Jack Wolf (S'54–M'60–F'73) was born in Newark, NJ, on March 14, 1935. He received the B.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1956 and the M.S.E., M.A., and Ph.D. degrees from Princeton University, Princeton, NJ, in 1957, 1958, and 1960, respectively.

He was a member of the Department of Electrical Engineering at New York University, New York, NY, from 1963 to 1965 and the Polytechnic Institute of Brooklyn, Brooklyn, NY, from 1965 to 1973. He was Chairman of the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst, from 1973 to 1975 and is currently a Professor there. During 1968–1969, he was a member of the Mathematics Research Center, Bell Laboratories, Murray Hill, NJ. From 1971 to 1972, he was an NSF Senior Postdoctoral Fellow at the University of Hawaii, Honolulu. From 1979 to 1980 he was a Guggenheim Fellow at the University of California at San Diego and the Linkabit Corporation. His research interests are in information theory, algebraic coding theory, communication systems, and computer networks.

Dr. Wolf was corecipient of the 1975 Information Theory Group Paper Award for the paper "Noiseless coding of correlated information sources" (coauthored with D. Slepian). He was Cochairman of the 1969 International Symposium on Information Theory and was Editor for Algebraic Coding for the IEEE TRANSACTIONS ON INFORMATION THEORY from 1969 to 1972. He served on the Board of Governors of the IEEE Information Theory Group from 1970 to 1976 and since 1980, and was the President in 1974. He is currently International Chairman of Commission C of the International Scientific Radio Union (URSI).

Synthesis of Communicating Finite-State Machines with Guaranteed Progress

MOHAMED G. GOUDA, MEMBER, IEEE, AND YAO-TIN YU

Abstract—**We present a methodology to synthesize two communicating finite-state machines which exchange messages over two one-directional, FIFO channels. The methodology consists of two algorithms. The first algorithm takes one machine M, and constructs two communicating machines M' and N' such that 1) M' is constructed from M by adding some receiving transitions to it, and 2) the communication between M' and N' is bounded and free from deadlocks, unspecified receptions, nonexecutable transitions, and state ambiguities. The second algorithm takes the two machines M' and N' which result from the first algorithm, and computes the smallest possible capacities for the two channels between them. Both algorithms require an $O(st)$ time, where s is the number of states in the given machine M, and t is the number of state transitions in M; thus, the methodology is practical to use.**

I. INTRODUCTION

MANY communication protocols can be modeled as two communicating, finite-state machines which exchange messages over two one-directional, unbounded, FIFO channels [1], [3], [11]–[14]. The communication between the two machines in each of these models is often expected to satisfy some "nice" progress properties [1], [18], [19].

Four of these progress properties are of interest to the discussion in this paper. They are boundedness and freedom from communication deadlocks, unspecified receptions, and nonexecutable state transitions. (Formal definitions of these properties are discussed later in Section II.)

Paper approved by the Editor for Computer Communications of the IEEE Communications Society for publication without oral presentation. Manuscript received July 15, 1981; revised December 10, 1983.

M. G. Gouda is with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712.

Y.-T. Yu was with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712. He is now with the Department of Computer Science, University of Iowa, Iowa City, IA 52240.

There are two basic approaches to ensure that the communication between two finite-state machines satisfies such progress properties.

1) Analysis: Develop techniques to prove that the communication between any two given machines satisfies the required progress properties.

2) Synthesis: Develop techniques to complete two given (incomplete) machines such that the communication between the completed machines is guaranteed to satisfy the required progress properties.

Brand and Zafiropulo [4] have shown that the analysis approach is undecidable in general; i.e., no algorithm can decide whether the communication between two finite-state machines satisfies any of the progress properties mentioned earlier. (Nevertheless, the problem can still be decided for some special classes of communicating finite-state machines [4], [7], [8], [10], [15]–[17].) This rather negative result of the analysis approach makes the synthesis approach more attractive. In this paper, we present a practical methodology to synthesize two communicating finite-state machines with guaranteed progress properties. A preliminary version of this methodology has been presented in [6].

A. Related Work

Previous work in the synthesis approach can be distinguished into two categories based on the objective of the synthesis.

1) Synthesis to Achieve Progress: Zafiropulo *et al.* [19] have presented a methodology, henceforth referred to as the ZWRCB methodology, to synthesize two finite-state machines whose communication satisfies some progress properties. The methodology proceeds in steps; at each step, the following three substeps are performed.

a) First, the designer adds one sending transition to one of the two incomplete machines.

b) Second, the designer executes an algorithm (based on three synthesis rules) to add the corresponding receiving transi-

0090-6778/84/0700-0779$01.00

tions in the other machine such that freedom from unspecified receptions and nonexecutable transitions is guaranteed. (This addition of receiving transitions to the second machine may necessitate the addition of sending transitions, that are copies of previously added sending transitions, to this same machine; this in turn necessitates the addition of corresponding receiving transitions to the first machine, and so on.)

c) Third, the designer checks whether or not the added transitions can lead to a deadlock or a channel overflow. If a deadlock or a channel overflow is detected, then the designer must take a proper action (e.g., remove all the added transitions in this step). Finally, the designer proceeds to the next step.

These steps continue until the designer does not need to add further sending transitions to any of the two machines. In this case, the two machines are complete and their communication is guaranteed to be bounded and free from deadlocks, unspecified receptions, and nonexecutable transitions.

The synthesis methodology presented in this paper also falls into this same category; it has the same objectives as the ZWRCB methodology. A comparison between the two methodologies is discussed later in Section VI.

2) Synthesis to Achieve Progress and Service: Bochmann and Merlin [2], [9] have considered a special class of communicating finite-state machines. In this special case, the sending of a message by one machine and its reception by another machine occur instantaneously in zero time. Therefore, channels are not needed to buffer messages between different machines, and the analysis problem becomes trivially decidable. (Actually, unboundedness and unspecified receptions cannot occur in this class; only deadlocks and nonexecutable transitions can occur.)

