PROTOCOL VALIDATION BY

MAXIMAL PROGRESS STATE EXPLORATION

M.G.Gouda and Y.T.Yu

Dept. of Computer Sciences
University of Texas at Austin
Austin, TX 78712

TR-211 July 1982

Table of Contents

ABSTRACT

Given two machines which communicate by exchanging messages over two finite-capacity channels, it is possible to generate all reachable states of the system and check whether any of them is a nonprogress state. This technique is called state exploration; and it usually requires large execution time and storage. In this paper, we discuss a more efficient variation of this technique. In particular, we show that the task of generating all reachable states can be divided into two independent subtasks. In each subtask, only the states reachable by allowing maximal progress for one machine are generated. We prove that a given system cannot reach a nonprogress state iff none of the states generated in each subtask is a nonprogress state. Since the two subtasks are completely independent, and since in most cases the time and storage requirements for each subtask are less than those for the origianal task, maximal progress state exploration can save time or storage over conventional state exploration. We also show that maximal progress state exploration can be used to check properties, other than progress, for the communicating machines; e.g., detecting all executable (and so all nonexecutable) transitions in the two machines.

I. INTRODUCTION

Many communication protocols can be modeled as two finite state machines that communicate by exchanging messages over two one-directional, FIFO channels [2,6,10,12]. Examples of such protocols are X.75 [7], and the call establishment/clear protocol in X.21 [9], and X.25 [1]. If the two channels in any such protocol are assumed to have finite capacities, then the protocol can be validated by generating all reachable states and checking whether any of them is a nonprogress state. This technique is referred to as state exploration. Some automatic protocol validation systems based on state exploration have been developed, and applied with reported success on actual protocols [3,5,8].

A major problem with state exploration is that it requires large execution time and storage. The problem is caused by the following assumption upon which state exploration is usually based. In order to generate all reachable states of a protocol, one needs to consider all possible progress speeds for the two machines in the protocol's model. This assumption leads to generating many "unimportant" states; it also leads to generating the same state many times since the same state can be reached by many different progress speeds for the two machines.

To counter this problem, Rubin and West [4] have shown that if the two machines progress in equal speeds, then the resulting reachable states can be used to detect deadlocks and unspecified receptions (but not necessarily overflows). This suggests to modify state exploration such that only states reachable by assuming equal progress speeds for the two machines are generated and examined. This modified state exploration requires, in most cases, less execution time and storage than conventional state exploration. (For example, the idea of restricting the two machines to equal progress speeds has led to

an efficient algorithm to detect deadlocks in the case where the two machines exchange one type of message [11].) On the other hand, this technique cannot be used to detect nonprogress since nonprogress can be caused by overflows as well as by deadlocks and unspecified receptions.

In this paper, we discuss another variation of state exploration where the state generation task is divided into two independent subtasks. In each subtask, only states reachable by allowing one of the two machines to make maximal progress are generated and examined. We show that in this case the generated states can be used to detect deadlocks, unspecified receptions, and overflows; thus they can be used to detect nonprogress, if any. We also show that this technique can be used to verify properties, other than progress, such as detecting all nonexecutable transitions in each machine.

The paper is organized as follows. The model of communicating machines is discussed in section II. Then in section III, the concept of legal sequences, upon which conventional state exploration is based, is presented. In section IV, a special class of legal sequences called maximal progress sequences are characterized; and we prove that these sequences can be used to detect deadlocks, unspecified receptions, and overflows. In section V, we discuss how to perform maximal progress state exploration (which is based on maximal progress sequences) to detect nonprogress states, if any. In section VI, we prove that maximal progress state exploration can be also used to detect all nonexecutable transitions. And in section VII, we discuss how to perform maximal progress state exploration on an extended model of communicating machines. Concluding remarks are in section VIII.

II. COMMUNICATING MACHINES

A communicating machine M is a directed labelled graph with two types of nodes called sending and receiving nodes. An output of a sending (or receiving) node is called a sending (or receiving) edge, and is labelled send(g) (or receive(g)) for some message g in a finite set G of messages. Each node in M must have at least one output edge; and outputs of the same node must have distinct labels. 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.

For simplicity, this model is a special case of the one in [2], where no node is allowed to have both sending and receiving outputs. Later in section VII, we extend the discussion to communicating machines with mixed nodes (i.e., nodes that have both sending and receiving outputs).

Let M and N be two comunicating machines with the same set G of messages. A state of M and 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 of messages from the set G such that $|x| \leq K$ and $|y| \leq K$, where $|x|$ (or $|y|$) is the length of string x (or y, respectively), and K is a positive integer called the channel capacity between M and N.

Informally, a state [v,w,x,y] means that the execution of machine M has reached node v and the execution of N has reached node w, while the input channels of M and N have the message sequences x and y respectively. The above definition also implies that each of the two channels between M and N has a finite capacity of K messages.

The initial state of M and 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 is the empty string.

Let s=[v,w,x,y] be a state of M and N, and let e be an output edge of node v or w. A state s´ is said to follow s over e, denoted s--e-->s´, iff the

following four conditions are satisfied:

i. If e is a sending edge, labelled send(g), from v to v' in M, then $|y|<K$ and $s'=[v',w,x,y.g]$, where "." is the concatenation operator.

ii. If e is a sending edge, labelled send(g), from w to w' in N, then $|x|<K$ and $s'=[v,w',x.g,y]$.

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

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

Let s and s' be two states of M and N, s' follows s, denoted s-->s', iff there is a directed edge e in M or N such that s--e-->s'. If s-->s' then s' is called a follower of s.

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

A state s of M and N is said to be reachable iff it is reachable from the initial state of M and N.

Since the two channels between M and N have finite capacities, the set of all reachable states is finite; and so usual state exploration techniques [3,8] can be used to generate all reachable states and check whether any of them is a nonprogress state. There are three types of nonprogress states, namely deadlock states, unspecified reception states, and overflow states. These are defined next:

A state [v,w,x,y] is a deadlock state iff v and w are receiving nodes, and x=y=E (the empty string).

A state [v,w,x,y] is an unspecified reception state iff one of the following two conditions holds.

i. $x=g_1 \cdot g_2 \cdot \ldots \cdot g_r$ and v is a receiving node and none of its outputs is labelled receive(g_1).

ii. $y=g_1 \cdot g_2 \cdot \ldots \cdot g_r$ and w is a receiving node and none of its outputs is labelled receive(g_1).

If condition i holds, then the state is called an unspecified reception state for M; if condition ii holds, it is called an unspecified reception state for N.

A state [v,w,x,y] of M and N is an overflow state iff one of the following two conditions holds.

i. Node v is a sending node; and $|y|=K$.

ii. Node w is a sending node; and $|x|=K$.

If condition i holds, then the state is called an overflow state for M; if condition ii holds, it is called an overflow state for N.

A state is called a nonprogress state iff it is a deadlock state, an unspecified reception state, or an overflow state; otherwise it is a progress state.

In the above discussion, a state s of M and N is reachable if there exist states $s_0,\ldots,s_r$ such that s_0 is the initial state of M and N, $s=s_r$, and $s_i \text{--}e_{i+1}\text{-->}s_{i+1}$ for some edge e_{i+1} in M or N, $i=0,\ldots,r-1$. One can view that state s is reachable by the "sequence" $\langle e_1,\ldots,e_r\rangle$ of edges. It is more convenient to replace each edge e_i in this sequence by a "symbol" q_i as follows:

If e_i is labelled send(g) in M, then q_i=´Msend(g)´.

If e_i is labelled send(g) in N, then q_i=´Nsend(g)´.

If e_i is labelled receive(g) in M, then q_i=´Mreceive(g)´.

If e_i is labelled receive(g) in N, then q_i=´Nreceive(g)´.

Therefore, s is reachable by the sequence of symbols $Q=q_1 \cdot q_2 \cdot \ldots \cdot q_r$, where "." is the concatenation operator. The concept of a sequence of symbols

which can reach a state is central to the discussion in this paper; and it is characterized formally in the next section.

