ON THE NOTION OF EQUIVALENCE FOR COMMUNICATING FINITE STATE MACHINES

M. G. Gouda and C. H. Youn

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

TR-84-14 May 1984

ABSTRACT

We present a new notion of equivalence for communicating finite state machines that exchange messages via one-to-one, unbounded, FIFO channels. This notion has a number of interesting properties: for example, it is decidable for any two communicating finite state machines M and N, whether M and N are equivalent. Moreover, if M and N are equivalent, then for any communicating finite state machine P, the communication between M and P can reach a nonprogress state iff the communication between N and P can reach a nonprogress state. We also show that this notion of equivalence applies not only to individual communicating finite state machines but also to networks of such machines.

I. INTRODUCTION

Many communication protocols can be modeled as networks of communicating finite state machines that exchange messages via one-to-one, unbounded, FIFO channels [3,7,9,10]. Each machine in such a network is defined as a directed graph where nodes represent the "machine states" and edges represent "state transitions." Each edge of a communicating finite state machine is labelled. The syntax and semantics of edge labels are discussed in Section II; we settle here for some informal explanation. An edge, labelled +g/C, from node v to node v' in machine M informally means that if the "head" message in channel C is g, and if M is currently at node v, then M can remove message g from C, and become at node v'. An edge, labelled -g/D, from node v to node v' in machine M informally means that if M is currently at node v, then M can add message g to the "tail" of the current contents of channel D, and become at node v'.

As an example, consider the two communicating finite state machines M and N in Figures 1a and 1b, respectively. Machine M starts from its initial node, node 1, by sending a Data message via channel A, then waits to receive either an Ack (for acknowledgement) or Nak (for negative acknowledgement) via channel B before returning to node 1. Machine N starts from its initial node, node 1, by sending a Data message via channel A; it then either waits to receive Ack via channel B, or waits to receive Nak via channel B. Now, let us consider the following question. Are M and N equivalent? Of course, the answer depends on what we mean by equivalent. If we view M and N as regular finite state machines, then the answer is "yes" since both machines accept the same regular language [8]. On the other hand, if machine P in Figure 1c is allowed to communicate with M and with N, then its communication with M is guaranteed to progress indefinitely, while its communication with N can reach the following nonprogress state: Machine N is at node 2, machine P is at node 1, channel A is empty, and channel B has one Nak message.

This example demonstrates that the notion of equivalence for regular finite state machines is not appropriate for communicating finite state machines. In this paper, we characterize a new notion of equivalence for communicating finite state machines that satisfies the following three properties:

i. It is decidable for any two communicating finite state machines M and N whether M and N are equivalent. (The same is true for regular finite state machines.)

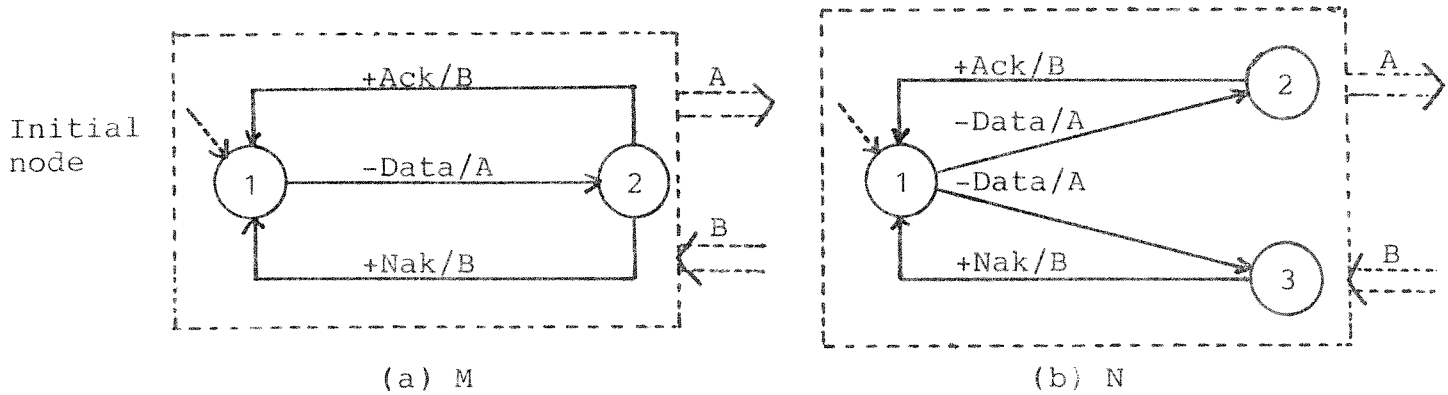
ii. If M and N are equivalent, then the regular language accepted by M is identical to that accepted by N. (The same is true for regular finite state machines.)

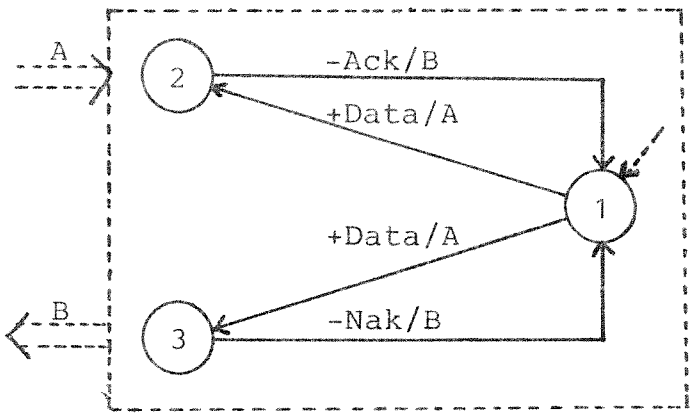
iii. If M and N are equivalent, then for any communicating finite state machine P, the communication between M and P can reach a nonprogress state iff the communication between N and P can reach a nonprogress state.

We also show that this notion of equivalence applies as well to networks of communicating finite state machines.