Bochmann and Merlin have also introduced the concept of a "service machine" which is a finite-state machine that defines the service performed by a set of communicating finite-state machines. They state the following problem. "Given $n-1$ communicating finite-state machines ($n \geqslant 2$), and a service machine, it is required to synthesize an nth communicating machine such that the service performed by the n communicating machines is defined by the given service machine or by a maximal submachine of it."

Their solution to the problem consists of a "formula" that defines the required communicating machine from the given machines. They observe, however, that the resulting communicating machine may have many redundant transitions and may reach a deadlock with the given communicating machines. Therefore, they suggest a subsequent procedure to "trim" the resulting machine by removing some of its transitions. The trimming procedure is based on state exploration to determine which transitions in the resulting machine can (or should) be removed; and so it consumes a large amount of time.

Later, Gouda and Chu [5] have discussed another solution to the Bochmann-Merlin problem in the special case of $n = 2$; their solution does not require any state exploration.

B. The Paper's Organization

Following the Introduction, the model of communicating finite-state machines is presented in Section II, along with its major progress properties. The two algorithms which comprise our synthesis methodology are presented in Sections III and IV. Then in Section V, we apply the methodology to synthesize a call establishment/clear protocol similar to that of X.25. Concluding remarks are in Section VI. Due to space limitations, we have omitted the correctness proofs for the two algorithms; these proofs are discussed in [15].

II. Communicating Machines

A *communicating machine* M is a labeled directed graph with two types of edges called *sending* and *receiving edges*. A sending (or receiving) edge is labeled send(g) (or receive(g), respectively), for some *message* g in a finite set G of messages. No two outputs of the same node in M have identical labels.

Each node in M has a distinct label and at least one output edge. A node is called a *sending* (or *receiving*) *node* iff all its output edges are sending (or receiving, respectively) edges; otherwise it is called a *mixed 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.

Fig. 1(a) shows a communicating machine M with one sending node (node 1), one receiving node (node 2), and one mixed node (node 3). Node 1 is the initial node of M.

Let M and N be two communicating 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 the labels of two nodes in M and N, respectively, and x and y are two strings of messages from the set G. Informally, a state $[v, w, x, y]$ means that the execution of 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 *initial state* of M and N is $[v_0, w_0, E, E]$ where v_0 and w_0 are the labels of the initial nodes of 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' of M and N is said to *follow s over e* iff the following four conditions are satisfied.

1) If e if from v to v' in M and is labeled send(g),
   then $s' = [v', w, x, y \cdot g]$, where "$\cdot$" is the concatenation operator.
2) If e is from w to w' in N and is labeled send(g),
   then $s' = [v, w', x \cdot g, y]$.
3) If e is from v to v' in M and is labeled receive(g),
   then $x = g \cdot x'$,
   and $s' = [v', w, x', y]$.
4) If e is from w to w' in N and is labeled receive(g),
   then $y = g \cdot y'$,
   and $s' = [v, w', x, y']$.

Let s and s' be two states of M and N. s' *follows* s iff there is an edge e in M or N such that s' follows s over e.

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

A state s of M and N is *reachable* iff it is reachable from the initial state of M and N.

In designing a pair of communicating machines M and N, a designer may commit some design mistakes which cause the resulting machines to exhibit some progress errors during the course of communication. Five of these progress errors are defined next.

1) Unbounded Communication: The communication between M and N is said to be *bounded by* a positive integer K iff for any reachable state $[v, w, x, y]$ of M and N, $|x| \leqslant K$ and $|y| \leqslant K$, where $|x|$ is the number of messages in the string x. The communication is said to be *bounded* iff it is bounded by some positive integer K. Otherwise, it is unbounded. Notice that an unbounded communication cannot be implemented correctly using finite-capacity channels.

2) Communication Deadlocks: A state $s = [v, w, x, y]$ of M and N is a *deadlock state* iff a) both v and w are receiving nodes, and b) $x = y = E$. A reachable deadlock state constitutes a progress error, since the two machines cannot progress after reaching a deadlock state.

3) Unspecified Receptions: A state $s = [v, w, x, y]$ of M and N is an *unspecified reception state* iff one of the following two conditions is satisfied.

a) $x = g_1 \cdot g_2 \cdot \cdots \cdot g_k$, from some $k \geqslant 1$ and v is a receiving node without any output edge labeled receive(g_1) in M.

b) $y = g_1 \cdot g_2 \cdot \cdots \cdot g_k$, for some $k \geqslant 1$ and w is a receiving node without any output edge labeled receive(g_1) in N.

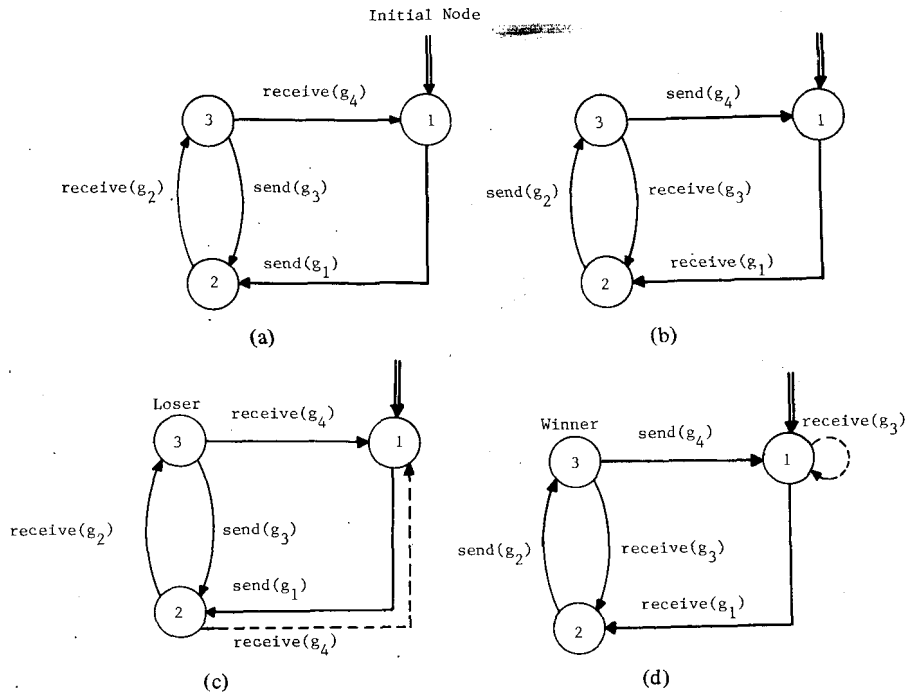

Fig. 1. A machine synthesis example. (a) M. (b) N. (c) M'. (d) N'.