III. SEQUENCES AND LEGAL SEQUENCES

Let M and N be two communicating machines; and assume that each of the two channels between M and N has a finite capacity of K. A sequence $Q=q_1 \cdot q_2 \cdot \ldots \cdot q_r$ of M and N is a finite string of symbols which satisfies the following two conditions:

i. Each symbol q_i in Q has one of the following four forms: ´Msend(g)´, ´Mreceive(g)´, ´Nsend(g)´, ´Nreceive(g)´, where g is some message in the set of messages of M and N. A symbol of the form ´Msend(g)´ or ´Mreceive(g)´ is called an M symbol; similarly, a symbol of the form ´Nsend(g)´ or ´Nreceive(g)´ is called an N symbol.

ii. There are two directed paths D_1 and D_2 which start from the initial nodes in M and N respectively such that the following two conditions are satisfied for each i=1,...,r.

   a. The ith edge in D_1 is labelled send(g) (or receive(g)) iff the ith M symbol in Q is Msend(g) (or Mreceive(g), respectively).

   b. The ith edge in D_2 is labelled send(g) (or receive(g)) iff the ith N symbol in Q is Nsend(g) (or Nreceive(g), respectively).

   Path D_1 (or D_2) is called the projection of Q onto M (or N), and can be referred to as Q_M (or Q_N, respectively).

Informally, a sequence Q defines a relative progress for the execution of M and N with respect to one another. So, if a symbol $Msend(g_1)$ occurs before a symbol $Nsend(g_2)$ in a sequence Q, then Q defines a relative progress where machine M sends a message g_1 before N sends a message g_2. Because of the dependency between the execution of M and that of N, not every relative progress is possible. In other words, not every sequence that can be defined

can be executed; for instance, a sequence where Mreceive(g) occurs before Nsend(g) cannot be executed. The class of sequences which can be executed is called legal sequences and is defined next.

A sequence $Q=q_1 \cdot \ldots \cdot q_r$ of M and N is called legal if it satisfies the following two conditions.

i. For any i=1,...,r, if the ith Mreceive symbol (or Nreceive symbol) in Q is Mreceive(g) (or Nreceive(g)), then the ith Nsend symbol (or Msend symbol) in Q must be Nsend(g) (or Msend(g)), and it must occur before the ith Mreceive symbol (or Nreceive symbol) in Q.

ii. For every prefix P of Q (i.e., $P=q_1 \cdot \ldots \cdot q_s$ where $s \leq r$), $n_{MN}(P) \leq K$ and $n_{NM}(P) \leq K$, where

$n_{MN}(P)$= The number of Msend symbols in P
- The number of Nreceive symbols in P,

$n_{NM}(P)$= The number of Nsend symbols in P
- The number of Mreceive symbols in P.

Condition i means that every message should be received after it is sent and in the same order it is sent. Thus, condition i implies that no deadlock state can be reached before completing Q. Condition ii means that as the two machines M and N execute according to the relative progress defined by Q, the two channels between M and N do not overflow. The concept of a state which is reached at the end of a legal sequence is defined next.

Let Q be a legal sequence of M and N. Q is said to reach a state [v,w,x,y] of M and N iff the following two conditions are satisfied

i. Nodes v and w are the last nodes in pathes Q_M and Q_N, respectively. Recall that Q_M and Q_N are the projections of Q onto M and N, respectively.

ii. String x (or y) consists of all the messages sent along Q_N (or Q_M) minus those received along Q_M (or Q_N, respectively).

If Q reaches a state s, then s is said to be reachable by Q.

The next three lemmas follow immediately from the definitions of rea chable state," "sequence," "legal sequence," and "reachable by."

Lemma 1: A state of M and N is reachable iff it is reachable by a legal sequence of M and N. []

Lemma 2: A prefix of a sequence (or a legal sequence) of M and N is a sequence (or a legal sequence, respectively) of M and N. []

Lemma 3: Let P be a proper prefix of a legal sequence of M and N. Then, the state reachable by P is not a deadlock state. []

Based on the concept of legal sequences of M and N, usual state exploration for M and N can be explained as the following three-step procedure:

i. Generate the legal sequences of M and N, one by one, in the order of their lengths. Start by the shortest sequence; then generate longer and longer sequences. (The shortest legal sequence is the empty string; it reaches the initial state of M and N.)

ii. For each generated legal sequence Q, construct the state s reachable by Q, and check whether or not s is a nonprogress state.

iii. Repeat steps i and ii until one of the following two conditions holds.

   a. A nonprogress state is detected, in which case machines M and N can reach a nonprogress state.

   b. No nonprogress state has been detected and no new states can be constructed any more, in which case M and N cannot reach a nonprogress state.

In the next section, we characterize a special class of legal sequences called maximal progress sequences; then in section V, we discuss how to perform state exploration based on this special class of sequences.

IV. MAXIMAL PROGRESS SEQUENCES

Let M and N be two machines which communicate over finite-capacity channels; and let Q be a legal sequence of M and N. Q is called a maximal progress sequence for M iff it satisfies the following three conditions:

i. $Q = A_1 \cdot B_1 \cdot A_2 \cdot B_2 \cdot \ \cdots \ \cdot A_n \cdot B_n$, where

$n \geq 1$,
A_i (i=1,...,n) is a string of M symbols,
B_i (i=1,...,n) is a string of N symbols, and
A_1 and B_n can be empty strings.

ii. For i=1,...,n, the sequence $A_1 \cdot B_1 \cdot \ \cdots \ \cdot A_i$ must reach a state [v,w,x,y] where v is a receiving node and x=E (the empty string).

iii. For i=1,...,n, the sequence $A_1 \cdot B_1 \cdot \ \cdots \ \cdot B_i$ must reach a state [v,w,x,y] where $x \neq E$, and the largest proper prefix of $A_1 \cdot B_1 \cdot \ \cdots \ \cdot B_i$ must reach a state $[v',w',x',y']$ where $x'=E$.

Informally, a maximal progress sequence for M defines the following relative order of execution for M and N: First, M executes as much as possible; i.e., until it reaches a receiving node while its input channel is empty. Then, N executes until it sends exactly one message to M; i.e., M can now resume execution. Then, M executes as much as possible, and so on.

The next four theorems, whose proofs are in the Appendix, state roughly that if a nonprogress state is reachable, then a nonprogress state (not necessarily the same one) is reachable by a maximal progress sequence. Notice that The converse is also true by lemma 1.

Theorem 1: If a deadlock state is reachable (by a legal sequence), then either a deadlock or an overflow state is reachable by a maximal progress sequence for M. []

Theorem 2: If an unspecified reception state for M is reachable (by a legal

sequence), then either an unspecified reception state for M or an overflow state is reachable by a maximal progress sequence for M. []

Theorem 3: If an unspecified reception state for N is reachable (by a legal sequence), then either an unspecified reception state or an overflow state is reachable by a maximal progress sequence for M. []

Theorem 4: If an overflow state for M is reachable (by a legal sequence), then an overflow state is reachable by a Maximal progress sequence for M. []

From these four theorems, if every state reachable by a maximal progress sequence for M is a progress state, then every reachable state is neither a deadlock state, an unspecified reception state, nor an overflow state for M. This does not guarantee that every reachable state is a progress state since some reachable states can still be overflow states for N. Therefore, to ensure that every reachable state is a progress state, it is sufficient (and necessary) to ensure that every state reachable by a maximal progress sequence for M or N is a progress state. The next theorem follows immediately.

Theorem 5: A nonprogress state is reachable (by a legal sequence) iff a nonprogress state is reachable by a maximal progress sequence for M or N. []

V. EFFICIENT STATE EXPLORATION

Based on Theorem 5, the following algorithm uses maximal progress state exploration to decide whether any two communicating machines with finite-capacity channels can reach a nonprogress state.

Algorithm 1:

Input: Two communicating machines M and N which communicate over two finite-capacity channels.

Output: A decision of whether M and N can reach a nonprogress state.