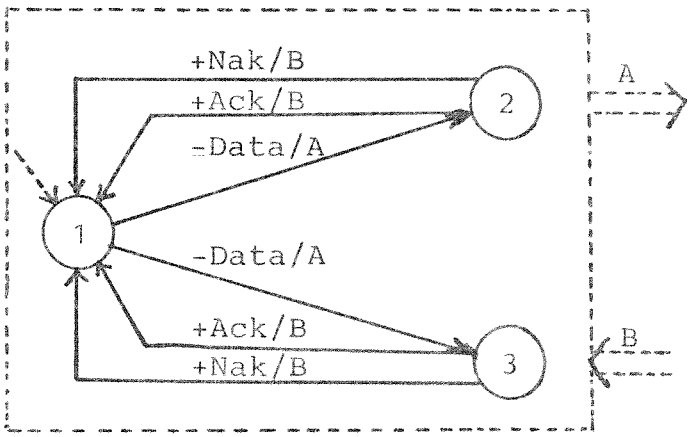
II. COMMUNICATING FINITE STATE MACHINES

A *Communicating Finite State Machine* M is a directed labelled graph with three types of edges called null, sending, and receiving edges. A *null edge* is labelled with the empty string E. A *receiving (sending) edge* is labelled +g/C (-g/D), where





(c) P



(d) Q

Figure 1. First Example.

g is a *message* from a finite set G_M of *messages*,
C is an *input channel* from a finite set I_M of *input chanels*,
D is an *output channel* from a finite set O_M of *output channels*, and
$I_M \cap O_M = \emptyset$ (the empty set).
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.

A communicating finite state machine is called *deterministic* iff it has no null edges, and the outgoing edges of each of its nodes have distinct labels. Otherwise, it is called nondeterministic.

Let M and N be two communcating finite state machines such that $G_M = G_N$, $I_M = O_N$, and $I_N = O_M$, then the pair [M,N] is called a *network* of the two machines M and N.

Let [M,N] be a network of two communicating finite state machines. A *state* of [M,N] is a four tuple [v,w,x,y], where
v is a node in M,
w is a node in N,
x is a string of symbols of the form g/C where g is in G_M, and C is in I_M, and
y is a string of symbols of the form g/D where g is in G_N, and D is in I_N.
Informally a state [v,w,x,y] indicates that the executions of the two machines M and N have reached nodes v and w respectively. It also indicates that the contents of the input channels of M are defind by string x, and the contents of the input channels of N are defined by string y.

The *initial state* of [M,N] is $[v_o,w_o,E,E]$, where v_o and w_o are the initial nodes of M and N respectively, and E denotes the empty string. (This implies that initially, all the channels between M and N are empty.)

Let s=[v,w,x,y] be a state of [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 six conditions is satisfied:

i. e is a null edge from v to v' in M, and s'=[v',w,x,y].

ii. e is a null edge from w to w' in N, and s'=[v,w',x,y].

iii. e is a receiving edge labelled +g/C from v to v' in M, and s'=[v',w,x'.x",y], where x=x'.g/C.x" and x' does not have any symbol of the form g'/C, for some message g'.

iv. e is a receiving edge labelled +g/D from w to w' in N, and s'=[v,w',x,y'.y"], where y=y'.g/D.y" and y' does not have any symbol of the form g'/D, for some message g'.

v. e is a sending edge labelled -g/D from v to v' in M, and s'=[v',w,x,y'], where y'=y.g/D.

vi. e is a sending edge labelled -g/C from w to w' in N, and s'=[v,w',x',y], where x'=x.g/C.

The next three definitions are based on the above definition of "follow over." Let s and s' be two states of [M,N].

i. s' *follows* s iff there is an edge e in M or N such that s' follows s over e.

ii. s' is *reachable from* s iff either 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.

iii. s' is *reachable* iff it is reachabl from the initial state of (M,N).

We have introduced a number of generalizations to this model of communicating finite state machines over previous models [2,11,12]. In particular, we allowed each machine to be nondeterministic, and allowed more than two channels between any two machines in a network. The reason for these generalizations is to make the definition of "equivalence," although primarily intended for machines, applicable to networks of machines as well. This will become apparent in Sections V and VI.

III. NONPROGRESS STATES

As mentioned earlier, the main reason for our dissatisfaction with the definition of equivalence for regular finite state machines as it applies to communicating finite state machines is that it does not capture the notion of "nonprogress." In this section, we formally characterize this notion.

A node in a communicating finite state machine M is called *final* iff it has no outgoing edges. A nonfinal node in M is called a *receiving node* iff all its outgoing edges are receiving. A nonfinal node in M is called an *escape node* iff it has at least one outgoing null edge and each of its outgoing edges is either null or receiving. A *trap* T in M is a set of escape nodes in M such that each outgoing null edge of any node in T must be ingoing to some (possibly the same) node in T.

Let v be a node in a communicating finite state machine M. The *expectancy* Exp(v) *of* node v is defined as follows:

i. If v is a final node, then
Exp(v)=$\emptyset$ (the empty set)

ii. If v is a receiving node, then
Exp(v)={g/C|there is an outgoing edge of node v labelled +g/C}

iii. If v is an escape node in a trap, then
Exp(v)={g/C|either there is an outgoing edge of node v labelled +g/C, or there is a directed path, whose edges are all null, from node v to some node with an outgoing edge labelled +g/C}

iv. Otherwise,
Exp(v) is undefined.

Let [v,w,x,y] be a state of a network [M,N]. A symbol g/C in string x is called the *head symbol of channel C in* x iff x is of the form x'.g/C.x", and x' does not have any symbol of the form g'/C for some message g'. Similarly, we can define the head symbol of some channel D is string y.

A reachable state [v,w,x,y] of network [M,N] is called a *nonprogress state* iff the following two conditions are satisfied:

i. The expectancy of node v in M is defined, and for any head symbol g/C in x, g/C is not in Exp(v).

ii. The expectancy of node w in N is defined, and for any head symbol g/D in y, g/D is not in Exp(v).

