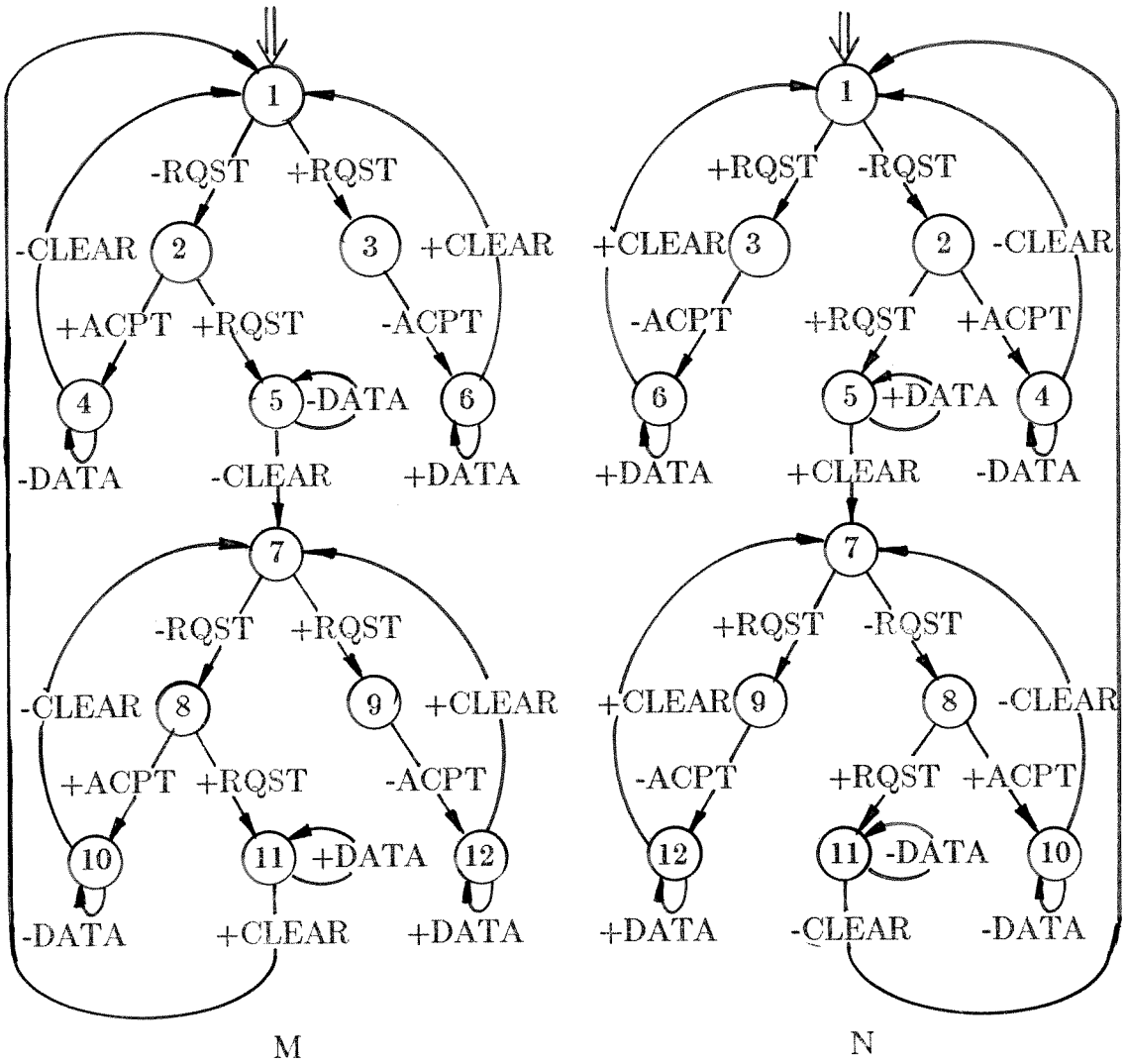


Figure 3a. A fair connection establishment/clearing protocol.

i. *An ACPT message is received.* It implies that machine N accepts the request, and the connection is established. M can now start sending data messages. After M sends all its data, it sends a CLEAR message to N and returns to node 1.

ii. *A RQST message is received.* It implies that machine N also wants to establish a connection yielding a request collision. In this case, machine M wins and can start sending its data messages; but this win by M will be remembered by both M and N so that when the next request collision occurs N will win. In other words, the two machines alternate the priority to win request collisions, and the protocol is fair.

Consider the following marking m: all edges are marked nonprogress except those labelled +DATA and +CLEAR. The reachability graph of this network is infinite. Hence it cannot be used to show that (M ,N ,m) is free from livelocks. On the other hand, the fair reachability graph of this network (in Figure 3b) is finite, and so can be used to establish that the marked network is indeed free from livelocks.

□

4.2. Two-machine networks where one channel is bounded

Consider the class of marked networks where one of the channels is bounded; the other channel may or may not be bounded. Detection of unboundedness, deadlocks, or unspecified receptions for this class of networks has been discussed earlier in [5,26]; in this section, we discuss detection of livelocks for the same class. Clearly, reachability graphs for this class of networks can have an infinite number of states, and so they cannot be used to detect livelocks. Instead, we show in the next theorem that fair reachability graphs for this class is finite, and so they can be used to detect livelocks.

Theorem 3: Let (M,N,m) be a marked network where the input channel of M or N is bounded. The fair reachability graph of (M,N,m) is finite.

□

Since the fair reachability graph for a network, where one channel is bounded, is finite, then livelocks in such a network can be detected using Algorithm 1 in the previous section.

Note that Theorem 3, along with the algorithm in [27], suggests a new algorithm to detect deadlocks and unspecified receptions for the class of networks where one of the two channels is bounded. Other algorithms to solve this problem have appeared earlier in [5,26].