Steps:

i. Use Algorithm 2 (discussed below) to generate each state reachable by a maximal progress sequence for M. If any of these states is a nonprogress state, then stop: M and N can reach a nonprogress state.

ii. Use an algorithm similar to Algorithm 2 to generate each state reachable by a maximal progress sequence for N. If any of these states is a nonprogress state, then stop: M and N can reach a nonprogress state.

iii. If all generated states in steps i and ii are progress states, then stop: M and N cannot reach a nonprogress state. []

Now it remains to describe an algorithm to generate and examine all states reachable by maximal progress sequences for M. (An algorithm to generate and examine all states reachable by maximal progress sequences for N is similar.)

Algorithm 2:

Input: Two communicating machines M and N which communicate over two finite-capacity channels.

Output: A decision of whether any state reachable by a maximal progress sequence for M is a nonprogress state.

Variables: Two sets of states called OPEN and CLOSE. OPEN contains all the generated states which have not been examined yet; and CLOSE contains the states which have been generated and examined earlier. Initially, OPEN has the initial state of M and N and CLOSE is empty. The algorithm terminates when OPEN becomes empty or when one of the generated states is recognized to be a nonprogress state.

Steps: while OPEN is not empty do

i. remove one state s, selected at random, from OPEN.

ii. if s is a nonprogress state then stop: There is a nonprogress state reachable by a maximal progress sequence for M.

iii. if s is in CLOSE
then skip
else add s to CLOSE;
if s=[v,w,x,y], where v is a receiving node
and x=E

then a. generate all states s´ such that
s--e-->s´, and
e is an edge in N
b. add all the generated states s´ to OPEN.

else a. generate all states s´ such that
s--e-->s´, and
e is an edge in M.
b. add all the generated states s´ to OPEN.

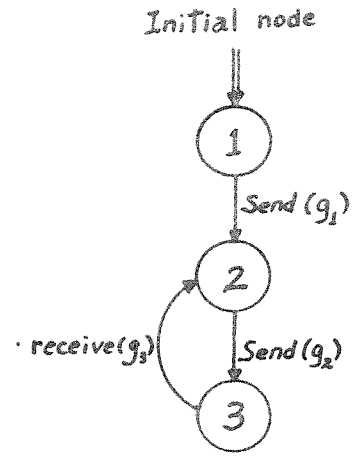
endwhile;
if OPEN is empty (at the end of the while-statement)
then stop:
Each state reachable by a maximal progress sequence for M is a progress state. []

From step iii, not all followers of a reachable state s are generated. In particular, from any state s=[v,w,x,y], either machine M or N, but not both, can progress. If v is a receiving node and x=E (i.e., machine M cannot progress), then only machine N can progress; otherwise, only machine M can progress. Since from each state, at most one machine can progress, the number of states generated by Algorithm 2 are, in most instances, less than the number of states generated by conventional state exploration. (It is possible, however, to construct an M and N such that from each reachable state of M and N, at most one machine can make a progress. In such cases, the number of states generated by Algorithm 2 equals the number of states generated by conventional state exploration.)

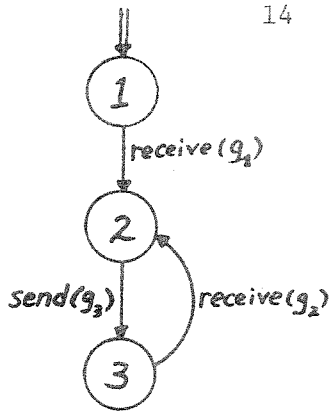
Example 1: Figures 1a and 1b show two communicating machines M and N respectively. Assume that M and N communicate over two channels, each with a capacity of two. Figure 1c shows the states of M and N reachable by maximal progress sequences for M; i.e., those generated by Algorithm 2. The states reachable by maximal progress sequences for N are shown in Figure 1d. Since each of the states in Figures 1c and 1d is a progress state, M and N cannot reach a nonprogress state by Theorem 5.

Let us compare this maximal progress state exploration with conventional state exploration. Figure 1e shows all the reachable states of M and N generated by conventional state exploration. There is a total of 16 states in Figures 1c and 1d, compared with 17 states in figure 1e. (Notice that we have counted repeated states since it takes the same amount of effort to generate and check a repeated state as it is to generate a new state.) From this example, maximal progress state exploration can reduce the total number of generated states. Nevertheless, its real advantage lies in dividing the space of reachable states into two independent (but not necessarily disjoint) subspaces, namely the states reachable by maximal progress sequences for M, and those reachable by maximal progress sequences for N. As discussed next, this can be exploited to reduce the execution time or storage, required to perform state exploration.

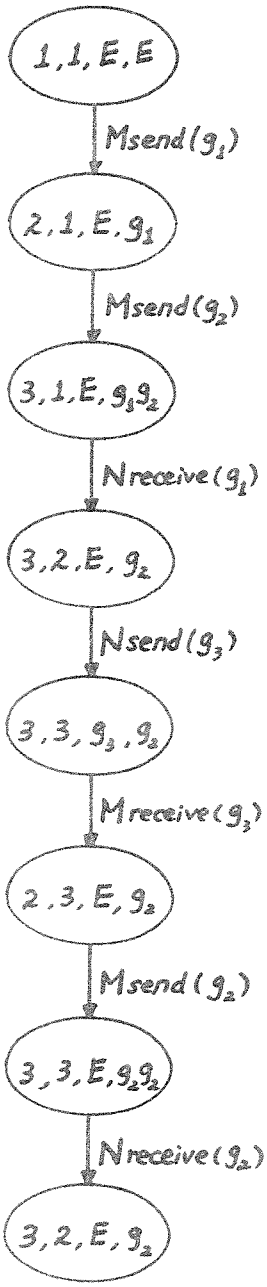
Reduction in execution time can be achieved by generating the states reachable by maximal progress sequences for M <u>in parallel</u> with those reachable by maximal progress sequences for N. For instance, if the states in Figure 1c are generated in parallel with those in FigurE 1d, then the execution time will be reduced by a factor of two. Obviously, two processors, instead of one, are required to execute this parallel state exploration.