Informally, a nonprogress state of a network is reached when and only when each machine in the network either reaches a final node, or is "stuck" at a receiving node or in a trap waiting to receive a message g_i from channel C_i to get out, but each of the channels C_i is either empty or has a "head message" other than g_i. (Notice that if a machine is stuck in a receiving node, then it can no longer progress, and if it is stuck in a trap, then it can only progress over null edges.)

Nonprogress states can be distinguished into different classes such as proper termination states, improper termination states, deadlock states, unspecified reception states [7], and so on. This classification, however, is irrelevant to the current paper, and so we choose to ignore it.

Now, we are ready to present our notion of equivalence for communicating finite state machines.

IV. EQUIVALENCE OF COMMUNICATING FINITE STATE MACHINES

Let M be a communicating finite state machine, and let p be a finite directed path that starts from the initial node in M; i.e. p can be defined as a finite sequence $<e_1,...,e_r>$ of directed edges in M. The *word* w(p) *of path* p is a string of symbols $x_1. \ldots .x_r$, where x_i is the label of edge e_i, i=1,...,r, and "." is the concatenation operator.

Let v be a node in a communicating finite state machine M. The *language of node* v is defined as the set

$L(v)=\{w(p)|$p is a finite directed path that starts from the initial node and ends at node v in M.$\}$

The *language of machine* M is defined as the set

$L(M)=\{w(p)|$p is a finite directed path that starts from the initial node in M.$\}$

Clearly, L(v) and L(M) are regular languages, and so can be defined by regular expressions [8].

Let M and N be two communicating finite state machines, and let v be a node in M, and $w_1,...,w_r$ (for some $r\geq 1$) be some nodes in N. Node v is said to *cover* nodes $w_1,...,w_r$ iff the following two conditions are satisfied:

i. If Exp(v) is defined, then for all i ($1\leq i\leq r$) $Exp(w_i)$ is defined and is a subset of Exp(v).

ii. $L(v) \subseteq L(w_1) \cup ... \cup L(w_r)$, where L(v) is the language of node v, and "$\cup$" is the set-union operator.

Let M and N be two communicating finite state machines. M is said to *cover* N, denoted by $M \geq N$, iff for each node v in M, there exist nodes $w_1,...,w_r$ (for some $r \geq 1$) in N such that v covers $w_1,...,w_r$.

Two communicating finite state machines M and N are said to be *equivalent*, denoted by $M = N$, iff $M \geq N$ and $M \leq N$.

The following theorems prove some interesting properties for the above definition of "equivalence."

Theorem 1: It is decidable for any two communicating finite state machines M and N, whether $M = N$.

Proof: Let M and N be any two communicating finite state machines. It is decidable for any node v in M and any set of nodes $w_1,...,w_r$ in N whether v covers $w_1,...,w_r$. Since N has a finite number of nodes, then it is decidable for any node v in M whether there exist a set of nodes $w_1,...,w_r$ in N such that v covers $w_1,...,w_r$. Since M has a finite number of nodes, it is decidable whether $M \geq N$. By symmetry, it is also decidable whether $N \geq M$. Hence, it is decidable whether $M = N$. □

Theorem 2: If $M = N$, then $L(M) = L(N)$. (The converse is not necessarily true.)

Proof: To show that if $M = N$, then $L(M) = L(N)$, it is sufficient to show that if $M \geq N$, then $L(M) \subseteq L(N)$. Assume that $M \geq N$. Then for any node v in M, there exists a set of nodes $w_1,...,w_r$ in N such that

$$L(v) \subseteq L(w_1) \cup \ldots \cup L(w_r).$$

Therefore,

$$\bigcup_{v \text{ is in } M} L(v) \subseteq \bigcup_{w \text{ is in } N} L(w), \text{ and}$$

$$L(M) \subseteq L(N).$$

To show that the converse is not necessarily true, consider the two machines M and N in Figures 1a and 1b respectively. □

Theorem 3: If $M = N$, then for any communicating finite state machine P, network [M,P] can reach a nonprogress state [v,z,x,y], iff network [N,P] can reach a nonprogress state [w,z,x,y].

Proof: Assume that $M = N$, and let P be any communicating finite state machine such that the network [M,P] can reach a nonprogress state of the form s=[v,z,x,y]. We show that the network [N,P] can reach a nonprogress state of the form [w,z,x,y].

Since M covers N, node v must cover some nodes $w_1,...,w_r$ in N. Since [v,z,x,y] is a non-progress state, then Exp(v) must be defined, and so are $Exp(w_1),...,Exp(w_r)$. Let the paths that M and P traverse to reach state s be A and B, respectively. Since v covers $w_1,...,w_r$, there is a path

C that is identical to A, and that ends at some node w in N, where w is one of the nodes $w_1,...,w_r$. Path C in N can communicate with path B in P so that the network [N,P] reaches a state s'=[w,z,x,y]. Since s is a nonprogress state, and since Exp(w) $\subseteq$Exp(v), then s' is a nonprogress state. □

Example 1: To show that the two communicating finite state machines M and N in Figures 1a and 1b (respectively) are not equivalent, it is sufficient to show that at least one node in N does not cover any node in M. Since in machine M, Exp(1) is undefined, and Exp(2) = {Ack/B,Nak/B}, and since in machine N, Exp(2) = {Ack/B}, then node 2 in N does not cover any node in M; hence M and N are not equivalent. (Node 3 in N also does not cover any node in M.)

On the other hand, the two communicating finite state machines M and Q in Figures 1a and 1d, respectively, are equivalent based on the following facts:

Node 1 in M covers node 1 in Q.
Node 2 in M covers node 2 in Q.
Node 1 in Q covers node 1 in M.
Node 2 in Q covers node 2 in M.
Node 3 in Q covers node 2 in M.

□

V. EQUIVALENCE OF BOUNDED NETWORKS

In this section and the next, we show that our notion of equivalence can be applied to networks of communicating machines. But first, we need to generalize our definition of a network.