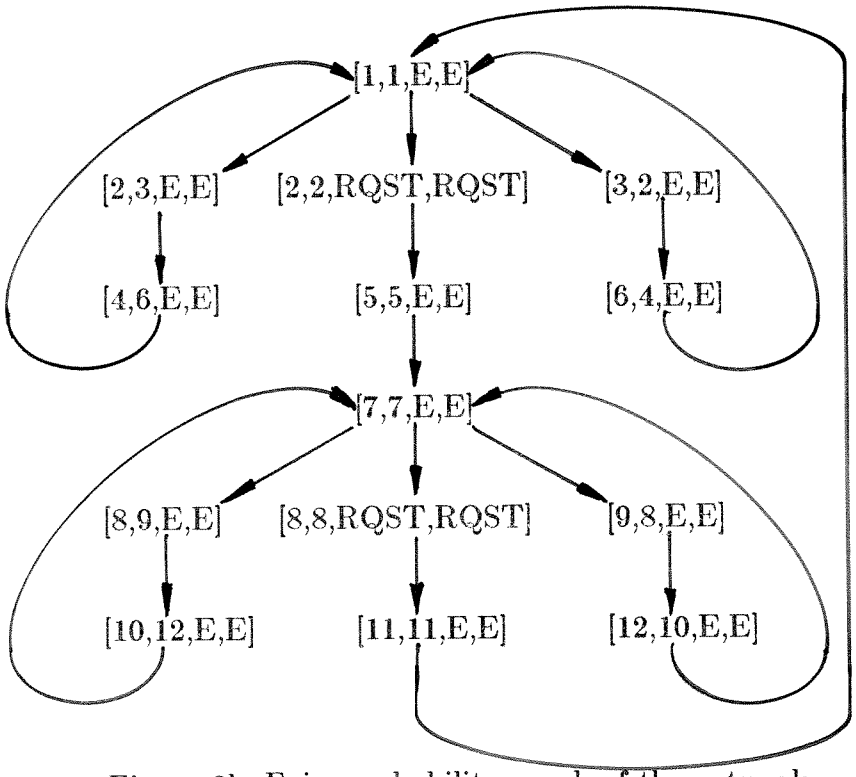


Figure 3b. Fair reachability graph of the network

4.3. Two-machine networks where one machine sends a single message type

Consider the class of marked two-machine networks where one machine sends a single message type. Detection of unboundedness, deadlocks, or unspecified receptions for this class of networks has been discussed earlier in [14,26]; in this section we discuss a sufficient condition that ensures freedom of livelocks for this class. (Finding a necessary and sufficient condition to detect livelocks is still an open question.)

Let (M,N,m) be a marked network and assume without loss of generality that machine N sends only one type of message. Since N sends only one type of message, we do not have to distinguish between the different messages in the input channel of M. Only the number of messages matters. Therefore, a state of this network can be represented as [v,w,k,y] where v and w are nodes in machines M and N respectively, y is a string of messages that defines the contents of the input channel of machine N, and k is a nonnegative number that defines the number of messages in the input channel of M.

A state [v,w,k,y] of this network is called *limited* iff $|y| \leq 1$. Notice that the initial state $[v_0,w_0,0,E]$ is limited.

An *abstract state* of this network is a four-tuple $[v,w,\tilde{\omega},y]$, where v and w are nodes in M and N respectively, $|y| \leq 1$ and ω is a symbol that represents a "very large number" of messages in the input channel of machine M.

Next we extend the definition of "follow over" to abstract states. Let $s_1=[v,w,\omega,y]$ be an abstract state of network (M,N,m). An abstract state s_2 is said to *follows* s_1 *over* edge e iff one the following four conditions is satisfied:

- e is a sending edge labelled -g from v to v' in M ,
  and $s_2=[v',w,\omega,y.g]$ where "." is the concatenation operator.
- e is a sending edge labelled -g from w to w' in N,
  and $s_2=[v,w',\omega,y]$.
- e is a receiving edge labelled +g from v to v' in M,
  and $s_2=[v',w,\omega,y]$.
- e is a receiving edge labelled +g from w to w' in N,
  and $s_2=[v,w',\omega,y']$ where $y=g.y'$.

The following algorithm can be used to decide whether a marked network (M,N,m), where M sends one type of message, is free from livelocks.

Algorithm 2:

i. Construct the *limited reachability graph* G for the given network (M,N,m) as

follows:

Initially G contains only one vertex labelled with the initial state of (M,N).

while G has a vertex labelld with a state (abstract state) s
such that vertex s has no outgoing arcs in G, and
there exists a limited state (abstract state) that follows state s
{* Notice that s refers to both the vertex and the state that labels the vertex *}
do **if** s is a limited state
then for each limited state s'=[v,w,k,y] that follows s over edge e
do **if** vertex s is (or has an ancestor vertex) labelled with [v,w,k',y]
where k'<k
then if there is a vertex s" labelled [v,w,ω,y] in G
then add an arc labelled e from vertex s to s"
else add a vertex s' labelled [v,w,ω,y] to G;
add an arc labelled e from vertex s to vertex s'
else if vertex s' has an ancestor vertex s" labelled with
[v,w,k',y] where k'=k
then add an arc labelled e from vertex s to vertex s"
else add a vertex labelled [v,w,k,y] to G;
add an arc labelled e from vertex s to vertex s'.
else {* s is an abstract state *}
for each abstract state s'=[v,w,ω,y] that follows s over edge e
do **if** there is a vertex s" labelled [v,w,ω,y] in G
then add an arc labelled e form vertex s to vertex s",
else add a vertex s' labelled [v,w,ω,y] to G;
add an arc labelled e from vertex s to vertex s'.

ii. Remove each directed cycle in G whose arc labels are all edges in M or are all edges in N.

iii. **If** G has a directed cycle where each arc is labelled with an edge e such that m(e)=n (such a cycle is called a nonprogress cycle in the abstract reachability graph G)
then (M,N,m) can reach a livelock.
else (M,N,m) is free from livelocks.

□

Termination of this algorithm is guaranteed by the fact that the number of limited and abstract states is finite. This follows from the following three assertions. (Let [v,w,k,y] be a typical limited or abstract state of (M,N).):

i. The number of distinct values of v (w) is bounded from above by the number m (n) of nodes in M (N).

ii. The number of distinct values of y is bounded from above by t+1, where t is the number of distinct message types sent by M.

iii. The value of k is either ω or bounded from above by m.n.(t+1).

Theorem 4 (Correctness of Algorithm 2): Let (M,N,m) be a marked network where machine M sends one type of message. The communication of (M,N,m) is free from livelocks if there is no nonprogress cycle in the limited reachability graph of (M,N,m).

□

As an example, consider the network (M,N) in Figure 4a. Machine M models a user, while N models a clock. Clock N sends to the user one type of message, namely a TICK message. The user on the other hand can send a $RQST_1$ message followed by a $RQST_2$ message to modify the rate of the TICK's sent by the clock. One possible marking m of this network is as follows: Only the edges labelled +TICK or $+RQST_2$ are marked progress; all other edges are marked nonprogress. Figure 4b shows the limited reachability graph of (M,N,m). Since there are no nonprogress cycles in G, (M,N,m) is free from livelocks.

4.4. Networks with closed covers

The technique of closed covers is presented in [9] to prove that a network is free from deadlocks and unspecified receptions. One advantage of this technique is that it can be used with networks whose communications are unbounded. (No other technique seems to be successful with such networks.) In this section, we extend this technique to detect livelocks in a marked network.