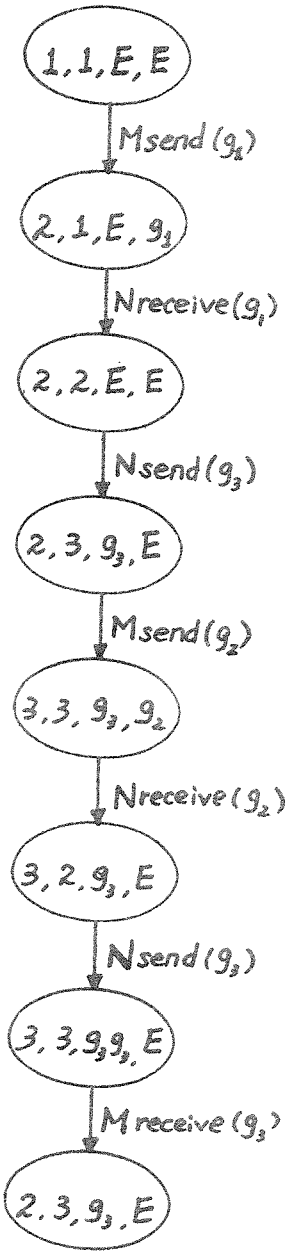
(a) M



(b) N

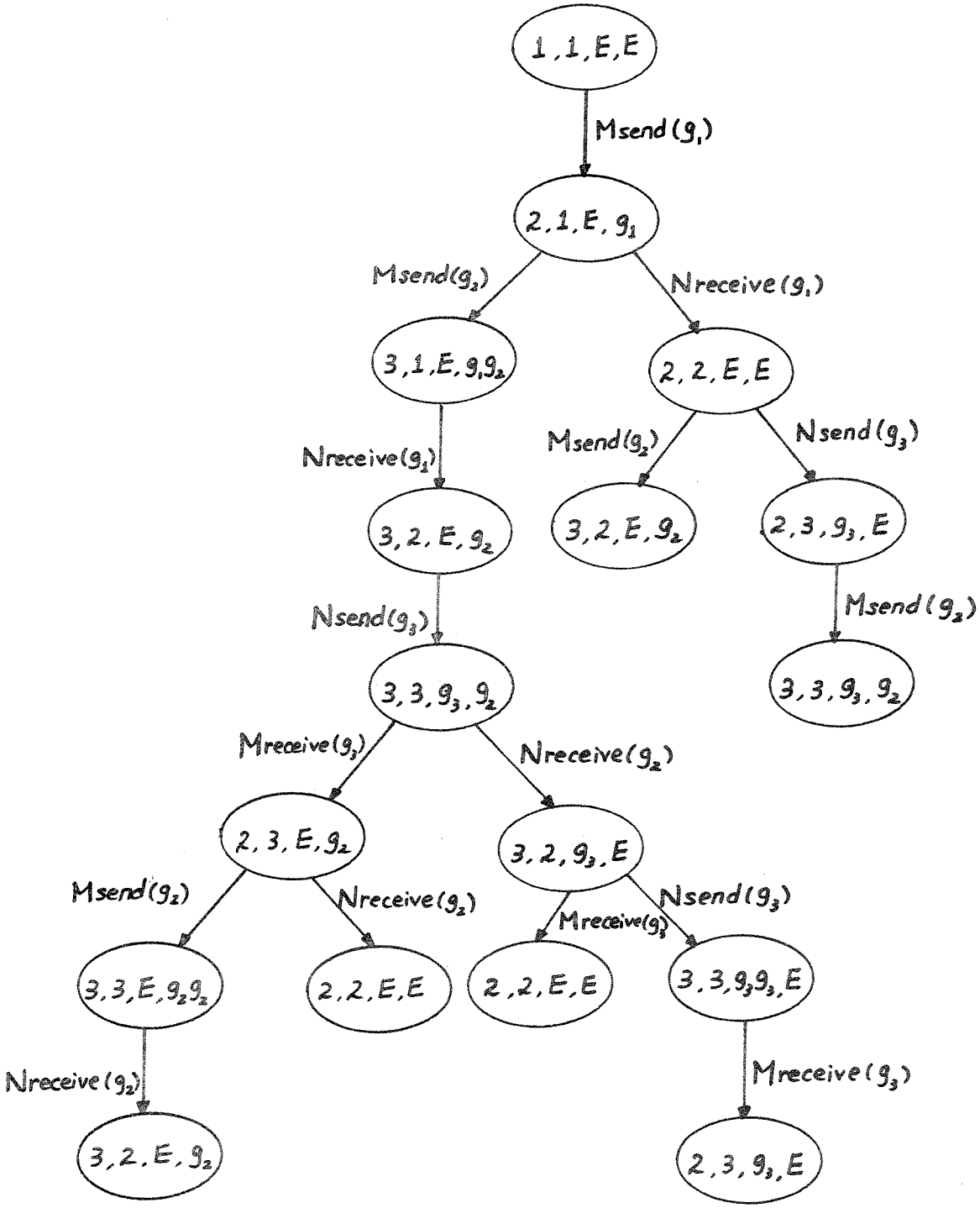


(c) State exploration by Maximal progress for M



(d) State exploration by maximal progress for N

Figure 1. First Example



(e) Conventional state exploration

Figure 1 Continues

During state exploration, the distinct states generated earlier are stored in a set (e.g.,set CLOSE in Algorithm 2) so that they can be compared with each state generated later. Since the number of distinct states reachable by maximal progress sequences for M (or N) is usually less than the number of distinct states reachable by legal sequences, the storage requirement of maximal progress state exploration is usually less than that of conventional state exploration. For exemple, a set of seven distinct states is needed to perform the maximal progress state exploration in Figures 1c and 1d provided that they are performed in sequence, one after another. This is better than the set of eleven distinct states needed to perform the conventional state exploration of Figure 1e. []

In summary, maximal progress state exploration reduces the state exploration task into two independent subtasks. In most instances, each subtask requires less execution time and storage than the original task. In these instances, if the two subtasks are executed in parallel, using two processors instesd of one, then the execution time of the task is reduced. And if the two subtasks are executed in sequence on the same processor, then the storage requirement of the task is reduced.

VI. EXAMINING OTHER PROPERTIES

So far, we have discussed how to use maximal progress state exploration to examine the indefinite progress of two communicating machines. After establishing the progress of two machines, it is possible to use the same maximal progress state exploration to examine other properties as well. On the other hand, there are some properties which cannot be examined by maximal progress state exploration at all. In this section, we first prove that all executable edges (and so all nonexecutable edges) in the two machines can be identified by maximal progress state exploration; then, we show that not all stable states can be identified by maximal progress state exploration.

A. Executable Edges:

Let M and N be two machines which communicate over finite-capacity channels; and assume that each state of M and N is a progress state. An edge e with a tail node v in M is said to be executable iff the following two conditions are satisfied.

i. If e is a sending edge, then a state of the form [v,w,x,y] is reachable.

ii. If e is a receiving edge labelled receive(g), then a state of the form [v,w,g.x,y] is reachable.

Conditions i and ii imply that the execution of M can progress to node v (the tail node of edge e); then it can progress along e to the next node in M. Notice that the definition of an executable (or nonexecutable) edge applies only if the two machines M and N can progress indefinitely.

Let e be an edge with a tail node v in M; and let Q be a maximal progress sequence for M. Edge e is said to be executable by sequence Q iff the following two conditions are satisfied.

i. If e is a sending edge, then Q reaches a state [v,w,x,y].

ii. If e is a receiving edge labelled receive(g), then Q reaches a state [v,w,g.x,y].

From the definitions of "executable" and "executable by", the following lemma is immediate.

Lemma 4: Let e be an edge in M. If e is executable by some maximal progress sequence for M, then e is executable. []

The converse of this lemma is also true; it is stated in the following theorem whose proof is in the Appendix.

Theorem 6: Let e be an edge in M. If e is executable, then it is executable by some maximal progress sequence for M. []

From Theorem 6, to establish that an edge in M (or N) is executable, it is sufficient to establish that it is executable by a maximal progres sequence for M (or N, respectively). For example, the receiving edge from node 3 to node 2 in machine M of Figure 1a is executable since it is executable by the following maximal progress sequence for M (see Figure 1c): $Msend(g_1).Msend(g_2).Nreceive(g_1).Nsend(g_3)$ which reaches state $[3,3,g_3,g_2]$. Similarly, the sending edge from node 2 to node 3 in machine N of Figure 1b is executable, by the following maximal progress sequence for N (see Figure 1d): $Msend(g_1).Nreceive(g_1)$ which reaches state [2,2,E,E]. Likewise, it can be shown that each edge in M or N of Figure 1 is executable.

Assume that a receiving edge labelled receive(g_4) is added from node 3 to node 2 in machine M of Figure 1a. The maximal progress sequences for M and N remain as they are in Figures 1c and 1d respectively. Then, the added edge is nonexecutable since it is nonexecutable by any maximal progress sequence for M.

B. Stable States

Let M and N be two machines which communicate over finite capacity channels. A state [v,w,x,y] of M and N is called stable iff x=y=E (the empty string).

We show by an example that maximal progress state exploration does not necessarily identify all stable states of two communicating machines. Consider the two machines M and N in Figures 2a and 2b respectively. From Figure 2e, M and N can reach the stable state [2,3,E,E]; but from Figures 2c and 2d, this state cannot be reached by any maximal progress sequence for M or N. Therefore, not all stable states can be identified by maximal progress state exploration.