As defined earlier, a network [M,N] of two communicating finite state machines M and N can be described as "closed." This is because each output channel of M is an input channel of N, and vice versa. In other words, there are no channels for the external environment to communicate with or to "observe" M or N. This is a severe restriction; it is relaxed by introducing the concept of an "open" network, next.

Let M and N be two communicating finite state machines such that $G_M=G_N$, $I_M\cap I_N=\emptyset$, and $O_M\cap O_N=\emptyset$. Then, the pair (M,N) is called an *open network*. Any channel in $(I_M\cap O_N) \cup (I_N\cap O_M)$ is called an *internal channel* of the open network. Any channel in $I_M\cup O_M\cup I_N\cup O_N$ that is not an internal channel is called an *external channel* of the open network. Notice that any closed network is a special case of an open network where all the channels are internal. Therefore, the following definitions for open networks are generalizations of previous definitions for closed networks.

A *state* of an open network (M,N) is a four tuple [v,w,x,y], where

v is a node in M,
w is a node in N,
x is a string of symbols of the form g/C where g is in G_M, and C is an *internal channel* in I_M, and

y is a string of symbols of the form g/D where g is in G_N, and D is an *internal channel* in I_N.

From this definition, only the contents of internal channels are part of the open network state; the contents of external channels are not part of the state.

The *initial state* of (M,N) is $[v_o,w_o,E,E]$ where v_o and w_o are the initial nodes of M and N respectively, and E denotes the empty string.

Let s=[v,w,x,y] be a state of an open 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 six conditions is satisfied:

i. e is a null edge, or is a receiving edge labelled +g/C, where C is external, or is a sending edge labelled -g/D, where D is external, from node v to v' in M, and s'=[v',w,x,y]

ii. e is a null edge, or is a receiving edge labelled +g/D, where C is external, or is a sending edge labelled -g/D, where D is external, from node w to w' in N, and s'=[v,w',x,y]

iii. e is a receiving edge labelled +g/C, where C is internal, from node v to v' in M, and s'=[v,w',x,y'.y"], where y=y'.g/C.y" and y' does not have any symbol of the form g'/C for some message g'.

iv. e is a receiving edge labelled +g/C, where C is internal, from node w to w' in N, and s'=[v',w,x'.x",y], where x=x'.g/C.x" and x' does not have any symbol of the form g'/C for some message g'.

v. e is a sending edge labelled -g/D, where D is internal, from node v to v' in M, and s'=[v',w,x,y'], where y'=y.g/D.

vi. e is a sending edge labelled -g/D, where D is internal, from node w to w' in N, and s'=[v,w',x',y], where x'=x.g/D.

Based on this definition of "follow over," we can define the "reachable states" of an open network in the same way as they are defined for a closed network earlier.

An open network is said to be *bounded* iff the set of its reachable states is finite. Otherwise it is called *unbounded*.

Let (M,N) be a bounded open network. Define R(M,N) as the communicating finite state machine constructed from (M,N) using the following steps:

i. For each reachable state s of (M,N), add a node, also called s for convenience, to R(M,N).

ii. The node in R(M,N) that corresponds to the initial state of (M,N) is identified as the initial node of R(M,N).

iii. For any two reachable states s and s' of (M,N), if s' follows s over an edge e in M or N, then add a labelled directed edge r from node s to node s' in R(M,N). Moreover, the label of r depends on the label of e as follows:

   a. If e is labelled E, or is labelled +g/C where C is internal, or is labelled -g/D where D is internal, then r is labelled E.

b. If e is labelled +g/C where C is external, then r is labelled +g/C.

c. If e is labelled -g/D where D is external, then r is labelled -g/D.

Let (M,N) be a bounded network, and let P be a communicating finite state machine. Network (M,N) and machine P are said to be *equivalent* iff R(M,N)=P.

Let (M,N) and (P,Q) be two bounded networks. The two networks (M,N) and (P,Q) are said to be *equivalent* iff R(M,N)=R(P,Q).

Therefore, the notion of machine equivalence in the previous section can be used to establish the equivalence of bounded networks as well.

Example 2: (An Alternating Bit Protocol) Consider the network of four communicating finite state machines S, M, N, and K in Figure 2a. Machine S (Figure 2b) is a source that keeps on sending Data messages via channel A, while machine K is a sink (Figure 2c) that keeps on receiving Data messages via channel D. The "service" performed by the two machines M and N is to transmit the Data messages, without corruption, from channel A to channel D. This is not a simple task, since we assume further that any transmitted message between M and N may be corrupted by "external noise." M and N employ an Alternating-bit protocol with the following rules [1]:

i. M attaches one bit to each Data message it sends to N.

ii. Whenever N receives a Data message, it checks its check-sum. If it detects a corruption in the received message, it sends a negative acknowledgement to M; otherwise, it sends a positive acknowledgement to M.

iii. Whenever M receives a negative acknowledgement or a corrupted acknowledgement, it resends the "last" data message, with the same value for its attached bit.

iv. Whenever M receives a positive acknowledgement, it sends the "next" data message after assigning a different value for its attached bit (different from the "last" value of the attached bit).

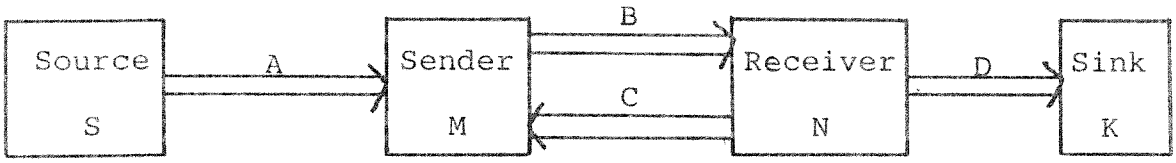
Machines M and N in Figures 2d and 2e (respectively) exchange the following messages:

Datai (i=0,1) is a Data message whose attached bit has the value i.

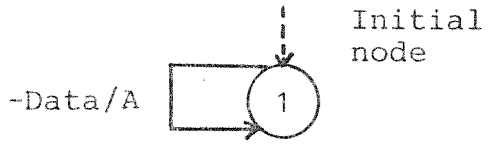
Acki (i=0,1) is a positive acknowledgement for Datai, and a negative acknowledgement for Data(i + 1 mod 2).