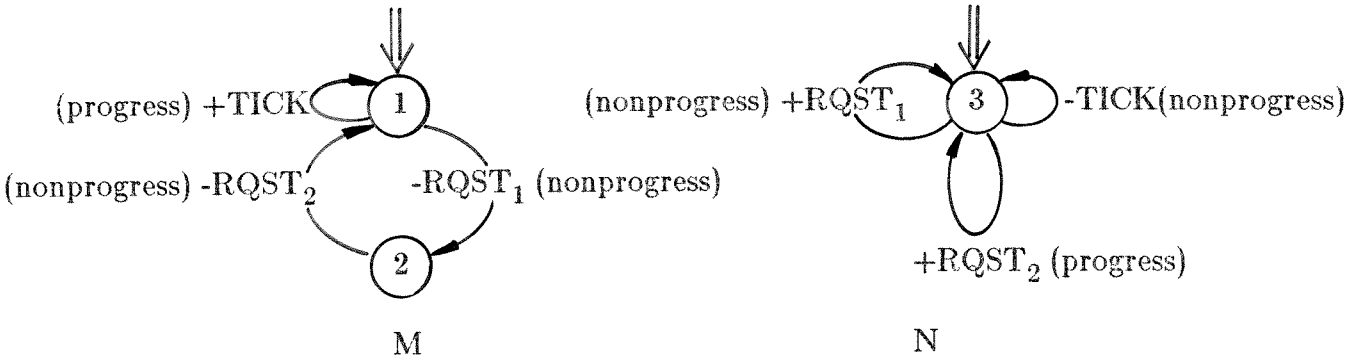
A *closed cover* C for a network (M,N) is a set of states of (M,N) that satisfies the following four conditions:

i. The initial state of (M,N) is in C.

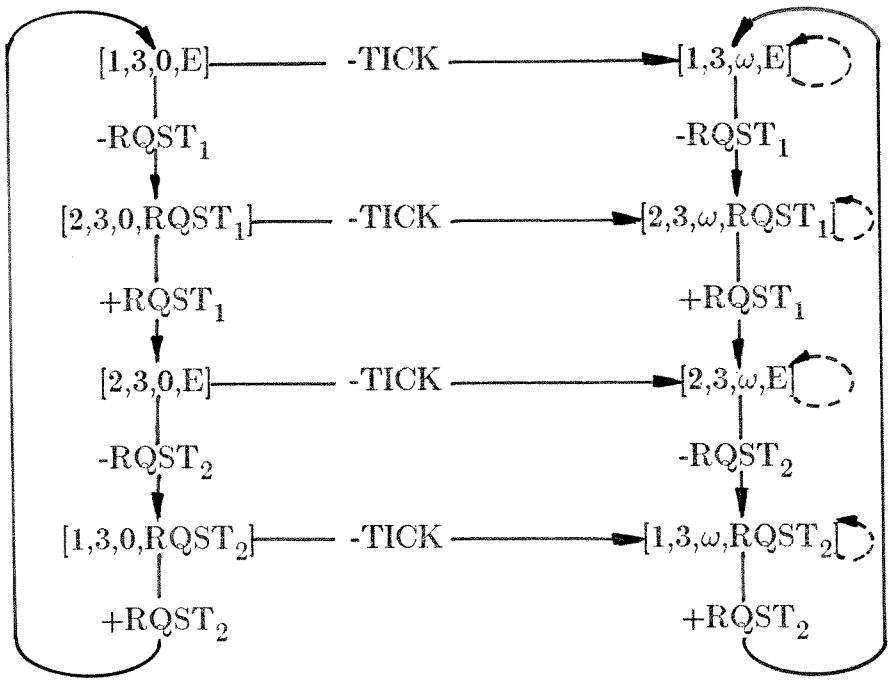
ii. Each directed cycle in the directed graph of M or N must have at least one node referenced in some state in C.

iii. The *acyclic version* AM of M with respect to C can be constructed from M by partitioning each node v, which is referenced in some state in C, into two nodes: One node, called the input version of v, has all the output edges of v and no input edges; the other node, called the output version of v, has all the input edges of v and no output edges. Similarly, the acyclic version AN of N with respect to C can be defined. The third condition can now be defined in terms of these acyclic versions. If the network (AM,AN) starts at a state s_1 in C and if it reaches a state s_2 after which no other state is reachable, then state s_2 must also be in C.

iv. The following condition should be satisfied for any state [v,w,x,y] in C, and for any two paths p and q, in the acyclic versions AM and AN, that start



(a) Network (M,N)



(b) Limited reachability graph of the network

Figure 4.

with the input versions of v and w respectively, and terminate at the output versions of some nodes: Let s_i (r_i) be the sequence of sent (received) messages along path i, where i=p,q; then

either [$(x.s_q \prec r_p)$ and $(y.s_p \prec r_q)$],
or [not$(x.s_q \prec r_p)$ and not$(y.s_p \prec r_q)$],

where

"." denotes the string concatenation operator, and
"$\prec$" denotes "is a proper prefix of".

It is shown in [9] that if a network has a closed cover then its communication is free from deadlocks and unspecified receptions. In this section, we show how to use the closed cover of such a network to decide whether its communication is also free from livelocks. But first we give an example of a closed cover; in particular we state and verify a closed cover for the network (M,N) in Figure 5a. M is a sender that sends data messages to a receiver N, and N responds by sending replies back to M. The exchanged messages between M and N have the following meanings:

STX	denotes the start-of-text character in a data message.
TXT	denotes a text character in a data message.
ETX	denotes the end-of-text character in a data message.
RPL	denotes a reply message.

Next, we show that the set C = {[1,1,E,E],[3,2,E,TXT],[4,2,E,ETX]} is a closed cover for the network (M,N) in Figure 5a:

i. First, the initial state [1,1,E,E] of (M,N) is in C.

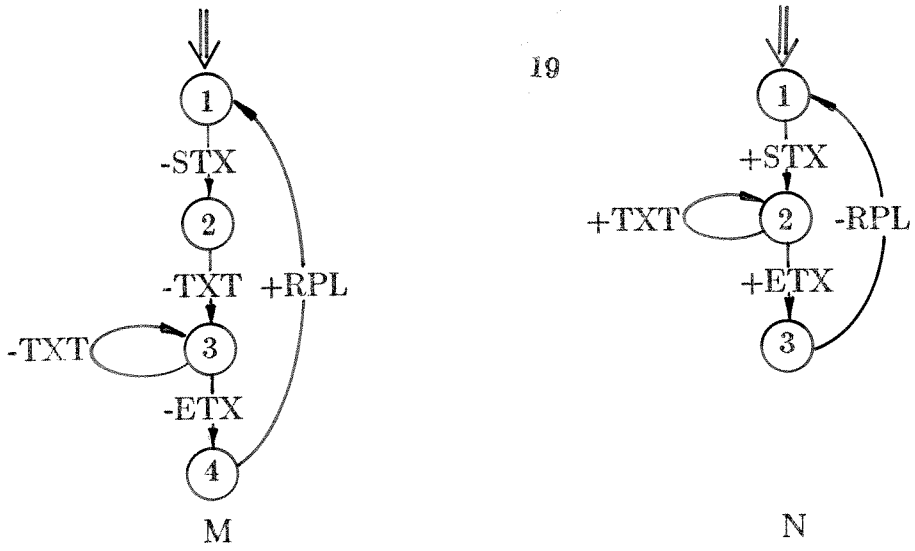
ii. Since node 3 in M and nodes 1 and 2 in N are referenced in C, every directed cycle in M or N has one node referenced in C.

iii. The acyclic versions AM and AN of M and N (respectively) with respect to C are shown in Figure 5b. If the network (AM,AN) starts at state [1,1,E,E], it must end its communication at [3,2,E,TXT]. If (AM,AN) starts at state [3,2,E,TXT], it must end at either [3,2,E,TXT] or [4,2,E,ETX]; both are in C. Similarly, if (AM,AN) starts at state [4,2,E,ETX], it must end its communication at [1,1,E,E].