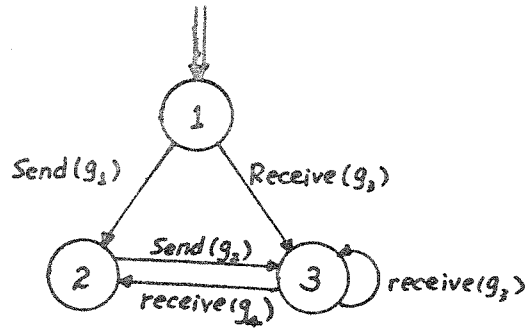
VII. EXTENSION TO MACHINES WITH MIXED NODES

The discussion so far has been limited to a model of communicating machines where no node is allowed to have both sending and receiving outputs. Since most "real" protocols are defined by communicating machines with such mixed nodes, it is useful to extend our discussion to communicating machines with mixed nodes.

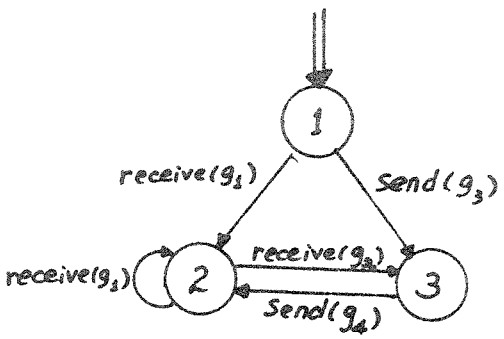
Let M and N be two communicating machines (possibly with some mixed nodes); and assume that the two channels between M and N have finite capacities. To decide whether M and N can reach a nonprogress state, Algorithm 1 for maximal progress state exploration can still be used with one modification, namely Algorithm 3 (discussed below) should be used instead of Algorithm 2. Algorithm 3 is a generalization of Algorithm 2; it generates and examines all the states reachable by maximal progress sequences for M taking into account the possibility that M and N may have some mixed nodes.

Algorithm 3:

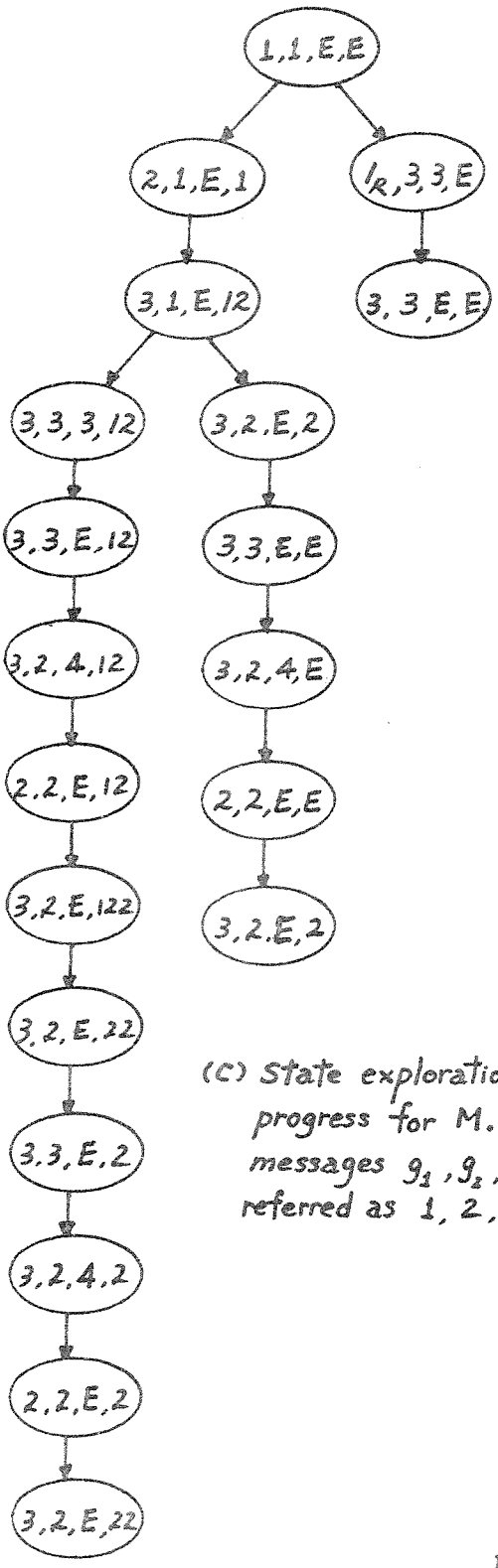
Input: Two communicating machines M and N, possibly with some mixed nodes,