Err is a "virtual" message used to simulate a message corruption situation: The situation that M sends a Datai message which is later corrupted into Err before it is received by N is simulated by M directly sending an Err message to N.

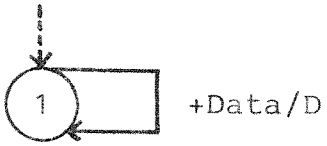
The pair (M,N) constitutes an open network where channels B and C are internal, and channels A and D are external. Thus, a reachable state of (M,N) is a four-tuple [v,w,x,y], where x defines the contents of channel C and y defines the contents of channel B. The communicating finite state machine R(M,N) is shown in Figure 2f. Each node in R(M,N) corresponds to a reachable state of (M,N); for example,



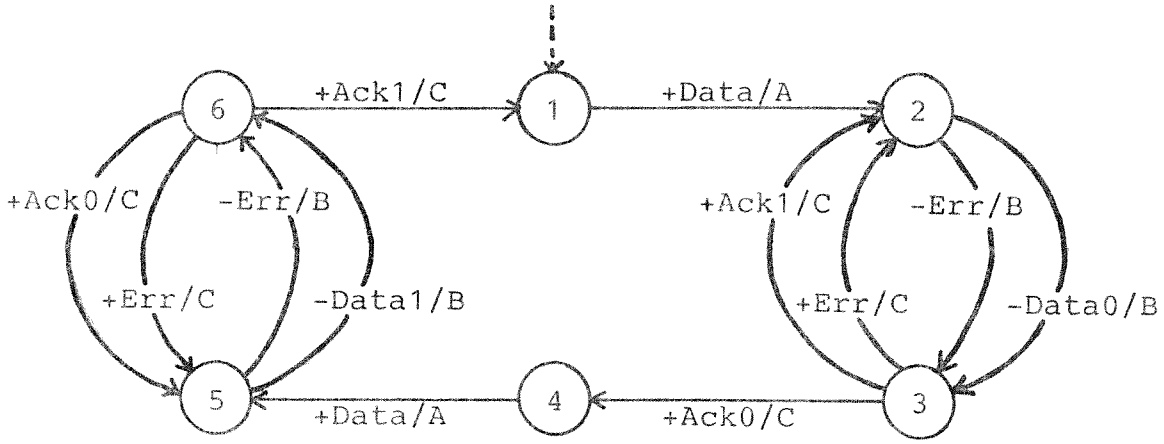
(a) Network



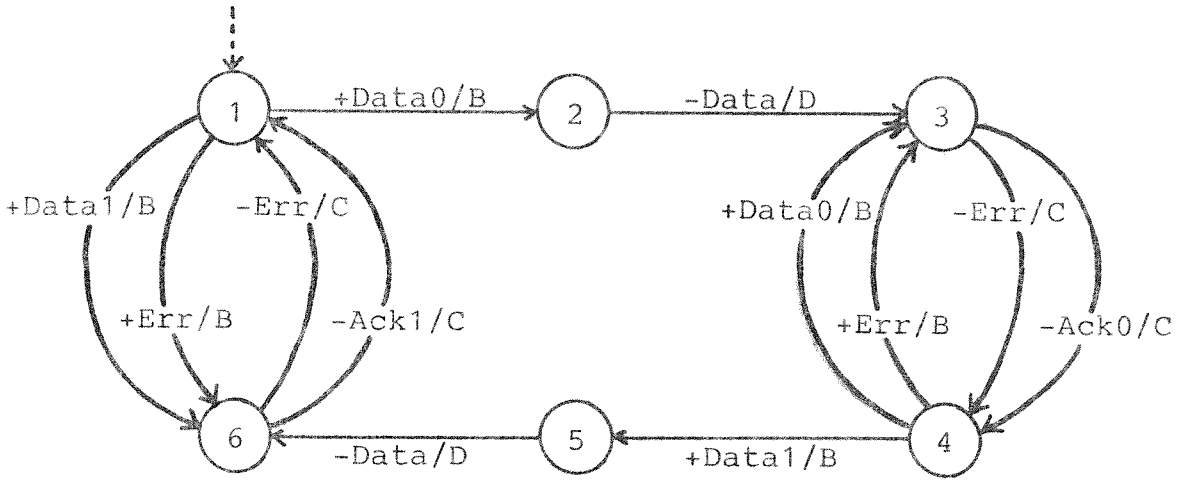
(b) S



(c) K



(d) M



(e) N

Figure 2. An Alternating-Bit protocol example.

node 1 in R(M,N) corresponds to state [1,1,E,E],
node 2 in R(M,N) corresponds to state [2,1,E,E], and
node 3 in R(M,N) corresponds to state [3,1,E,Err/B].

In order to show that R(M,N) is equivalent to machine L in Figure 2g, it is straightforward to show the following facts:

Node 1 in L covers node 1 (or 15) in R(M,N).
Node 2 in L covers node 8 (or 22) in R(M,N).
Each of the nodes 1, 9, 10, 11, 12, 13, 14, 15, 23, 24, 25, 26, 27, 28 in R(M,N) covers node 1 in L.
Each of the nodes 2, 3, 4, 5, 6, 7, 8, 16, 17, 18, 19, 20, 21, 22 in R(M,N) covers node 2 in L.