A reachable unspecified reception state is a progress error since at least one of the two machines cannot progress after reaching an unspecified reception state.

Notice that according to this definition, unspecified receptions can occur only at receiving nodes. Hence, this definition is different from the one in [19] where unspecified receptions can occur at receiving or mixed nodes.

4) Nonexecutable Transitions: Let e be an edge in machine M (or N). e is said to be *nonexecutable* during the communication between M and N iff there is no pair of reachable states s_1 and s_2 such that s_2 follows s_1 over edge e.

A nonexecutable edge is not strictly a progress error, since the communication between M and N can proceed properly even in the presence of nonexecutable edges. Nonetheless, nonexecutable edges serve no function, and it is desirable to remove them from M and N.

5) State Ambiguities: A state $[v, w, E, E]$ of M and N is said to be *stable* iff $x = y = E$ (the empty string). A *stable state* ambiguity exists between M and N iff there are two reachable stable states $[v_1, w_1, E, E]$ and $[v_2, w_2, E, E]$ such that either $v_1 = v_2$ and $w_1 \neq w_2$, or $v_1 \neq v_2$ and $w_1 = w_2$. A state ambiguity is not necessarily an error unless the designer intends to have no state ambiguities in the resulting two communicating machines. □

In this paper, we present a methodology to construct pairs of communicating machines whose communication is free of the above progress errors. The methodology consists of two algorithms named the machine synthesis algorithm and the channel capacity algorithm.

The *machine synthesis algorithm* takes as an input one communicating machine M and constructs two communicating machines M' and N' which satisfy the following two conditions.

1) M' is constructed from M by adding some receiving edges to it.

2) The communication between M' and N' is free of the above five progress errors.

The *channel capacity algorithm* takes as an input the two machines M' and N' which result from the first algorithm and computes the smallest possible channel capacities between them. In other words, it computes the smallest positive integers c_1 and c_2 such that for an reachable state $[v, w, x, y]$ of M' and N', $|x| \leq c_1$ and $|y| \leq c_2$. The integer c_1 is the smallest possible capacity for the output channel of N', and c_2 is the smallest possible capacity for the output channel of M'.

III. The Machine Synthesis Algorithm

In this section, the machine synthesis algorithm is discussed. First, two machine synthesis examples are discussed in Sections III-A and III-B to motivate the algorithm. Then, the algorithm itself is given in Section III-C.

A. Dual Machines and Winner/Loser Nodes

Consider the communicating machine M in Fig. 1(a). Assuming that M can be modified slightly by adding receiving edges to it, it is required to synthesize another machine N' such that the communication between the modified M, called M', and N' is free from the five progress errors discussed in Section II.

The first step is to construct a *dual machine* N [Fig. 1(b)] which is identical to M except that each sending (or receiving) edge in M is replaced by a receiving (or sending, respectively) edge in N. Thus, each sending (or receiving or mixed) node in M corresponds to a receiving (or sending or mixed, respectively) node in N. Two corresponding nodes in M and N are called *dual nodes*. For convenience, every node in N has the same label as its dual node in M.

During the communication between M and N, the two machines traverse dual paths in harmony; i.e., while one machine sends some message the other machine receives the same message. This continues until M and N reach dual mixed nodes. In this case, both M and N may traverse paths of sending edges, causing a loss of synchronization. For example, the two machines M and N in Fig. 1 can start from the initial state $[1, 1, E, E]$ and traverse dual paths to reach the state $[3, 3, E, E]$. From this state, M can send message g_3 and reach receiving node 2, and N can send message g_4 and reach receiving node 1; i.e., the state $[2, 1, g_4, g_3]$ is reached. There are two problems with this state.

1) Machine M does not expect to receive message g_4 at receiving node 2, and N does not expect to receive message g_3 at receiving node 1.

2) Assuming that M receives g_4 at node 2 and N receives g_3 at node 1, and so they both recognize loss of synchronization, what should they do to restore their synchronization?

To solve the first problem, an output edge labeled receive(g_4) should be added to receiving node 2 in M, and an output edge labeled receive(g_3) should be added to node 1 in N. These added edges are called *correcting edges.* Notice that we have not yet defined the head nodes of these correcting edges; this is done next as we discuss a solution to the second problem.

When M receives g_4 at node 2, it should recognize that a loss of synchronization with N has occurred at node 3. In particular, it should recognize that while M itself has reached node 2, N has reached node 1. Therefore, to restore the lost synchronization, M should leave node 2 and reach node 1; i.e., the correcting output edge of node 2 should be input to node 1 in M. On the other hand, when N recognizes the loss of synchronization, it should remain at node 1 knowing that eventually M will reach node 1 also, and the synchronization will be restored. Hence, the correcting output edge of node 1 should be input to node 1 in N. The resulting M' and N' after adding the correcting edges to M and N are shown in Fig. 1(c) and (d), respectively. For convenience, the correcting edges are shown as dashed edges.