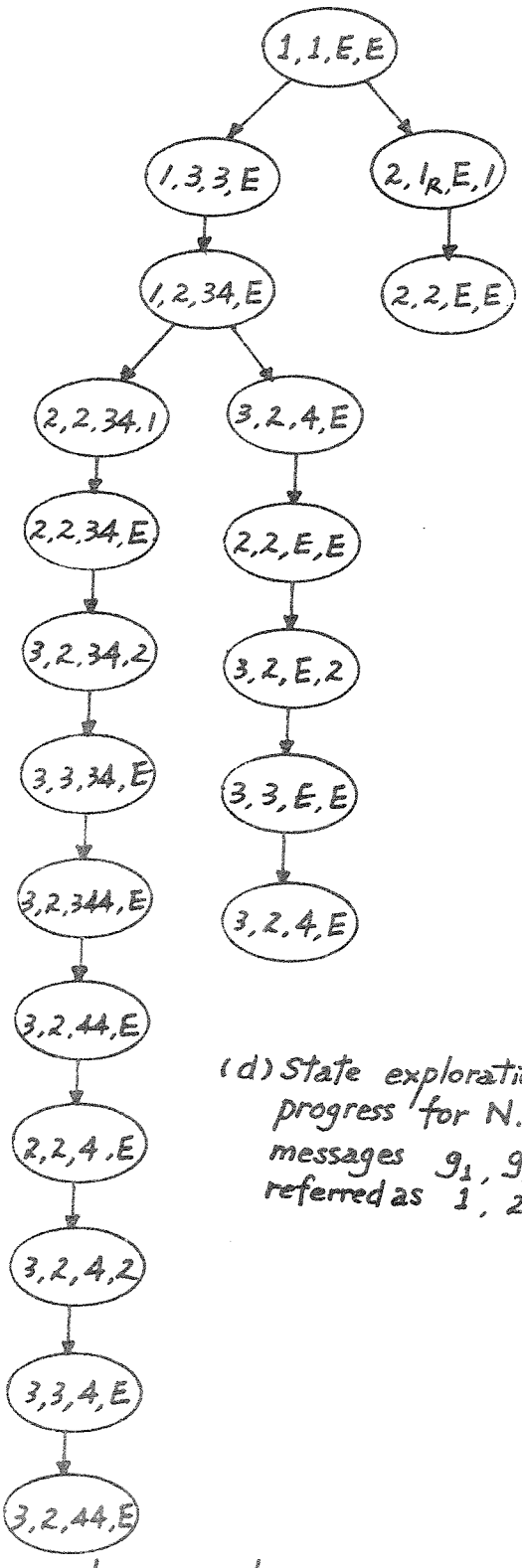
(a) M



(b) N

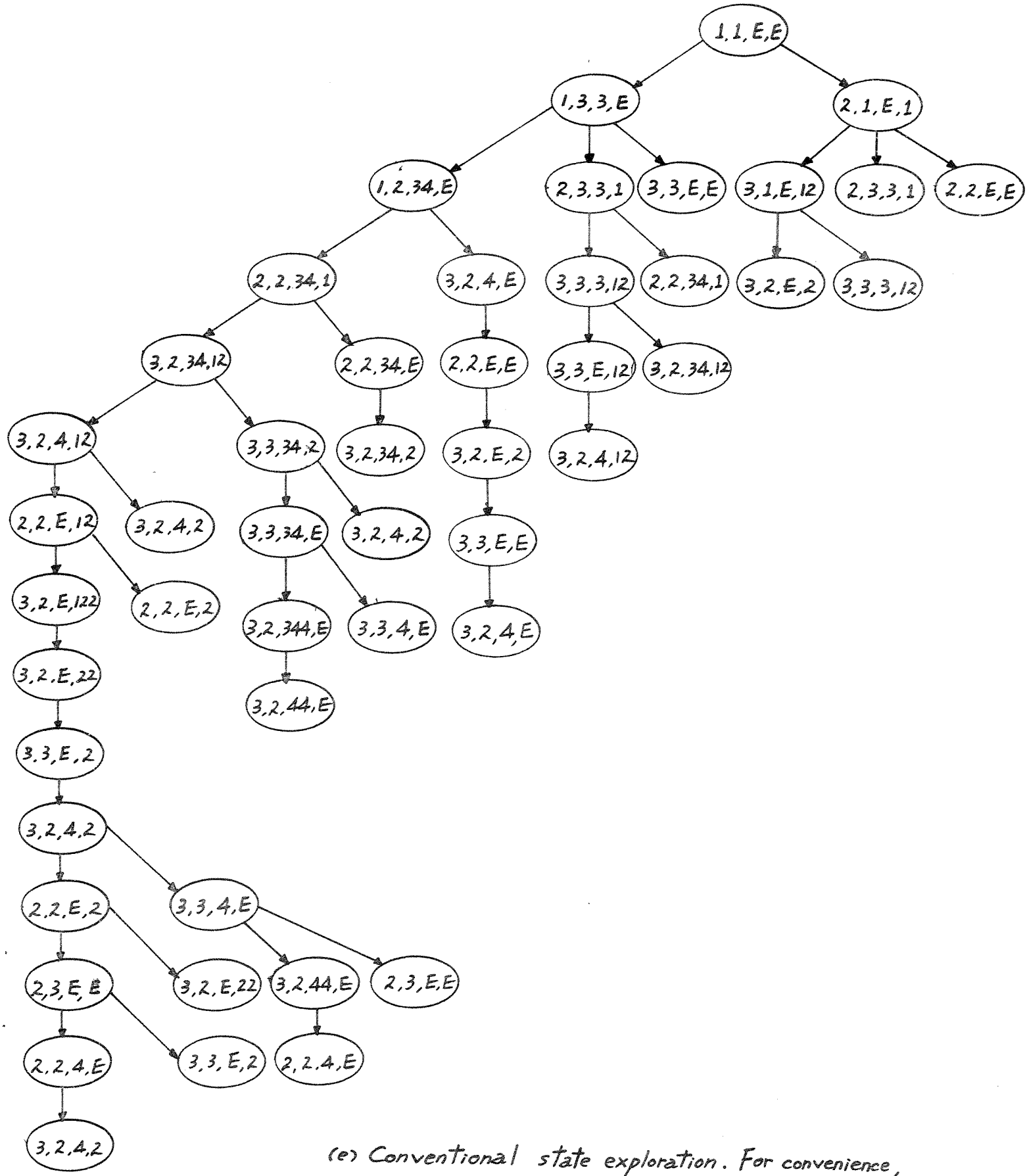


(c) State exploration by maximal progress for M. For convenience, messages $g_1, g_2, \cdots$ are referred as 1, 2, ⋯.



(d) State exploration by maximal progress for N. For convenien messages $g_1, g_2, \cdots$ are referred as 1, 2, ⋯.

Figure 2 Second example



(e) Conventional state exploration. For convenience, messages $g_1, g_2, \cdots$ are referred as $1, 2, \cdots$.

Figure 2 (cont

which communicate over two finite-capacity channels.

Output: A decision of whether any state reachable by a maximal progress sequence for M is a nonprogress state.

Variables: Two sets of states called OPEN and CLOSE. Initially, OPEN has the initial state of M and N and CLOSE is empty.

Steps: while OPEN is not empty do

i. remove one state s, selected at random, from OPEN.

ii. if s is a nonprogress state then stop: There is a nonprogress state reachable by a maximal progress sequence for M

iii. if s is in CLOSE
then skip
else add s to CLOSE;
case s=[v,w,x,y] of

a. v is a sending node or
(v is a receiving or mixed node and x≠E):
--generate all states s´ such that
s--e-->s´, and
e is in machine M
--add all generated s´ to OPEN

b. v is a receiving node and x=E:
-- generate all states s´ such that
s--e-->s´, and
e is in machine N
-- add all generated s´ to OPEN.

c. v is a mixed node and x=E:
-- generate all states s´ such that
s--e-->s´, and
e is in machine M or N.
-- for each generated s´
do let s--e-->s´
if e is in M
then add s´ to OPEN
elsif {* e is in M and s´=[v,w´,x´,w´] where
v is the same mixed node v in s *}
s´ is in OPEN or CLOSE
then skip
else write s´ as [v_R,w´,x´,y´] and
add it to OPEN
{* v_R is called a receiving mixed node.
Dealing with states in which the first
component is a receiving mixed node is
discussed in the next three cases.*}

```
  d. v is a receiving mixed node and x≠E:
        -- generate all states s´ such that
             s--e-->s´, and
             e is a receiving edge in M
           {* None of the generated states has a receiving
             mixed node in its first component*}
        -- add all generated s´ to OPEN.

  e. v is a receiving mixed node and x=E and
     (w is a receiving node and y=E):
        -- skip.

  f. v is a receiving mixed node and x=E and
     (w is not a receiving node or y≠E):
        -- generate all states s´ such that
             s--e-->s´, and
             e is an edge in N
           {* Each generated s´ has the same receiving mixed node
             in its first component as s*}
        -- for each generated s´
           do let s´=[v´R,w´,x´,y´]
              if [v´,w´,x´,y´] is in OPEN or CLOSE
              then skip
              else add s´ to OPEN
   endcase
endwhile;
if OPEN is empty (at the end of the while-statement)
  then stop : Each state reachable by a maximal progress
    sequence for M is a progress state.
```

[]

Example 2: Consider the two communicating machines M and N in Figures 2a and 2b respectively; notice that nodes 1 in M and N are mixed nodes. Assuming that each channel between the two machines has a capacity of three, it is required to decide whether or not M and N can reach a nonprogress state. This can be decided by maximal progress state exploration as well as by conventional state exploration; both techniques are compared next.

Figure 2c shows all the states reachable by maximal progress sequences for M (i.e., those generated by Algorithm 3). One of these states is $[1_R,3,3,E]$, where 1_R is a receiving mixed node. Similarly, Figure 2d shows all the states reachable by maximal progress sequences for N. Since each state in Figure 2c or 2d is a progress state, M and N cannot reach a nonprogress state. This same result can be obtained by conventional state exploration whose generated states are shown in Figure 2e.

The total number of states in Figures 2c and 2d is 40 which is slightly better than the 49 states generated in Figure 2e. But as discussed earlier, the real advantage of maximal progress state exploration is in dividing the state exploration task into two independent subtasks that can be executed in parallel to save time or executed in sequence to save storage. Since the number of states in Figure 2c (or 2d) is 20, compared with 49 states in figure 2e, then performing the two maximal progress state explorations in parallel can reduce the execution time by a factor of 2.5. Also, since the number of distinct states in Figure 2c (or 2d) is 17, compared with 29 distinct states in Figure 2e, then performing the two maximal progress state explorations in sequence can reduce the required storage by a factor of 1.5. []

Example 3: Consider the two communicating machines M and N in Figures 3a and 3b respectively. They represent the call establishment/clear protocol in x.25 [1]; in particular, M models the DTE and N models the DCE and the exchanged messages stand for the following meanings:

g_1 stands for call request.

g_2 stands for call connected.

g_3 stands for incoming call.