Three comments concerning this example are in order:

i. Although R(M,N) has many null cycles (i.e. cycles whose edges are all null), the equivalent machine L does not have any such cycle. This demonstrates that our definition of equivalence is implicitly based on the following *fairness* assumption: If a communicating finite state machine M reaches a null cycle that has a way out, then in a finite time M will choose the way out. (A null cycle with no way out in M is equivalent to a final node without outgoing edges. See Section VII.)

ii. Machine L in Figure 2g defines the "service" [4], as seen by an external observer, of the open network (M,N): L merely transmits the Data messages, one by one, from channel A to channel D.

iii. After showing that machine L is equivalent to the open network (M,N), it is possible to replace (M,N) by L during the analysis of the closed network [S,M,N,K]. In particular, to prove that [S,M,N,K] cannot reach a nonprogress state, it is sufficient to prove that [S,L,K] cannot reach a nonprogress state, which is a much simpler task. □

VI. EQUIVALENCE OF UNBOUNDED NETWORKS

The above definition of equivalence can be extended to communicating *infinite* state machines (i.e., communicating machines as defined earlier except that they have infinite numbers of nodes and edges, however each node has a finite number of ingoing and outgoing edges). This is an important extension since it allows us to define the notion of equivalence for unbounded open networks. For the sake of discussion in this section, let M and N be two communicating (finite or infinite) state machines.

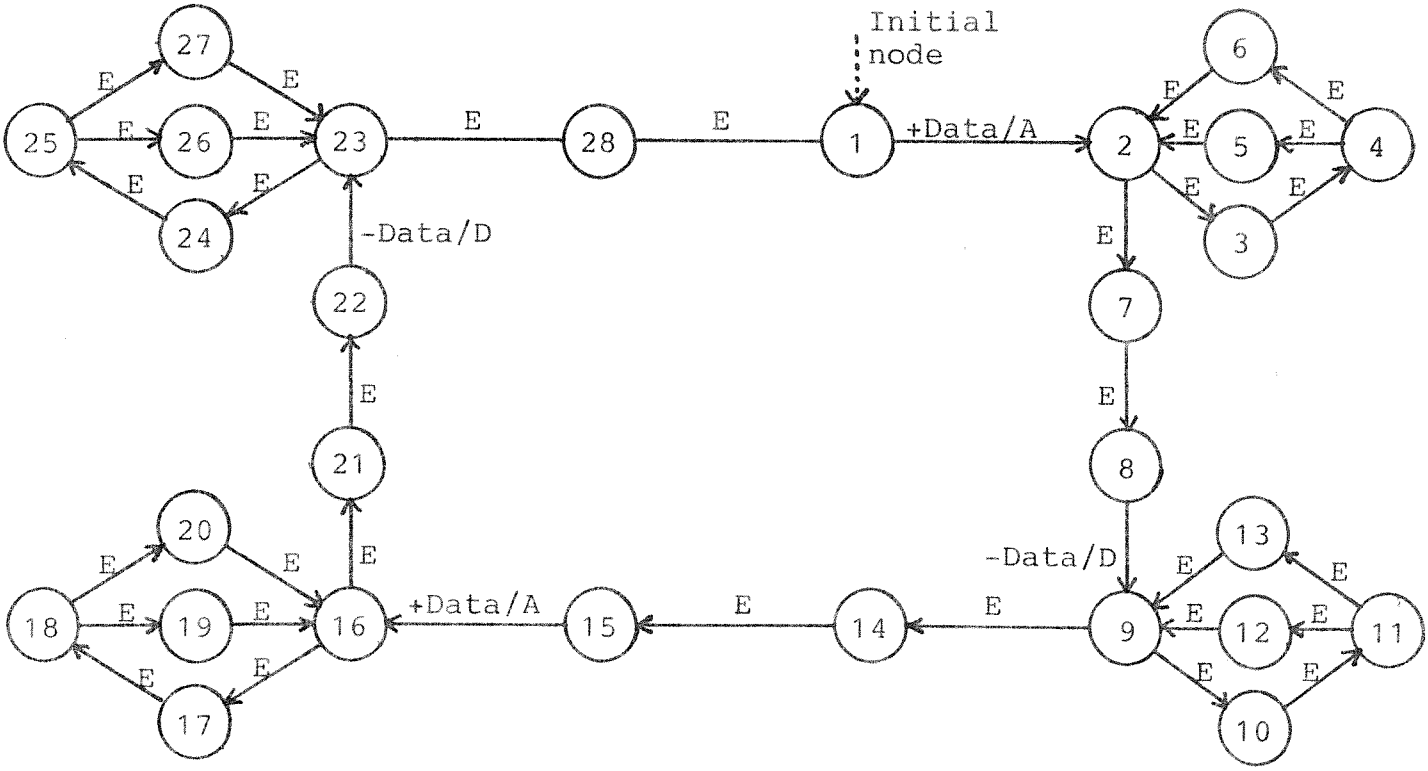
Let v be a node in M, and let $w_1, w_2, \ldots$ be (possibly an infinite number of) nodes in N. Node v is said to *cover* nodes $w_1, w_2, \ldots$ iff the following two conditions are satisfied:

i. If Exp(v) is defined, then for all i ($i \geq 1$), Exp(w_i) is defined and is a subset of Exp(v).

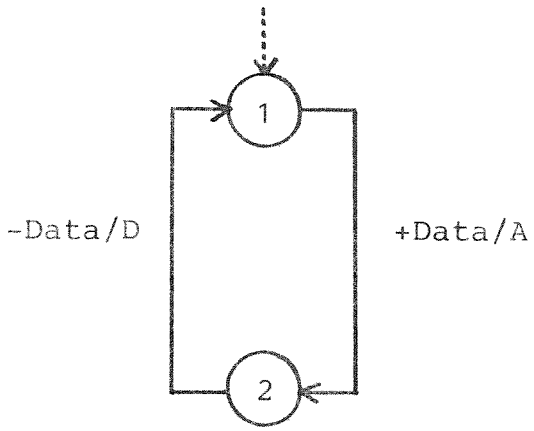
ii. $L(v) \subseteq \bigcup_{i=1}^{\infty} L(w_i)$.

M is said to *cover* N, denoted by $M \geq N$, iff for each node v in M, there exist (possibly an infinite number of) nodes $w_1, w_2, \ldots$ in N such that v covers $w_1, w_2, \ldots$.

M and N are said to be *equivalent*, denoted by $M = N$, iff $M \geq N$ and $M \leq N$.



(f) R(M,N)



(g) L

Figure 2. Continued.