During the course of communication between M' and N', whenever a loss of synchronization occurs at the dual mixed node pair (3, 3), machine M' is forced to stop its progress and rejoin N', thus restoring the synchronization between the two machines. Hence, mixed node 3 in M' is called a *loser,* while mixed node 3 in N' is called a *winner.*

It would have been also possible to make the correcting edge of M' from node 2 to node 2, and the correcting edge of N' from node 1 to node 2. In this case, whenever a loss of synchronization occurs, machine N' would be the one forced to stop its progress and rejoin the other machine M'. Therefore, in this case, mixed node 3 in N' would be the one called a loser, while mixed node 3 in M' would be called a winner.

From the above example, we reach the following conclusions concerning loss of synchronization between dual communicating machines.

1) Loss of synchronization can start at any dual mixed node pair.

2) Loss of synchronization can be detected by either machine receiving an unexpected message at the first receiving node following the mixed node where the loss of synchronization has started.

3) Loss of synchronization can be corrected by one machine stopping its progress and rejoining the other machine. The mixed node where loss of synchronization has started in the former machine is called a loser, and its dual mixed node in the latter machine is called a winner.

4) From 1) and 3) above, one node in each dual mixed node pair should be selected as a loser while the other node in the pair is selected as a winner.

5) Which node in a dual mixed node pair is selected as a loser or winner is, in principle, an arbitrary decision.

B. Receiving a Sequence of Messages

Consider the communicating machine M in Fig. 2(a). As before, it is required to modify M slightly by adding receiving edges to it to become M' and to synthesize another communicating machine N' such that the communication between M' and N' is free of the progress errors discussed in Section II.

Fig. 2(b) shows the dual machine N for the given machine M. The only dual mixed node pair in M and N is (1, 1). Assume that node 1 in M is selected as a loser; then node 1 in N

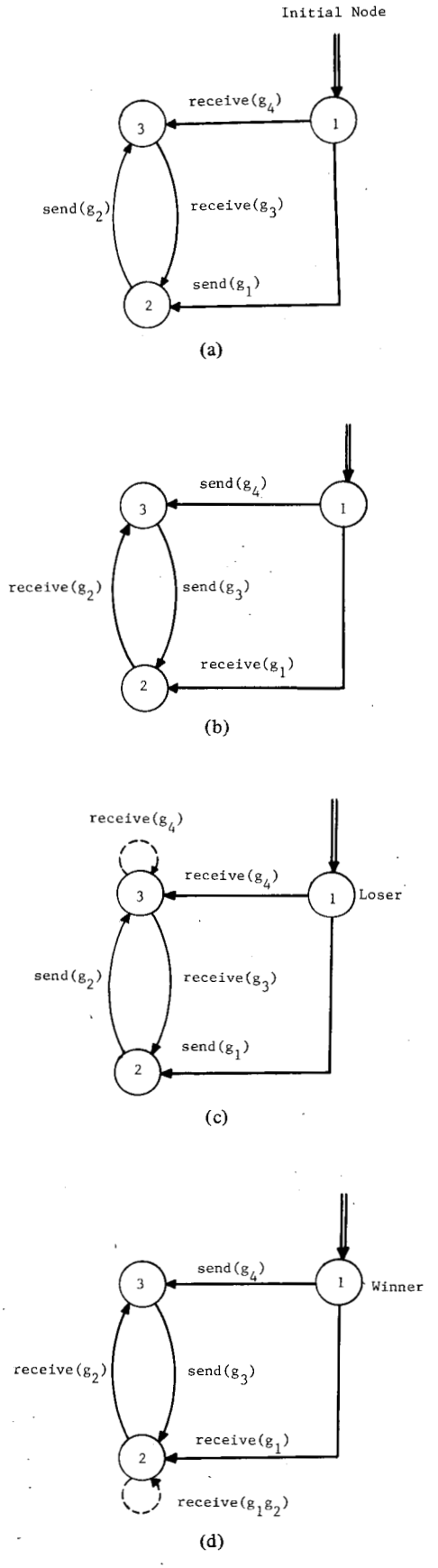

Fig. 2. A second machine synthesis example. (a) M. (b) N. (c) M'. (d) N'.

must be selected as a winner. It remains now to add the correcting edges to M and N, thus constructing the required machines M' and N'.

The correcting edges of M should be added as outputs to the first receiving nodes which follow mixed node 1 in M. There is only one such node, namely receiving node 3, in M. Also, each correcting edge should be labeled receive(g) where g is a message that can be received at mixed node 1 in M. There is only one such message, namely g_4. Thus, one correcting edge labeled receive(g_4) should be added as an output of receiving node 3 in M. The destination of this edge should be the node which can be reached from mixed node 1 by the edge labeled receive(g_4), namely node 3 in M, as shown in Fig. 2(c).

Adding correcting edges for a winner node is more complicated than for a loser node. In case of a loser node, a correcting edge receives the *first* message sent by the other machine during the loss of synchronization, and redirects its machine to rejoin the other machine. In case of a winner node, a correcting edge receives *all* the messages sent by the other machine during the loss of synchronization, and "discards" them, and directs its machine to stay at its current node. Thus, a correcting edge for a winner node should satisfy the following two conditions.

1) It should form a self-loop at a first receiving node following the winner mixed node.

2) It should be labeled receive(x), where x is a complete sequence of messages sent by the other machine during the loss of synchronization. (An edge labeled receive($g_1 \cdot g_2 \cdot \cdots \cdot g_r$) is equivalent to a directed path of r receiving edges labeled receive(g_1), receive(g_2), $\cdots$, and receive(g_r), respectively. To refer to it as an edge rather than a path is a notational convenience.)

The only sequence of messages sent by machine M during its loss of synchronization with N is $g_1 g_2$; thus, each correcting edge added to N should be labeled receive($g_1 g_2$). Also, N has only one receiving node, namely node 2, that follows the winner mixed node 1; hence, one correcting edge, labeled receive($g_1 g_2$), should be added as a self-loop at node 2 in N. The resulting machine N' is shown in Fig. 2(d).