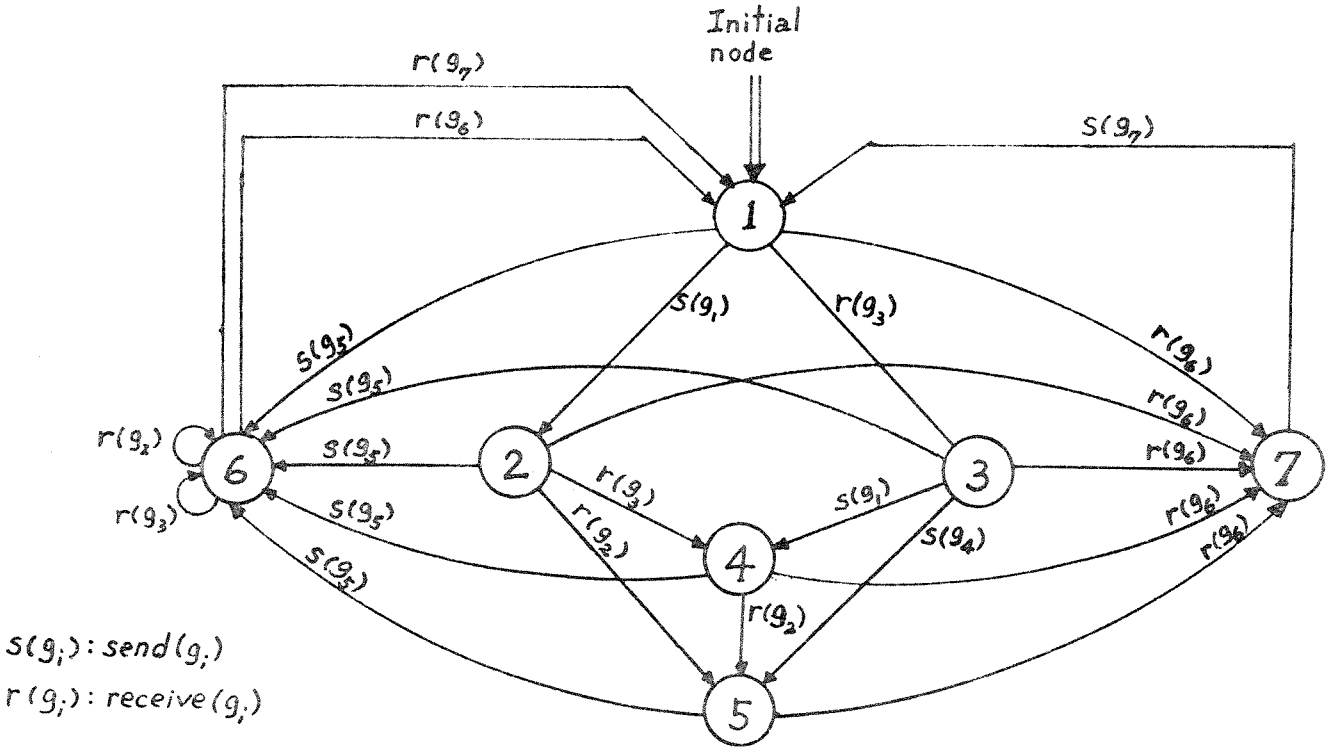
g_4 stands for call accepted.

g_5 stands for clear request.

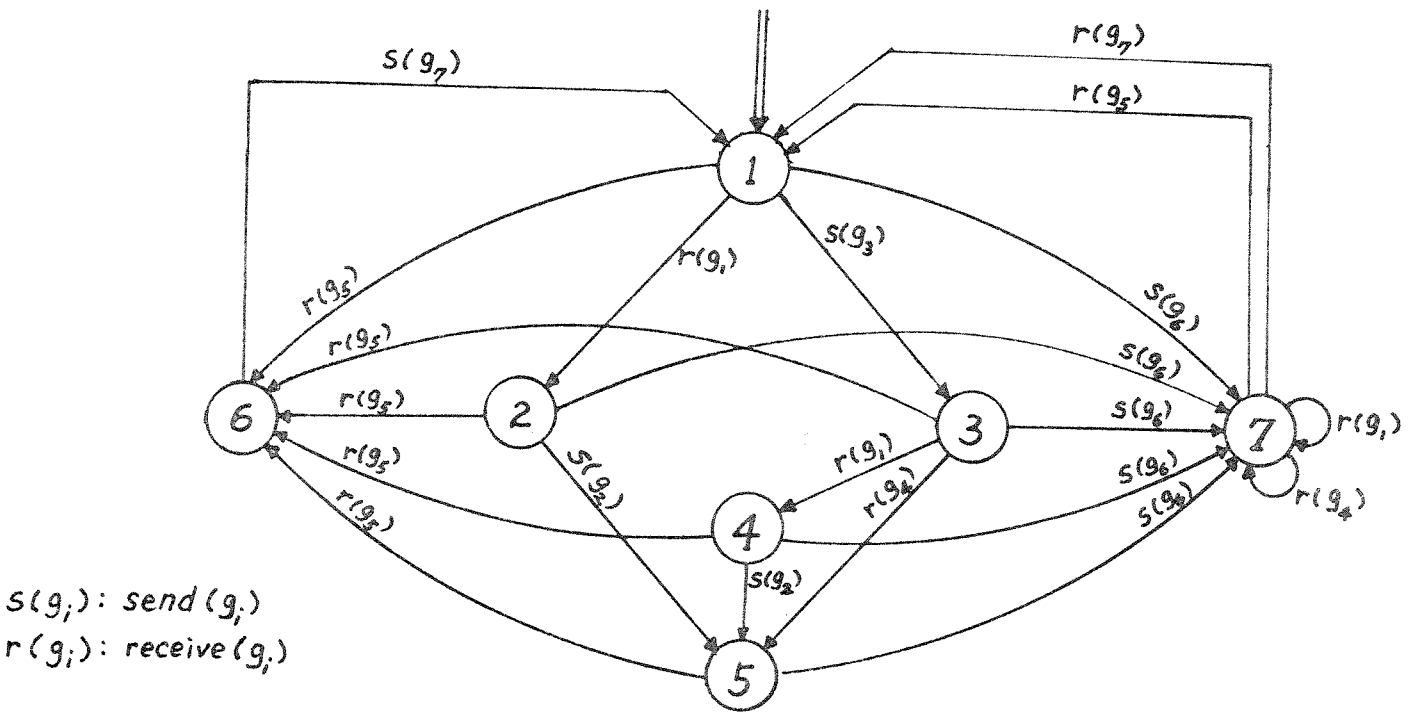
g_6 stands for clear indication.

g_7 stands for clear confirmation.

If maximal progress state exploration and conventional state exploration are applied to these two machines, the following results can be obtained.



(a) M



(b) N

Figure 3: The call establishment/clear protocol in X.25

State exploration method	Number of generated states	Number of generated distinct states
Maximal progress for M	107	62
Maximal progress for N	129	79
Conventional method	246	112

Therefore, maximal progress state exploration can reduce the execution time by a factor of $246/129 \simeq 2$, and can reduce the storage requirment by a factor of $112/79 \simeq 1.5$. []

VIII. CONCLUDING REMARKS

We have presented a variation of state exploration called maximal progress state exploration. It has the advantage of dividing the state exploration task into two completely independent subtasks. In most cases, each subtask requires less execution time and storage than the original task. Thus, by executing these two subtasks in parallel, on the expense of using two processors instead of one, the required execution time is reduced. By executing these two subtasks in sequence on the same processor, the required storage is reduced.

The actual amount of reduction in execution time and/or storage depends on the two communicating machines under consideration. In all the examples we have tried (including those presented in this paper), the reduction in execution time, measured by the total number of generated states, is about 50%; and the reduction in storage, measured by the number of distincted states, is about 30%. More accurate results concerning the actual gain require considering a large population of examples and long experience with the technique.

In this paper, we have only addressed the case of two communicating machines. Extending maximal progress state exploration to more than two machines is still an open problem.

We have shown that maximal progress state exploration can be used to examine properties other than progress; but we have also shown that there is some property that it cannot examine. A list of all the properties which can (or cannot) be examined by this technique still requires further research.

REFERENCES

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

[2] D. Brand and P. Zafiropulo, "On communicating finite-state machines," IBM Research Report, RZ1053(#37725), Jan. 1981. To appear in JACM.

[3] J. Hajek, "Automatically verified data transfer protocols," Proc. Int. Comp. Conf., 1978, pp. 749-756.

[4] J. Rubin, and C. H. West, "An improved protocol validation technique," Computer Network, April 1982.

[5] H. Rudin, and C.H. West, "A validation technique for tightly coupled protocols," IEEE Trans. Computers, Vol. C-31, No.7, July 1982.