With this extension to communicating infinite state machines, the decidability result of Theorem 1 is no longer valid. However, both Theorems 2 and 3 are still valid.

If (M,N) is a bounded (unbounded) open network, then R(M,N) is a communicating finite (infinite) state machine.

Let (M,N) be a (bounded or unbounded) open network, and let P and Q be two communicating (finite or infinite) state machines. Network (M,N) and machine P are said to be equivalent iff R(M,N)=P. Also, the two networks (M,N) and (P,Q) are said to be *equivalent* iff R(M,N)=R(P,Q).

Example 3: Consider the two communicating finite state machines K and L in Figure 2c and 2g respectively. The pair (K,L) is an open network with one internal channel D, and one external channel A. A reachable state of this network is a triple [v,w,x] where v is a node in K, w is a node in L, and x defines the contents of the internal channel D.

Network (K,L) is unbounded since it has an infinite number of reachable states. Hence, R(K,L) is a communicating infinite state machine. A portion of R(K,L) is shown in Figure 2h:

For i=0,1,2,...,

node m_i in R(K,L) corresponds to the reachable state [1,1,(Data/D)^i] of (K,L), and

node n_i in R(K,L) corresponds to the reachable state [2,1,(Data/D)^i] of (K,L).

R(K,L) is equivalent to the communicating finite state machine R in Figure 2i. This can be proved by showing the following:

Node 1 in R covers node m_0 in R(K,L).

For i=0,1,..., node m_i in R(K,L) covers node 1 in R.

For i=0,1,..., node n_i in R(K,L) covers node 1 in R.

Therefore, the unbounded network (K,L) is equivalent to machine R. □

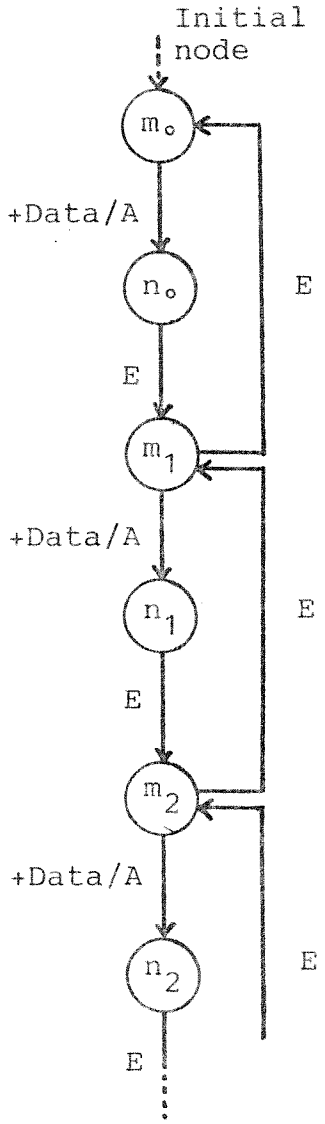
VII. NONDETERMINISM

As a side-effect to the above definition of equivalence, nondeterminism cannot always be removed from communicating finite state machines. For example, it is impossible to construct a deterministic communicating finite state machine that is equivalent to machine N in Figure 1b. (This is different from the corresponding phenomenon for regular finite state machines where nondeterminism can always be removed [8].)

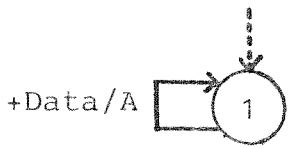
As mentioned earlier, nondeterminism can be expressed in a communicating finite state machine M using two mechanisms: Either M has a node with two or more identically-labelled outgoing edges, or M has a null edge. These two mechanisms are equivalent in the following sense:

i. For any communicating finite state machine M, it is possible to construct a communicating finite state machine M' such that M=M', and the outgoing edges of each node in M' have distinct labels.

ii. For any communicting finite state machine M, it is possible to construct a communicating finite state machine M' such that M=M', and M' has no null edges.



(h) R(K,L)



(i) R

Figure 2. Continued.

The proof for assertion i is straightforward. Assertion ii is proved by presenting two transformations that can be applied repeatedly to any communicting finite state machine M yielding, after a finite number of applications, a machine that is equivalent to M, and has no null edges.

The first transformation is illustrated in Figure 3a, where M and M' denote the machines before and after the transformation, respectively. This transformation consists of removing all the null self-loops at a node u. It is straightforward to show that node u in M covers node u' in M', and vice versa; hence M==M'.

The second transformation (illustrated in Figure 3b) consists of the following steps:

a. Remove a null edge from one node u to another v.

b. For each edge from some node a_i to u, add an edge with an identical label from a_i to v.

c. For each edge from v to some node d_l, add an edge with an identical label from u to d_l.

To show that this transformation from M to M' is equivalence-preserving (i.e. M==M'), it is straightforward to show the following:

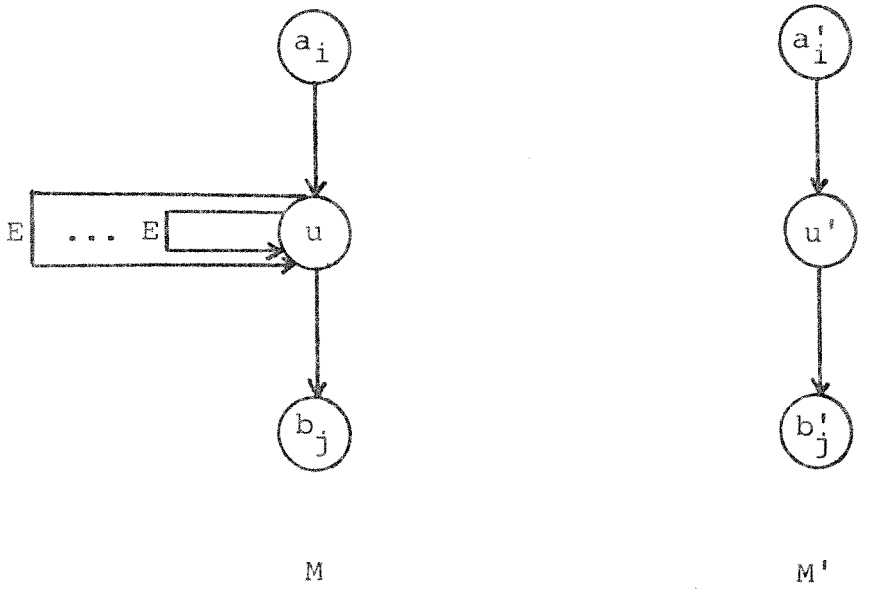
Node a_i in M covers node a_i' in M', and vice versa.
Node b_j in M covers node b_j' in M', and vice versa.
Node c_k in M covers node c_k' in M', and vice versa.
Node d_l in M covers node d_l' in M', and vice versa.
Node u in M covers node v' in M'.
Node v' in M' covers node v in M.
Node v in M covers node v' in M'.
Node u' in M' covers node v in M.