C. The Algorithm

The above examples are intended to give some insight into the different steps of the machine synthesis algorithm. The algorithm is presented next. (A correctness proof for the algorithm is given in [15].)

Algorithm 1:

Inputs: A communicating machine M which satisfies the following two conditions.

a) All edges in M have distinct labels.

b) Each directed cycle in M must have at least one sending and one receiving edge.

Outputs: Two communicating machines M' and N' which satisfy the following two conditions.

a) M' is constructed from M by adding some receiving edges to it.

b) The communication between M' and N' is bounded, deadlock-free, and has no unspecified receptions, no nonexecutable transitions, and no state ambiguities.

Steps:

a) Construct the dual machine N from the given machine M by replacing each edge labeled send(g) [or receive(g)] in M by an edge labeled receive(g) [or send(g)] in N.

b) Select each mixed node in M, at random, to be either a loser or a winner. If a mixed node in M is selected a loser (or a winner), then the corresponding dual node in N must be selected a winner (or a loser, respectively).

c) Construct M' from M by applying the loser (or winner) transformation, defined next, to every loser (or winner, respectively) mixed node in M. Similarly, construct N' from N by applying the loser (or winner) transformation to every loser (or winner, respectively) mixed node in N. □

Notice that if M has no mixed nodes, then steps b) and c) will not modify M and N in any way, and the required M' and N' are the original M and N. Next, we define the loser and winner transformations for mixed nodes in M. (The loser and winner transformations for mixed nodes in N are similar.)

Loser Transformation: for a loser mixed node v in M [Fig. 3(a)].

Let u_i ($i = 1 \cdots m$) be all the receiving nodes such that there is a directed path of sending edges from node v to node u_i in M.

Let receive(g_j) ($j = 1 \cdots n$) be the label of a receiving edge from node v to some node v_j ($j = 1 \cdots n$) in M.

Then add a correcting edge labeled receive(g_j) from each u_i ($i = 1 \cdots m$) to each v_j ($j = 1 \cdots n$) in M. □

Winner Transformation: for a winner mixed node v in M [Fig. 3(b)].

Let u_i ($i = 1 \cdots m$) be all the receiving nodes such that there is a directed path of sending edges from node v to node u_i in M.

Let x_j ($j = 1 \cdots n$) be an ordered sequence of messages which label the edges of a directed path of receiving edges from node v to a sending node in M.

Then add a correcting self-loop labeled receive(x_j) ($j = 1 \cdots n$) at each receiving node u_i ($i = 1 \cdots m$) in M. □

Later in Section V, we discuss how to apply Algorithm 1 to synthesize two communicating machines which represent a call establishment/clear protocol similar to that of X.25.

IV. The Channel Capacity Algorithm

In this section, the channel algorithm is discussed. First, we discuss two examples in Sections IV-A and IV-B to motivate the algorithm. Then, the algorithm itself is given in Section IV-C.

A. Dealing with Loser Mixed Nodes

Consider the communicating machine M' in Fig. 1(c) and assume that it is required to compute the smallest possible capacity for its output channel to N' (i.e., compute the maximum number of messages which can exist simultaneously in the output channel of M').

First, we observe that each sending edge in M' contributes one message to the output channel of M'. So, we assign each sending edge a weight "1", and assign each receiving edge a weight "0", as shown in Fig. 4(a). Next, we apply a number of transformations on M' to remove some of its directed paths such that the following condition holds. For each removed path p_1, M' has a remaining path p_2 such that $m_2 \geq m_1$, where m_i is the maximum number of messages which can exist simultaneously in the output channel of M' as M' "executes" path p_i ($i = 1, 2$). These transformations leave M' acyclic; thus, the smallest possible capacity for the output channel of M' is the weight of the directed path with the maximum weight in M'. (Recall that each edge in M' has a weight; hence, the weight $|p|$ of a directed path p is the sum of weights of its edges.)

During the communication between M' and N', M' can go from node 3 to node 1 either by receiving message g_4 or by sending g_3, then receiving the correcting message g_1. The second path adds one message to the output channel of M', but the first path does not. Therefore, removing the first path from M' will not change the output channel capacity of M'. The procedure to remove the first path may seem strange at first. Remove the correcting edge in the second path, and change the weight assigned to the receiving edge in the first path from 0 to 1, as shown in Fig. 4(b). So now, M' must traverse the receiving edge to go from node 3 to node 1; but in doing so, it simulates the effect of the second path, namely,