[6] C. A. Sunshine, "Formal modeling of communication protocols," USC/Inform. Sc. Institute, Res. Rep. 81-89, March 1981.

[7] S.T. Vuong, and D.D. Cowan, "Automated protocol validation via resynthesis: The CCITT X.75 packet level recommendation as an example," Tech. Report CS-80-39, Dept. of Computer Science, Univ. of Waterloo, revised May 1981.

[8] C. H. West, "An automated technique of communication protocol validation," IEEE Trans. Comm., Vol. COM-26, pp.1271-1275, Aug. 1978.

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

[10] Y. T. Yu "Communicating finite state machines: analysis and synthesis," PH.D Thesis, Dept. of Computer Sciences, Univ. of Texas at Austin, in progress.

[11] Y.T. Yu, and M.G. Gouda, "Deadlock detection for a class of communicating finite state machines," To appear in IEEE Trans. on Comm., Dec. 1982.

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

APPENDIX: PROOFS OF THEOREMS

Proof of Theorem 1:

Let s=[v,w,E,E] be a reachable deadlock state of M and N. Also let Q be a legal sequence which reaches s, and Q_M and Q_N be the projection paths of Q onto M and N respectively. Hence, Q_M is from the initial node to receiving node v in M; and Q_N is from the initial node to receiving node w in N. Also, each sent message in Q_M (or Q_N) must be received in Q_N (or Q_M, respectively).

Construct a sequence Q´ from Q by the following two-step algorithm.

Algorithm A:

```
i.  Initialize Q´:= E (The empty string);
               i := 0;
               j := 0;
ii. while Q´ does not reach an overflow state
          and i+j < the number of symbols in Q
    do
    let [v´,w´,x´,y´] be the state reachable by Q´;
    if v´ is a receiving node and x´ = E
      then i := i +1;
           add a copy of the ith N symbol in Q to Q´
      else j := j+1 ;
           add a copy of the jth M symbol in Q to Q´
    endwhile
```

It is straightforward to show that the sequence Q´ constructed by Algorithm A is a maximal progress sequence for M. There are two ways for this algorithm to terminate, either Q´ reaches an overflow state (in which case, Theorem 1 is correct), or i+j= the number of symbols in Q. In this latter case, the symbols in Q´ are the same as those in Q, but they are in a different order. Thus, the projections of Q´ onto M and N are identical to those of Q; i.e., $Q'_M=Q_M$ and $Q'_N=Q_N$. Therefore, Q'_M is from the initial node to receiving node v in M; and Q'_N is from the initial node to receiving node w in N. Also, each sent message in Q'_M (or Q'_N) must be received in Q'_N (or Q'_M, respectively). Hence, Q´ must reach the deadlock state [v,w,E,E], and Theorem 1 is correct. []

Proof of Theorem 2:

Let s=[v,w,x,y] be a reachable unspecified reception state for M; i.e., $x=g_1 \cdot \ldots \cdot g_r$, and v is a receiving node without an output edge labelled receive(g_1) in M. Let Q be a legal sequence which reaches s; and let Q_M and Q_N be the projection paths of Q onto M and N respectively. Q_M is from the initial node to receiving node v in M; and Q_N is from the initial node to node w in N.

Apply a two-step algorithm, called Algorithm B, to construct a sequence Q´ from Q. Algorithm B is similar to Algorithm A in the proof of Theorem 1, except that the condition of the while statement in the second step is modified to become:

Q´ does not reach an unspecified reception state
and Q´ does not reach an overflow state
and i+j < the number of symbols in Q

It is straightforward to show that the constructed Q´ is a maximal progress sequence for M.

There are three ways for this algorithm to terminate, either Q´ reaches an unspecified reception or overflow state (in either case, Theorem 2 is correct), or i+j= the number of symbols in Q. In this latter case, the symbols in Q´ are the same as those in Q, but they are in a different order. Therefore Q´, like Q, must reach a state $[v,w,g_1 \cdot \ldots g_r,y]$ where v is a receiving node without any output labelled receive(g_1). This state is an unspecified reception state for M; and Theorem 2 is correct. []

Proof of Theorem 3:

Let s=[v,w,x,y] be a reachable unspecified reception state for N; i.e., $y=g_1 \cdot g_2 \cdot \ldots \cdot g_r$, and w is a receiving node without an output edge labelled receive(g_1) in N. Let Q be a legal sequence which reaches s; and let Q_M and Q_N be the projection paths of Q onto M and N respectively. Q_M is from the initial node to node v in M, and Q_N is from the initial node to receiving node w in N.

If node v is a sending node or $x \neq E$, then extend Q_M until one of the following three conditions is met.

i. The extended Q_M, and Q_N reach a state $[v_1,w_1,x_1,y_1]$ where v_1 is a receiving node, and $x_1=E$ (the empty string).

ii. The extended Q_M, and Q_N reach an unspecified reception state for M.

iii. The extended Q_M, and Q_N reach an overflow state for M.

Let the edges in the extension be labelled $b_1,b_2,..,b_s$, where any b_i is either send(g) or receive(g) for some message g Extend sequence Q to become $\bar{Q}$ as follows.

$$\bar{Q} = Q.Mb_1.Mb_2 \ldots .Mb_s$$

The projections of $\bar{Q}$ onto M and N are the extended Q_M, and Q_N respectively.

Apply Algorithm B in the proof of Theorem 2, to construct a sequence Q´ from $\bar{Q}$. It is straightforward to show that the constructed Q´ is a maximal progress sequence for M. (Extending the sequence Q to $\bar{Q}$ before applying the algorithm is intended to ensure that Q´ is a maximal progress sequence for M.)

There are three ways for Algorithm B to terminate, either Q´ reaches an unspecified reception or overflow state (in either case, Theorem 3 is correct), or i+j=the number of symbols in $\bar{Q}$. In this latter case, the symbols in Q´ are the same as those in $\bar{Q}$, but in a different order. Therefore Q´,

like $\bar{Q}$, must reach a state $[v',w,x',y.z]$ where $y=g_1 \cdots g_r$, and w is a receiving node without an output labelled receive(g_1). This state is an unspecified reception state for N; and Therorm 3 is correct. []

Proof of Theorem 4:

Let s=[v,w,x,y] be an overflow state for M, i.e., v is a sending node in M, and |y|=K. Let Q be a legal sequence which reaches s; and let Q_M and Q_N be the projection paths of Q onto M and N respectively. Q_M is from the initial node to sending node v in M; and Q_N is from the initil node to node w in N.

Apply Algorithm A in the proof of Theorem 1 to construct a sequence Q' from Q. It is straightforward to show that the constructed Q' is a maximal progress sequence for M. There are two ways for the above algorithm to terminate and produce Q', either Q' reaches an overflow state, (in which case, Theorem 4 is correct), or i+j=the number of symbols in Q. In this latter case, the symbols in Q' are the same as those in Q, but in a different order. Therefore Q', like Q, must reach the state [v,w,x,y], where v is a sending node and |y|=K. This state is an overflow state; and Theorem 4 is correct. []

Proof of Theorem 6:

Let e be an executable edge with a tail node v in M. We consider the case where e is a sending edge; the proof where e is a receiving edge is similar. Since e is an executable sending edge, then there is a reachable state s of the form [v,w,x,y]. Let Q be a legal sequence which reaches s; and let Q_M and Q_N be the projection paths of Q onto M and and N, respectively. Q_M is from the initial node to sending node v in M; and Q_N is from the initial node to node w in N.

Apply a two-step algorithm, called Algorithm C, to construct a sequence Q' from Q. Algorithm C is similar to Algorithm A in the proof of Theorem 1,

except that the condition of the while statement in the second step is modified to become:

Q´ does not reach a state of the form [v,w´,x´,y´]
and i+j < the number of symbols in Q

It is straightforward to show that the constructed Q´ is a maximal progress sequence for m, and that it reaches a state of the form [v,w´,x´,y´]. Therefore e is reachable by Q´. []