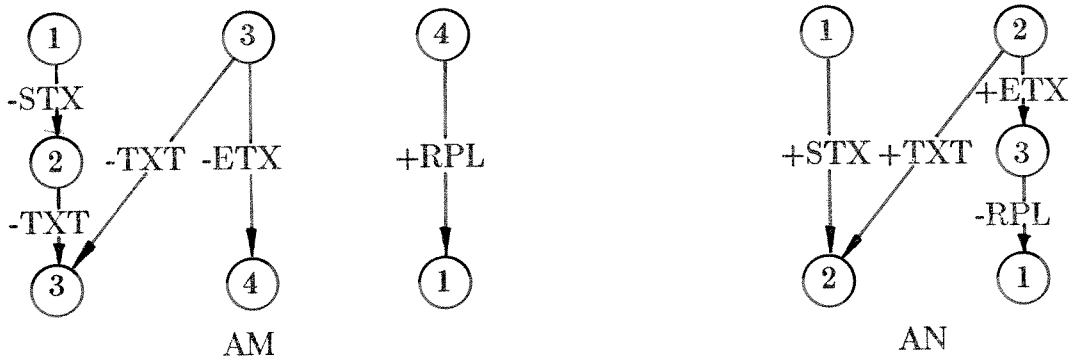
iv. For any state [v,w,x,y] in C, and for any two paths p and q, in the acyclic versions AM and AN, that start with the input versions of v and w respectively, and terminate at the output versions of some nodes: Let s_i (r_i) be the sequence of sent (received) messages along path i, where i=p,q; the following boolean expression is true.

[not$(x.s_q \prec r_p)$ and not$(y.s_p \prec r_q)$],

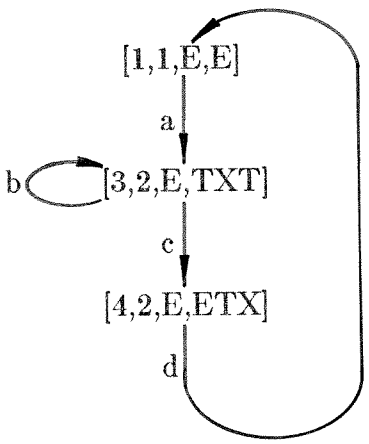
This completes the proof that C is a closed cover of (M,N).



(a) Network (M,N)



(b) Acyclic versions of M and N with respect to closed cover
C={[1,1,E,E],[3,2,E,TXT],[4,2,E,ETX]}



List of arc labels and their markings

a = { (1,-STX,2),(2,-TXT,3),(1,+STX,2) }	nonprogress
b = { (3,-TXT,3),(2,+TXT,2) }	progress
c = { (3,-ETX,4),(2,+TXT,2) }	progress
d = { (2,+ETX,3),(3,-RPL,1),(4,+RPL,1) }	nonprogress

(c) Closed cover graph for (M,N) with respect to closed cover C

Figure 5.

The following algorithm can be used to decide whether a marked network (M,N,m) that has a closed cover C, is free from livelocks.

Algorithm 3:

i. Construct the closed cover graph G for (M,N,m) as follows:

   a. For each state s in C, add a vertex, also called s, to G.

   b. Let AM and AN be respectively the acyclic versions of M and N with respect to C. If the network (AM,AN) can reach from state s in C to state s' in C over a finite sequence $<e_0,e_1,...,e_r>$ of directed edges in M or N,[1] then add an arc from vertex s to vertex s' in G; this arc is labelled with the set of edges $\{e_0,e_1,...,e_r\}$, provided that no such arc is already in G.

ii. **If** G has a directed cycle where each arc is labelled with a set $E=\{e_1,...,e_k\}$ such that m(e)=n for each edge e in the set E (such a cycle is called a nonprogress cycle in the closed cover graph G)
then (M,N,m) can reach a livelock.
else (M,N,m) is free from livelocks.

Theorem 5 (Correctness of Algorithm 3): Let (M,N,m) be a marked network and C be a closed cover for network (M,N). The communication of (M,N,m) is free from livelocks iff there is no nonprogress cycle in the closed cover graph of (M,N,m).

□

As an example, Figure 5c shows the closed cover graph G of the closed cover C={[1,1,E,E], [3,2,E,TXT],[4,2,E,ETX]} for network (M,N) in Figure 5a. Notice that each directed edge e in M or N is defined in G by a tuple (i, j, k), where

i	is the source node of e,
j	is the label of e, and
k	is the destination node of e.

and each arc in G is labelled $\{e_0,e_1,...,e_r\}$. One possible marking m for the network is as follows: Only the edges labelled +TXT in N are marked progress; all other edges are marked nonprogress. Since none of the cycles in the closed cover graph in Figure 5c is a nonprogress cycle, the communication of (M,N,m) is free from livelocks. The next example demonstrates that this technique can be easily extended to protocols with any number of communicating machines not just two machines.

---

[1] In other words, there exists a finite subsequence $<s_0,s_1,...,s_{r+1}>$ of states of (AM,AN) such that $s=s_0$, $s'=s_{r+1}$, and s_{i+1} follows s_i over e_i, i=0,1,...,r.

Example 4 (A CSMA Protocol): Consider the four communicating finite state machines M_1,M_2,M_3,M in Figure 6. Machines M_1,M_2, and M_3 model three identical stations connected to a medium that is modeled by machine M. The exchanged messages have the following meanings:

DATA	denotes a data message.
IDLE	denotes a "virtual message" indicating that the sending machine M does not want to send a data message at this time.
CLSN	denotes a "virtual message" indicating the detection of a message collision.
OFF	denotes a "virtual message" indicating the detection of a "carrier-off" condition.

Starting at node 1, each of the three stations M_1, M_2 and M_3 sends either a DATA message or an IDLE message. There are three possibilities:

- If all the three stations send IDLE messsages, the medium responds by sending an OFF message to each of them. All stations then return to their initial nodes.

- If only one station sends a DATA message, the medium responds by sending the DATA message to each of the three stations. All stations then return to their initial nodes. This corresponds to the successful data delivery situation.

- If more than one station send DATA messages, the medium responds by sending a CLSN message to each of the three stations. All stations then return to their initial nodes.

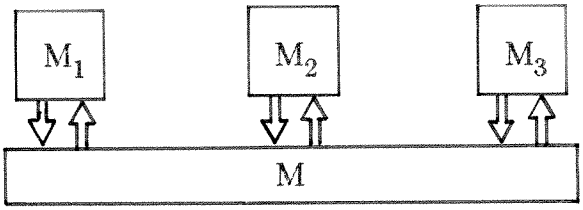
One natural marking m for this network is as follows: Only the edges labelled +DATA and +OFF are marked progress; other edges are marked nonprogress. The closed cover graph of this network is shown in Figure 7. Notice that the edge notation in the arc label, (1,M_1/-IDLE,2), represents an edge in machine M_1 which is from node 1 to node 2 and labelled with -IDLE. For simplicity reason, only source of the arcs are listed. (The other arcs can be obtained by reordering the edges in the arc labels shown in Figure 7.) Since the arcs labelled A, D, F, G are marked nonprogress and each of them forms a cycle, the marked network (M_1,M_2,M_3,M,m) can reach a livelock. Each of the nonprogress arcs represents a message collision situation.