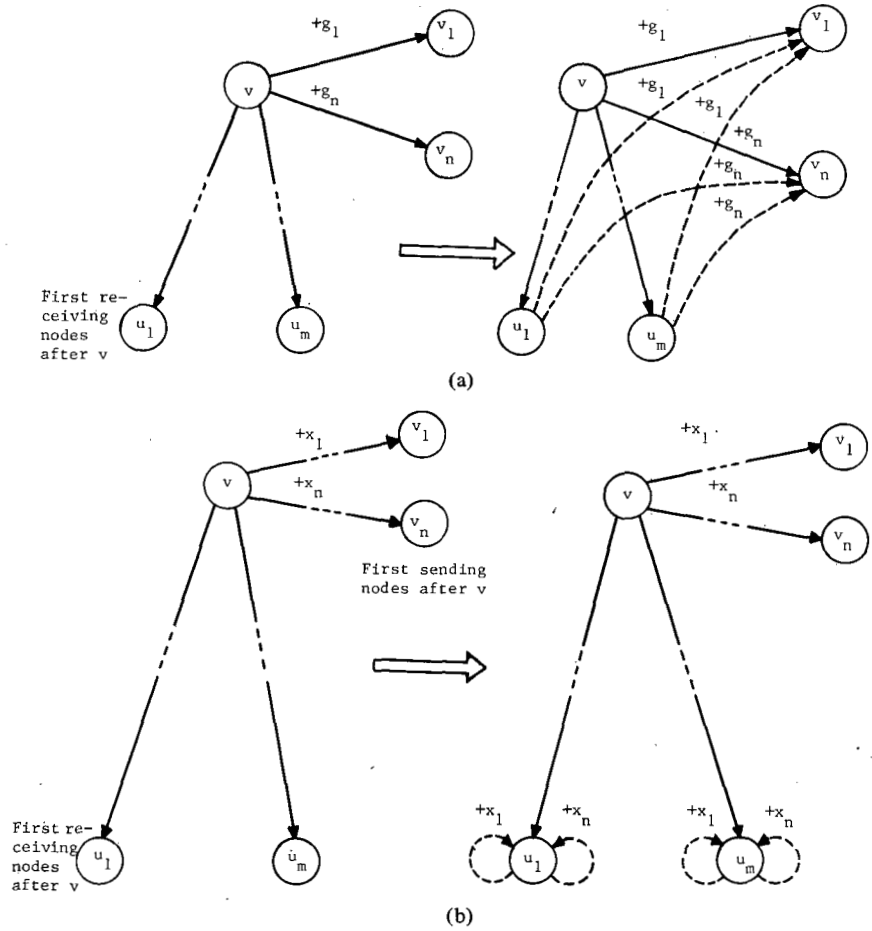

Fig. 3. Winner and loser transfigurations. [Notation: $+g$ = receive (g).] (a) Loser transformation. (b) Winner transformation.

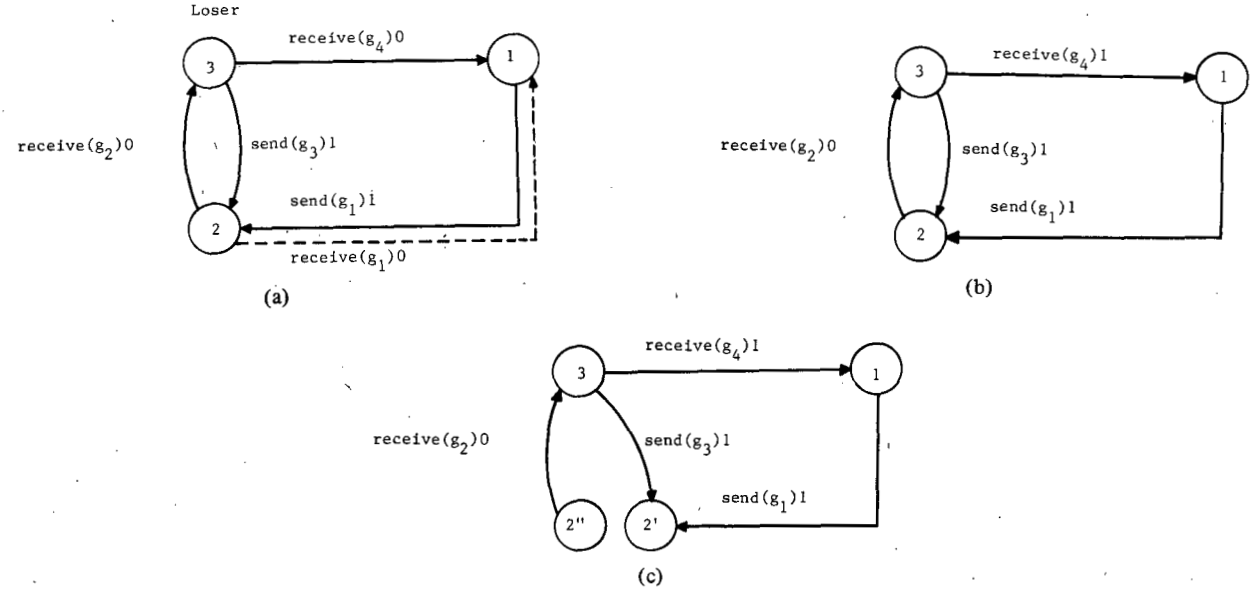

Fig. 4. Computing the smallest possible capacity for the output channel of M' in Fig. 1(c).

it sends one message and receives g_4. The reason for selecting this indirect procedure to remove the first path is to ensure that this transformation with other transformations will leave M' acyclic.

Because of the way M' and N' are constructed by Algorithm 1, the following property holds during the communication between M' and N'. If M' or N' ever sends a message, then receives a noncorrecting message, then its output channel must be empty immediately before the message reception. (A proof of this property is given in [15].) From this observation, whenever M' reaches node 2, then immediately before receiving g_2 its output channel must be empty. Therefore, it is possible to partition node 2 into two nodes $2'$ and $2''$ such that $2'$ has all the sending input edges of node 2, and $2''$ has all the receiving input edges and all the receiving output edges of node 2, as shown in Fig. 4(c). Notice that in this case node 2 (and so node $2''$) has no receiving input edges. Notice also that this partitioning of node 2 removes many directed paths from M', namely, those paths which contain node 2. However, for each removed path p_1, M' still has a path p_2 which contains $2'$ or $2''$ (but not both) such that $m_2 \geq m_1$ where m_i is the maximum number of messages which can exist simultaneously in the output channel of M', as M' traverses path p_i $(i = 1, 2)$.

The resulting M' is acyclic, and so the smallest capacity of its output channel is the weight of the directed path with the maximum weight in M'. From Fig. 4(c), the directed path with the maximum weight in M' is $(2'', 3, 1, 2')$; its weight is $0 + 1 + 1 = 2$; hence, the smallest output channel capacity for M' is two.

B. Dealing with Winner Mixed Nodes

Assume that it is required to compute the smallest capacity for the output channel N' in Fig. 1(d). As before, assign each sending edge a weight of 1, and each receiving edge a weight of 0, as shown in Fig. 5(a).

Correcting edges for winner nodes can be removed without affecting the output channel capacity. There is only one such edge in N', and so it can be removed as shown in Fig. 5(b).

As discussed earlier, whenever N' (or M') sends a message then receives a noncorrecting message, the output channel of N' (or M', respectively) must be empty immediately before the message reception. Based on this observation, the following two transformations can be applied on N'.