VIII. CONCLUDING REMARKS

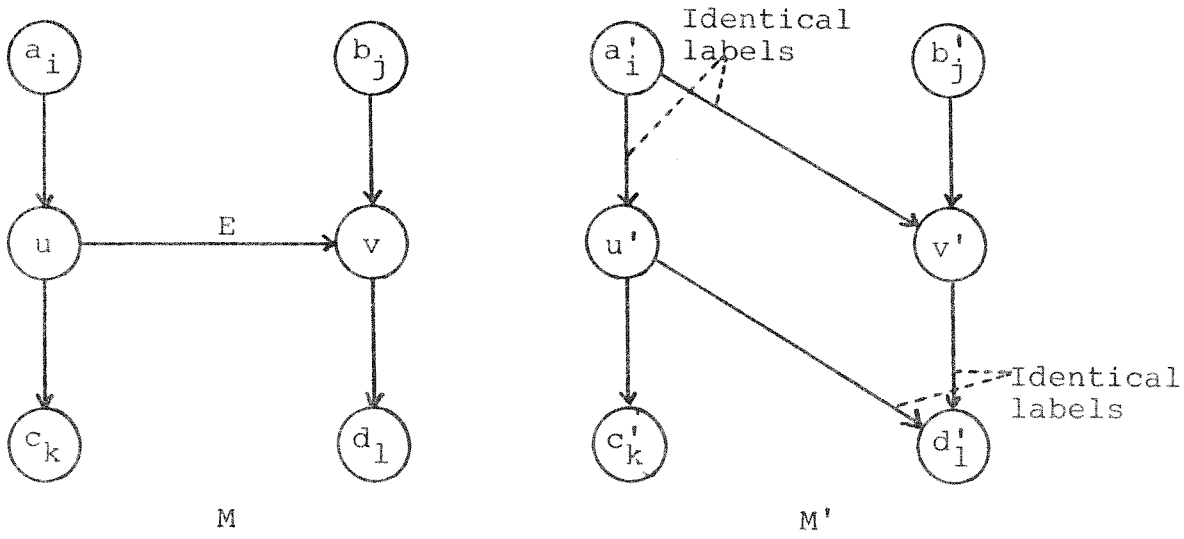
We have presented a new notion of equivalence for communicating finite state machines that exchange messages via FIFO channels, and shown that this notion satisfies some desirable properties. This notion is "robust," as it applies to finite and infinite machines, and to bounded and unbounded networks with any number of machines. This notion can also be applied (with minor modifications) to communicting finite state machines that exchange messages via priority or random channels [5,6].

This notion of equivalence for communicating finite state machines is useful in two respects:

i. It can be used to simplify the task of proving freedom of nonprogress (e.g. freedom of deadlocks) for "large" networks of communicating finite state machines. This can be done by repeatedly replacing some open (sub)networks within the network with equivalent simple communicting finite state machines, and showing that the resulting reduced network cannot reach nonprogress states. We have demonstrated this technique with the Alternating-bit protocol example, where the network [S,M,N,K] is reduced to [S,R] whose analysis is straightforward.



(a)



(b)

Figure 3. Two equivalence-preserving transformations.

ii. The existence of an "equivalence" criterion for communicating finite state machines, encourages one to search for "cost" criteria for such machines to discriminate between equivalent alternatives. One possible criterion is the number of nodes in the communicating machines, the less the number of nodes in a machine the better. (For example, based on this criterion machine R in Figure 2i is infinitely better than the equivalent machine R(K,L) in Figure 2h.) This point needs (and deserves) further research.

REFERENCES

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

[2] Brand, D. and P. Zafiropulo, "On communicating finite-state machines," Journal ACM, Vol. 30, No. 2, April 1983, pp. 323-342.

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

[4] Bochmann, G. V. and C. Sunshine, "Formal methods in communication protocol design," IEEE Trans. on Commun., April 1980, pp. 624-631.

[5] Gouda, M. G. and L. E. Rosier, "Communicating finite state machines with priority channels," in Proc. of the Eleventh International Colloquium on Automata, Languages, and Programming (ICALP), 1984.

[6] Gouda, M. G. and L. E. Rosier, "Priority networks of communicating finite state machines," to appear in SIAM Journal on Computing, Sept. 1984.

[7] Gouda, M. G. and Y. T. Yu, "Synthesis of communicating finite state machines with guaranteed progress," TR-179, Dept. of Computer Sciences, Univ. of Texas at Austin, June 1981. Revised Jan. 1983; revised Oct. 1983. To appear in IEEE Trans. on Commun., July 1984.

[8] Hopcroft, J. E. and J. D. Ullman, "Formal languages and their relation to automata," Addison-Wesley, Reading, MA, 1969.

[9] West, C. H. and P. Zafiropulo, "Automated validation of a communications protocol: the CCITT X.21 recommendation," IBM J. Res. Develop., Vol. 22, Jan. 1978, pp. 60-71.

[10] Yu, Y. T. and M. G. Gouda, "Deadlock detection for a class of communicating finite state machines," IEEE Trans. on Commun., Dec. 1982, pp. 2514-2519.

[11] 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.

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