□

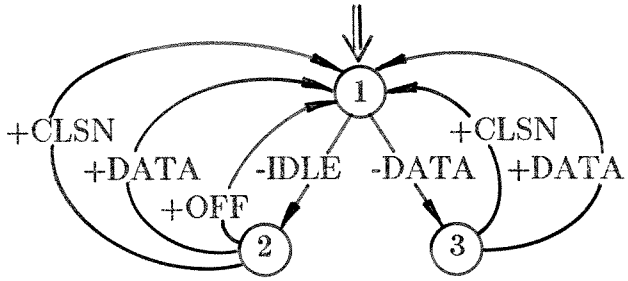
5. Concluding Remarks

The problem of detecting livelocks in networks of communicating finite state machines has been shown to be undecidable in general. However for some (interesting) special classes of networks, the detection of livelocks is decidable and we have presented three algorithms to do so.

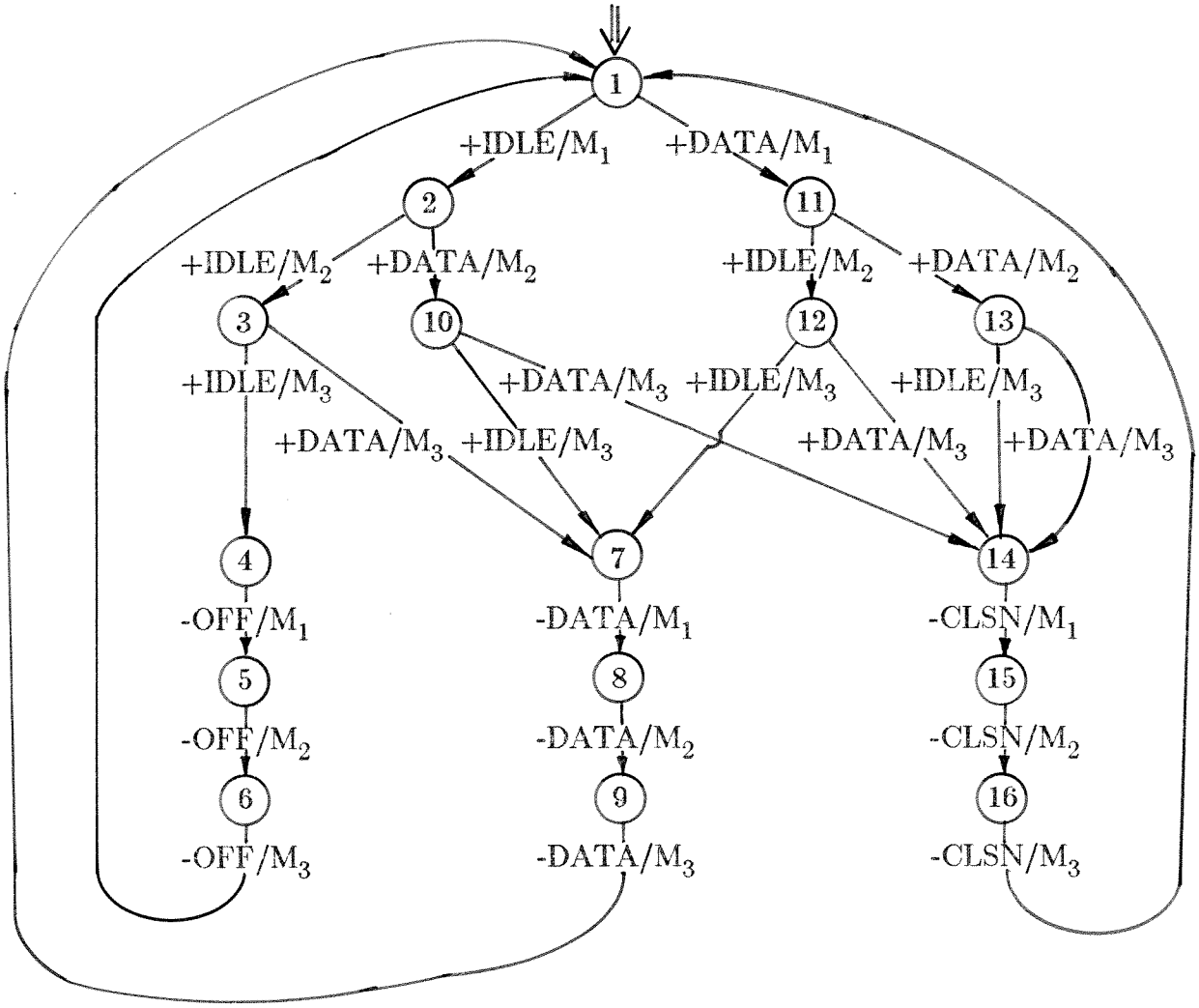
A livelock in a network of communicating finite state machines is characterized in this paper by a nonprogress cycle in the reachability graph of the network. This



(a) network (M_1,M_2,M_3,M)



(b) machine M_1, M_2, or M_3



(c) machine M

Figure 6. A CSMA protocol.

List of arc labels and their markings

A={(1,M_1/-IDLE,2),(1,+IDLE/M_1,2),(1,M_2/-IDLE,2),(2,+IDLE/M_2,3) nonprogress
(1,M_3/-IDLE,2),(3,+IDLE/M_3,4),(4,-OFF/M_1,5),(2,M_1/+OFF,1)
(5,-OFF/M_2,6),(2,M_2/+OFF,1),(6,-OFF/M_3,1),(2,M_3/+OFF,1)}

B={(1,M_1/-IDLE,2),(1,+IDLE/M_1,2),(1,M_2/-IDLE,2),(2,+IDLE/M_2,3) progress
(1,M_3/-DATA,3),(3,+DATA/M_3,7),(7,-DATA/M_1,8),(2,M_1/+DATA,1)
(8,-DATA/M_2,9),(2,M_2/+DATA,1),(9,-DATA/M_3,1),(3,M_3/+DATA,1)}

C={(1,M_1/-IDLE,2),(1,+IDLE/M_1,2),(1,M_2/-DATA,3),(2,+DATA/M_2,10) progress
(1,M_3/-IDLE,2),(10,+IDLE/M_3,7),(7,-DATA/M_1,8),(2,M_1/+DATA,1)
(8,-DATA/M_2,9),(3,M_2/+DATA,1),(9,-DATA/M_3,1),(2,M_3/+DATA,1)}