1) As shown in Fig. 5(c), receiving node 1 is partitioned into two nodes $1'$ and $1''$ such that $1'$ has all the sending input edges of node 1, and $1''$ has all the receiving input edges and all the receiving output edges of node 1. (Notice that node 1, and so $1''$, has no receiving input edges.)

2) As shown in Fig. 5(d), the winner mixed node 3 is partitioned into two nodes $3'$ and $3''$ such that $3'$ has all the sending input edges and all the sending output edges of node 3, and $3''$ has all the receiving input edges and all the output edges (whether sending or receiving) of node 3. (Notice that node 3, and so $3''$, has no receiving input edges.)

The resulting N' in Fig. 5(d) is acyclic. The directed path with maximum weight in N' is $(3'', 2, 3', 1')$; its weight is $0 + 1 + 1 = 2$. Therefore, the smallest output channel capacity for N' is two.

C. The Algorithm

The above examples are intended to give some insight into the different steps of the channel capacity algorithm. The algorithm is presented next. (A correctness proof for the algorithm is given in [15].)

Algorithm 2:

Inputs: Two communicating machines M' and N' which result from Algorithm 1.

Outputs: The smallest possible capacities for the two channels between M' and N'.

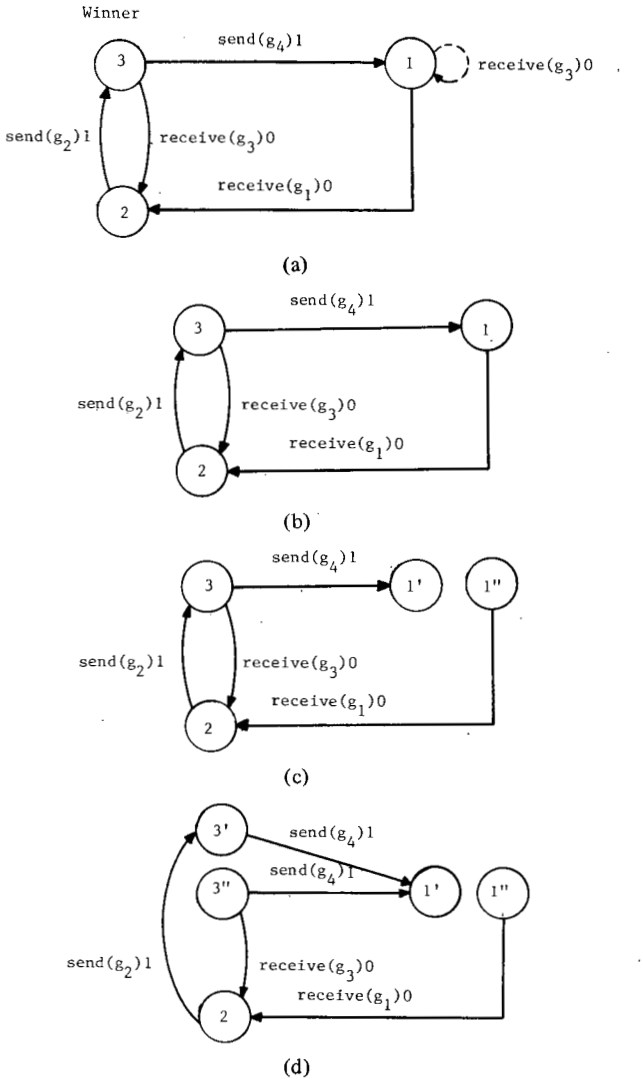

Fig. 5. Computing the smallest possible capacity for the output channel of N' in Fig. 1(d).

Steps:

a) Assign each sending edge in M' a weight of "1", and each receiving edge in M' a weight of "0". In such a weighted graph, the weight $|p|$ of a directed path p is the sum of weights of its edges.

b) Construct a directed weighted graph from M' by the following four steps.

i) **for** each loser mixed node v
do find a directed path p, of sending edges, which starts with v such that $|p| \geq |q|$, where q is any directed path, of sending edges, which starts with v; change the weight of each receiving output edge of v (from "0") to $|p|$.

ii) Remove all the correcting edges.

iii) **for** each loser mixed (or receiving) node v that follows immediately a sending edge
do partition node v into two nodes v' and v'', where v' has all the sending input edges and all the sending output edges of v, and v'' has all the receiving input edges and all the receiving output edges of v.

iv) **for** each winner mixed node v that follows immediately a sending edge
do partition node v into two nodes v' and v'',

where v' has all the sending input edges and all the sending output edges of v, and v'' has all the receiving input edges and all the output edges (whether sending or receiving) of v.

c) The resulting graph is acyclic. Thus, the smallest capacity for the output channel of M' is $|p|$ where p is the directed path with maximum weight in the resulting graph.

d) Repeat steps a)–c) on the other machine N' to compute the capacity of its output channel. □

Notice that if M' (and so N') has no mixed nodes, then the smallest output channel capacity of M' (or N') equals the length of longest sending path in M' (or N', respectively). Next, we discuss how to use Algorithms 1 and 2 to synthesize a call establishment/clear protocol similar to that of X.25.

V. Synthesizing an X.25 Protocol

Consider the two communicating machines $\bar{M}$ and $\bar{N}$ in Fig. 6. They represent the call establishment/clear protocol in X.25 [13], where $\bar{M}$ models the DTE, and $\bar{N}$ models the DCE, and the exchanged messages have the following meaning:

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.

The *functionality* of $\bar{M}$ and $\bar{N}$ can be defined as the set of all sequences of sending and receiving operations executed by the two machines starting from their initial state until they both return to nodes 1 and 1. Five examples of these sequences are now discussed.