D={(1,M_1/-IDLE,2),(1,+IDLE/M_1,2),(1,M_2/-DATA,3),(2,+DATA/M_2,10) nonprogress
(1,M_3/-DATA,3),(10,+DATA/M_3,14),(14,-CLSN/M_1,15),(2,M_1/+CLSN,1)
(15,-CLSN/M_2,16),(3,M_2/+CLSN,1),(16,-CLSN/M_3,1),(3,M_3/+CLSN,1)}

E={(1,M_1/-DATA,3),(1,+DATA/M_1,11),(1,M_2/-IDLE,2),(11,+IDLE/M_2,12) progress
(1,M_3/-IDLE,2),(12,+IDLE/M_3,7),(7,-DATA/M_1,8),(3,M_1/+DATA,1)
(8,-DATA/M_2,9),(2,M_2/+DATA,1),(9,-DATA/M_3,1),(2,M_3/+DATA,1)}

F={(1,M_1/-DATA,3),(1,+DATA/M_1,11),(1,M_2/-IDLE,3),(11,+IDLE/M_2,12) nonprogress
(1,M_3/-DATA,3),(12,+DATA/M_3,14),(14,-CLSN/M_1,15),(3,M_1/+CLSN,1)
(15,-CLSN/M_2,16),(2,M_2/+CLSN,1),(16,-CLSN/M_3,1),(3,M_3/+CLSN,1)}

G={(1,M_1/-DATA,3),(1,+DATA/M_1,11),(1,M_2/-DATA,3),(11,+DATA/M_2,13) nonprogress
(1,M_3/-IDLE,2),(13,+IDLE/M_3,14),(14,-CLSN/M_1,15),(3,M_1/+CLSN,1)
(15,-CLSN/M_2,16),(3,M_2/+CLSN,1),(16,-CLSN/M_3,1),(2,M_3/+CLSN,1)}

H={(1,M_1/-DATA,3),(1,+DATA/M_1,11),(1,M_2/-DATA,3),(11,+DATA/M_2,13) nonprogress
(1,M_3/-DATA,3),(13,+DATA/M_3,14),(14,-CLSN/M_1,15),(3,M_1/+CLSN,1)
(15,-CLSN/M_2,16),(3,M_2/+CLSN,1),(16,-CLSN/M_3,1),(3,M_3/+CLSN,1)}

Figure 7. Closed cover graph for the CSMA protocol in Figure 6.

characterization can be used to distinguish two types of livelocks, as suggested by Hajek [17]:

i. *Temporary blocking:* In this type of livelocks, there is a way for the network to "exit" the nonprogress cycle after it is reached. Examples 1 and 2 in this paper illustrate two instances of this rather "mild" livelock.

ii. *Threshold of no return:* In this type of livelocks, there is no way for the network to exit the nonprogress cycle after it is reached. Examples of this "severe" livelock can be found in [16,17].

It is straightforward to extend Algorithms 1 and 3 to check the type of a detected livelock once it is detected. (Recall that Algorithm 2 does not detect livelocks; it can only prove freedom from livelocks.)

This work suggests two open problems that merit further research. The first problem is to find a necessary and sufficient condition to detect livelocks in networks of two machines where one machine sends a single message type. (In Section 4.3, we have merely presented a sufficient condition.) The second problem is to compute the asymptotic complexity of the algorithms presented, and to estimate their efficiency for very large communicating finite state machines.

Appendix

Proof of Theorem 2:

[If part]: We show that if there is a livelock in the marked network (M,N,m) then there is a nonprogress cycle in the fair reachability graph of (M,N,m). Assume that there is a livelock (C,D) in (M,N,m). Therefore, the following three assertions hold:

i. All the edges of cycle C ,in M, are nonprogress.

ii. All the edges of cycle D ,in N, are nonprogress.

iii. There exists a sequence $(s_1,...,s_r)$ of reachable states of network (M,N) such that the following two conditions hold:

   a. For i=1,...,r-1, state s_{i+1} follows s_i over an edge e_i in M or N. Also state s_1 follows s_r over an edge e_r in M or N.

   b. Each edge in $\{e_1,e_2,...,e_r\}$ is either in cycle C or cycle D.

Cycles C and D are not necessarily fundamental; hence, without loss of generality, we assume that they have an equal number of edges. Let q be the number of edges in C (or D), then q=r/2. Since the state s_1 is reachable and since the sequence $(s_1,...,s_r)$ can be

executed infinitely, there exist two infinite paths α and β in M and N respectively that lead the network (M,N) into the sequence $(s_1,s_2,...,s_r)$ and repeat the sequece infinitely. These two paths α and β can be defined as follows:

$\alpha=a_1a_2.....a_nc_1.....c_qc_1.....c_qc_1....$
$\beta=b_1b_2...b_md_1.....d_qd_1.....d_qd_1.....$

where a_i (i=1,...,n) are edges in machine M,
b_i (i=1,...,m) are edges in machine N,
c_i (i=1,...,q) are edges in cycle C,
d_i (i=1,...,q) are edges in cycle D, and
assume without loss generality that $n \geq m$.

Let α_i and β_i be the i th edges in paths α and β respectively, and let s_0 denote the initial state of (M,N,m). For i=0,1,..., let s_{i+1} be the fairly reachable state that fairly follows the fairly reachable state s_i over edges α_i and β_i, i=0,.... Let s_{n+1}=[v,w,x,y] and s_{n+q+1}=[v',w',x',y']. From s_{n+1} to s_{n+q+1}, machine M goes through cycle C once and machine N goes through cycle D once, and the message sequence sent out by M (or N) is the message sequence received by N (or M) as mentioned above. Therefore v=v' and w=w'. Since x and x' both can be received by the same edge sequence of C, then x=x'. Similarly we can show that y=y'. In other words, $s_{n+1}=s_{n+q+1}$. Therefore the fairly reachable states $s_{n+1},s_{n+2},...,s_{n+q}$ form a cycle L in the fair reachability graph of the marked network (M,N,m) where arc labels are the nonprogress edges of cycles C and D. Hence L is a nonprogress cycle in the fair reachability graph of (M,N,m).

[Only If part]: This part is straightforward since the fair reachability graph is a subgraph of the reachability graph of the network.

□

Proof of Theorem 3:

Each vertex in the fair reachability graph G of (M,N,m) corresponds to a distinct fair reachable state s=[v,w,x,y], where |x|=|y|, of the network. Since the input channel of M or N is bounded by some constant K, any fair reachable state [v,w,x,y] of (M,N,m) is such that $|x|=|y| \leq K$. Therefore, there is a finite number of distinct fair reachable states of (M,N,m), and G is finite.

□

Lemma 1: The abstract reachability graph G of (M,N,m) is finite. In particular for any reachable limited state s=[v,w,k,y] in G, k is bounded by m.n.(t+1), where m is the number of nodes in machine M, n is the number of nodes in machine N, and t is the number of message types sent by machine M.

□

Proof of Lemma 1

Let state s=[v,w,k,y] be a reachable limited state in G. Since in the path from initial

state to s there can be at most m.n.(t+1) reachable limited states (otherwise, there is a state [v,w,k',y] in the path with $k'<k$ and by Algorithm 3 state s must be an abstract state [v,w,ω,y] which violates the assumption that s is a limited state.), $k \leq m.n.(t+1)$. Therefore there are at most $m^2.n^2.(t+1)^2$ reachable limited states. Based on a similar argument as above, the number of reachable abstract states is less than or equal to m.n.(t+1). Hence the abstract reachability graph of (M,N,m) is finite.

□

Proof of Theorem 4:

We show that if there is a livelock in the marked network (M,N,m) then there is a nonprogress cycle in the abstract reachability graph of (M,N,m). Assume that there is a livelock (C,D) in (M,N,m). Therefore, the following three assertions hold:

i. All the edges of cycle C ,in M, are nonprogress.

ii. All the edges of cycle D ,in N, are nonprogress.

iii. There exists a sequence $(s_1,...,s_r)$ of reachable states of network (M,N) such that the following two conditions hold:

   a. For i=1,...,r-1, state s_{i+1} follows s_i over an edge e_i in M or N. Also state s_1 follows s_r over an edge e_r in M or N.

   b. Each edge in $\{e_1,e_2,...,e_r\}$ is either in cycle C or cycle D.

Cycles C and D are not necessarily fundamental; hence, without loss of generality, we assume that they have an equal number of edges. Let q be the number of edges in C (or D), then q=r/2. Since the state s_1 is reachable and since the sequence $(s_1,...,s_r)$ can be executed infinitely, there exist two infinite paths α and β in M and N respectively that lead the network (M,N) into the sequence $(s_1,s_2,...,s_r)$ and repeat the sequece infinitely. These two paths α and β can be defined as follows:

$\alpha = a_1 a_2 \ldots\ldots\ldots\ldots a_n c_1 \ldots\ldots\ldots\ldots\ldots\ldots c_q c_1 \ldots\ldots\ldots\ldots\ldots\ldots c_q c_1 \ldots\ldots$
$\beta = b_1 b_2 \ldots b_m d_1 \ldots\ldots\ldots\ldots d_q d_1 \ldots\ldots\ldots\ldots\ldots\ldots d_q d_1 \ldots\ldots\ldots\ldots$

where a_i (i=1,...,n) are edges in machine M,
b_i (i=1,...,m) are edges in machine N,
c_i (i=1,...,q) are edges in cycle C,
d_i (i=1,...,q) are edges in cycle D, and
assume without loss generality that $n \geq m$.

Starting from the initial state, let machine N progress along path β as far as possible and we can find a reachable limited state $s_1=[v_0,w,k,E]$, where w is a receiving node since N cannot progress any further. Then let M progress along path β until it sends a message g to N, at that time we reach a limited state $s_2=[v,w,k',g]$ where $k' \leq k$. Then let M

progress as far as possible again, etc. It is straightforward to show that the above process can proceed forever along paths α and β. By repeating the above process, we can generate a sequence $s_i=[v_i,w_i,k_i,y_i]$, i=0,1,..., of reachable limited states.

Assume without loss of generality that $s_t=[v_t,w_t,k_t,y_t]$ be the first reachable limited state generated by the above process, and nodes v_t and w_t are in cycles C and D respectively. Since the messages sent out by M along cycle C can be received by N along cycle D, based on a similar argument as that in proof of Theorem 2, there exists a reachable limited state $s_{t+k}=[v_t,w_t,k_t,y_t]$. Therefore the limited reachable states $s_t,s_{t+1},\ldots,s_{t+k}$ form a cycle L. There are two cases:

i. *There exist two states s_i and s_j such that $v_i=v_j$, $w_i=w_j$, $k_i<k_j$, and $y_i=y_j$.*
By Algorithm 3, $s_j=[v_i,w_i,\omega,y_i]$ and there exists a cycle L' in G whose vertices are labelled with abstract states $s'_l=[v_l,w_l,\omega,y_l]$, l=t,...,t+k, and whose arcs are all labelled with nonprogress edges of cycles C or D. Hence L' is a nonprogress cycle in the abstract reachability graph of (M,N,m).

ii. *The other case.*
By Algorithm 3, all the states in set $S=\{s_i|\ i=0,1,\ldots,t+k\}$ are reachable limite states and there exists a cycle L' in G whose vertices are labelled with states in S and whose arcs are all labelled with nonprogress edges of cycles C or D. Hence L' is a nonprogress cycle in the abstract reachability graph of (M,N,m).

□

Proof of Theorem 5:

[If part]: We show that if there is a livelock in the marked network (M,N,m) then there is a nonprogress cycle in the closed cover graph of (M,N,m). Assume that there is a livelock (C,D) in (M,N,m). Therefore, the following three assertions hold:

i. All the edges of cycle C ,in M, are nonprogress.

ii. All the edges of cycle D ,in N, are nonprogress.

iii. There exists a sequence $(s_1,\ldots,s_r)$ of reachable states of network (M,N) such that the following two conditions hold:

   a. For i=1,...,r-1, state s_{i+1} follows s_i over an edge e_i in M or N. Also state s_1 follows s_r over an edge e_r in M or N.

   b. Each edge in $\{e_1,e_2,\ldots,e_r\}$ is either in cycle C or cycle D.

Cycles C and D are not necessarily fundamental; hence, without loss of generality, we assume that they have an equal number of edges. Let q be the number of edges in C (or

D), then q=r/2. Since the state s_1 is reachable and since the sequence $(s_1,...,s_r)$ can be executed infinitely, there exist two infinite paths α and β in M and N respectively that lead the network (M,N) into the sequence $(s_1,s_2,...,s_r)$ and repeat the sequece infinitely. These two paths α and β can be defined as follows:

$\alpha = a_1 a_2 \ldots\ldots a_n c_1 \ldots\ldots c_q c_1 \ldots\ldots c_q c_1 \ldots\ldots$
$\beta = b_1 b_2 \ldots b_m d_1 \ldots\ldots d_q d_1 \ldots\ldots d_q d_1 \ldots\ldots$

where a_i (i=1,...,n) are edges in machine M,
b_i (i=1,...,m) are edges in machine N,
c_i (i=1,...,q) are edges in cycle C,
d_i (i=1,...,q) are edges in cycle D, and
assume without loss generality that $n \geq m$.

Let AM (AN) be the acyclic machine of M (N) for the closed cover C. Since AM (AN) contains all the possible edge sequence in M (N) from its input version nodes to its output version nodes, there exists an edge sequence in AM (AN) that is a prefix of the edge sequence in path α (β). Starting from the initial state s_0, let machine M (N) progress along path α (β) (or for that matter along a path in AM (AN)) until it reaches an output version node in AM (AN). It is straightforward to show that the network reaches a state s_1 in the closed cover C. Starting from s_1, repeat the same process as above, the network will reach a sequence of states $s_2,s_3,...$

Based on a similar argument as that in theorem 4, there exist nonnegative integers t and k such that states $s_t,s_{t+1},...,s_{t+k}$ form a cycle L. By Algorithm 4, there exists a cycle L' in the closed cover graph of (M,N,m), whose vertices are labelled with states $s_t,s_{t+1},...,s_{t+k}$ and whose arc labels consist of the nonprogress edges of cycles C or D. Hence L' is a nonprogress cycle of the closed cover graph of (M,N,m).

[**Only If part**]: This part is straightforward since the states in the closed cover are reachable.

□

REFERENCES

1. Bartlett, K. A., R. A. Scantlebury, and P. T. Wilkinson, "A note on reliable full-duplex transmission over half-duplex links," *Comm. ACM*, Vol. 12, May 1969, pp.260-261.

2. Bochmann, G. V. "Finite state description of communication protocols," *Computer Networks*, Vol. 2, 1978, pp. 361-371.

3. Bochmann, G. V. and C. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. on Commun.*, Vol. COM-28, No. 4, April 1980, pp.624-631.

4. Bochmann, G. V. et al, "Experience with formal specifications using an extended state transition model," *IEEE Trans. on Commun.*, Vol. COM-30, No. 12, Dec. 1982, pp. 2506-2513.

5. Brand, D. and P. Zafiropulo, "On communicating finite-state machines," *JACM*, Vol. 30, No. 2, April 1983, pp. 323-342.

6. Chow, C. H., M. G. Gouda, and S. S. Lam, "An exercise in constructing multi-phase communication protocols," *Proc. of SIGCOMM'84 Symposium*, June 1984.

7. Day, J. D., "Terminal protocols," *IEEE Trans. on Commun.*, Vol. COM-28, No. 4, April 1980, pp. 585-593.

8. Duenki, A. and P. Schicker, "Symmetry and attention handling: Comments on a virtual terminal," *ACM Comput. Commun. Rev.*, Vol. 7, pp. 56-58, July 1977.

9. Gouda, M. G., "Closed covers: to verify progress for communicating finite state machines," *TR-191*, Dept. of Computer Sciences, Univ. of Texas at Austin, Jan. 1982. Revised Jan. 1983. To appear in *IEEE Trans. on Software Engineering.*

10. Gouda, M. G., "An example for constructing communicating machines by step-wise refinement," *Proc. 3rd IFIP Workshop on Protocol Specification, Testing, and Verification,* edited by H. Rudin and C. H. West, North-Holand, 1983, pp. 63-74.

11. Gouda, M. G., E. G. Manning, and Y. T. Yu, "On the progress of communication between two finite state machines," *TR-200*, Dept. of Computer Sciences, Univ. of Texas at Austin, May 1982. Revised Oct. 1983.

12. Gouda, M. G. and Y. T. Yu, "Protocol validation by maximal progress state exploration", *IEEE Trans. on Commun.* Vol. COM-32, No. 1, Jan. 1984, pp. 94-97.

13. Gouda, M. G. and C. K. Chang, "Proving liveness for networks of communicating finite state machines," *TR-84-04*, Dept. of Computer Sciences, Univ. of Texas at Austin, Feb. 1984.

14. Gouda, M. G. and Y. T. Yu, "Synthesis of communicating machines with guaranteed progress", To appear in *IEEE Trans. on Commun.*, July 1984.

15. Gouda, M. G. and L. E. Rosier, "Communicating finite state machines with priority channels," *Proc. of the Eleventh International Colloqium on Automata, Languages, and Programming (ICALP)*, 1984.

16. Gouda, M. G., C. H. Chow and S. S. Lam, "On the decidability of livelock detection in networks of communicating finite state machines," *Proc. 4th Int. Workshop on Protocol Specification, Testing and Verification,* June 11-14,

1984.

17. Hajek, J. "Automatically verified data transfer protocols," *Proc. of 4th International conference on computer communication,* Kyoto, Sept. 1978, pp. 749-756.

18. IFIP WG 6.1, "Proposal for a standard virtual terminal protocol," *Proc. Comput. Network Protocols Symp.*, Liege, Belgium, pp. H.27-H.49, Feb. 13-15, 1978.

19. Lai, W. S., "Protocol traps in computer networks -- a catalog," *IEEE Trans. on Commun.*, vol. COM-30, No. 6, pp. 1434-1450, June 1982.

20. Lam, S. S. and A. U. Shankar, "Protocol projections: a method for analyzing communication protocols," *Conf. Rec. National Telecommunications Conference*, Nov. 1981, New Orleans.

21. Lam, S. S. and A. U. Shankar, "Protocol verification via projections," To appear in *IEEE Trans. on Software Engineering*, July 1984.

22. Lam, S. S., "Data link control procedures," in *Computer Communications*, Vol. 1, ed. W. Chou, Prentice-Hall, Englewood Cliffs, 1983, pp. 81-113.

23. Merlin, P. M. and G. V. Bochmann, "On the construction of submodule specifications and communication protocols," *ACM TOPLAS*, Vol. 5, No. 1, Jan. 1983, pp. 1- 25.

24. Razouk, R. R. and G. Estrin, "Modeling and verification of communication protocols in SARA: The X.21 interface," *IEEE Trans. on Comput.*, Vol. C-29, No. 12, Dec. 1980, pp. 1038-1052.

25. Razouk, R., "Modeling X.25 using the graph model of behavior," *Proc. 2nd Int. Workshop on Protocol Specification, Testing and Verfication*, May 1982, Idyllwild, CA.

26. Rosier, L. E. and M. G. Gouda, "Deciding progress for a class of communicating finite state machines," *Proc. of Conference on Information Sciences and Systems,* Princeton University, 1984.

27. Rubin, J. and C. H. West, "An improved protocol validation technique," *Computer Networks*, April 1982.

28. Shankar, A. U. and S. S. Lam, "Specification and verification of an HDLC protocol with ARM connection management and full-duplex data transfer," *Proc. ACM SIGCOMM '83 Symposium*, March 1983, Univ. of Texas at Austin.

29. Shankar, A. U. and S. S. Lam, "An HDLC protocol specification and its verification using image protocols," *ACM Trans. on Computer Systems*, Vol. 1, No. 4, Nov. 1983, pp. 331-368.

30. Sherman, M. and H. Rudin, "Using automated validation techniques to detect lockups in packet-switched networks," *IEEE Trans. on Commun.*, Vol. COM-30, No. 7, July 1982, pp. 1762-1767.

31. West, C. H. and P. Zafiropulo, "Automated validation of a communications protocol: The CCITT X.21 recommendations," *IBM J. Res. Develop.*, Vol. 22, Jan. 1978, pp. 60-71.

32. Yu, Y. T. and M. G. Gouda, "Deadlock detection for a class of communicating finite state machines," *IEEE Trans. on Commun.*, Vol. COM-30, No. 12, Dec. 1982, pp. 2514-2519.

33. Yu, Y. T. and M. G. Gouda, "Unboundedness detection for a class of communicating finite-state machines," *Information Processing Letters*, Vol. 17, Dec. 1983, pp. 235-240.

34. Zafiropulo, P., West, C. H., Rudin, H., Cowan, D. D., and Brand, D., "Towards analyzing and synthesizing protocols," *IEEE Trans. on Commun.*, Vol. COM-28, No. 4, April 1980, pp. 651-661.