1) In this sequence, $\bar{M}$ establishes a connection with $\bar{N}$, then clears it:

$\langle \bar{M}$ sends $g_1, \bar{N}$ receives $g_1, \bar{N}$ sends $g_2, \bar{M}$ receives $g_2,$

$\bar{M}$ sends $g_5, \bar{N}$ receives $g_5, \bar{N}$ sends g_7, M receives $g_7\rangle$

2) In this sequence, $\bar{N}$ establishes a connection with $\bar{M}$, then clears it:

$\langle \bar{N}$ sends $g_3, \bar{M}$ receives $g_3, \bar{M}$ sends $g_4, \bar{N}$ receives $g_4,$

$\bar{N}$ sends $g_6, \bar{M}$ receives $g_6, \bar{M}$ sends $g_7, \bar{N}$ receives $g_7\rangle$

3) This is a "collision" sequence, where each of $\bar{M}$ and $\bar{N}$ tries to establish a connection, then the DTE $\bar{M}$ wins (i.e., it establishes the connection, then clears it):

$\langle \bar{M}$ sends $g_1, \bar{N}$ sends $g_3, \bar{M}$ receives $g_3, \bar{N}$ receives $g_1,$

$\bar{N}$ sends $g_2, \bar{M}$ receives $g_2, \bar{M}$ sends $g_5, \bar{N}$ receives $g_5,$

$\bar{N}$ sends $g_7, \bar{M}$ receives $g_7\rangle$

4) This is a "collision" sequence, where each of $\bar{M}$ and $\bar{N}$ tries to clear the connection; they both succeed:

$\langle \bar{M}$ sends $g_5, \bar{N}$ sends $g_6, \bar{M}$ receives $g_6, \bar{N}$ receives $g_5\rangle$

5) In this sequence, $\bar{M}$ tries to establish a call; but before it receives a response from $\bar{N}$, it clears the call:

$\langle \bar{M}$ sends $g_1, \bar{N}$ receives $g_1, \bar{N}$ sends $g_2, \bar{M}$ sends $g_5,$

$\bar{M}$ receives $g_2, \bar{N}$ receives $g_5, \bar{N}$ sends $g_7, \bar{M}$ receives $g_7\rangle$

Let us use our synthesis methodology to try to construct

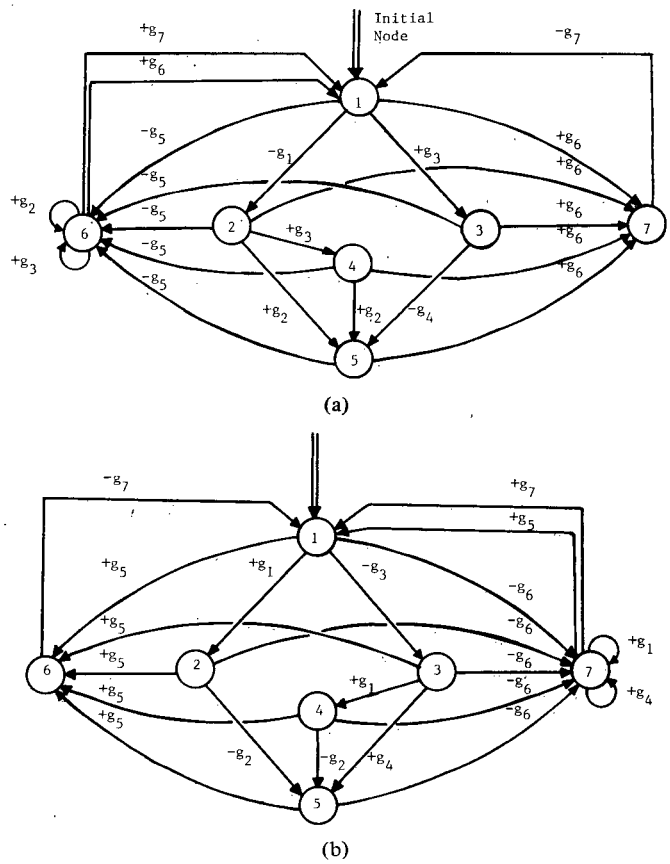

Fig. 6. The call establishment/clear protocol in X.25. [Notation: $-g$ = send (g), $+g$ = receive (g).] (a) $\bar{M}$. (b) $\bar{N}$.

two communicating machines M' and N' with the same functionality as $\bar{M}$ and $\bar{N}$. We start with the communicating machine M in Fig. 7(a). (Constructing this initial machine is not part of our synthesis methodology.) This initial machine should satisfy conditions a) and b) in the input section of Algorithm 1. Hence, instead of having three occurrences of the message label g_5, we distinguish them into $g_5{}^1$, $g_5{}^2$, and $g_5{}^3$, similarly for g_6 and g_7.

Communicating machine M in Fig. 7(a) has two mixed nodes; each of them can be selected arbitrarily as a loser or a winner. Assume that all the mixed nodes in M are selected as winners, and apply the winner transformation to each of them. The resulting communicating machine M' is shown in Fig. 7(b). (Notice that in M' all the message labels $g_5{}^i$, $g_6{}^j$, and $g_7{}^k$ are replaced by g_5, g_6, and g_7, respectively. This is possible since the replacement does not cause a node in M' to have two output edges with identical labels.)

Let N be the dual communicating machine for M. Like M, N has two mixed nodes. Each mixed node in N should be selected as a loser, and the loser transformation should be applied to each of them. The resulting communicating machine N' is shown in Fig. 7(c). (As in M', the message labels $g_5{}^i$, $g_6{}^j$, and $g_7{}^k$ are replaced by g_5, g_6, and g_7, respectively, in N'.)

Algorithm 2 can now be applied to M' and N' to deduce that the smallest output channel capacity of M' is two and that the smallest output channel capacity of N' is three. (The smallest output channel capacity of the original $\bar{M}$, or $\bar{N}$, is four.)

Define the functionality of the network M' and N' as the set of all sequences of sending and receiving operations executed by the two machines starting from their initial state until they both return to nodes 1 and 1. Comparing the functionality of

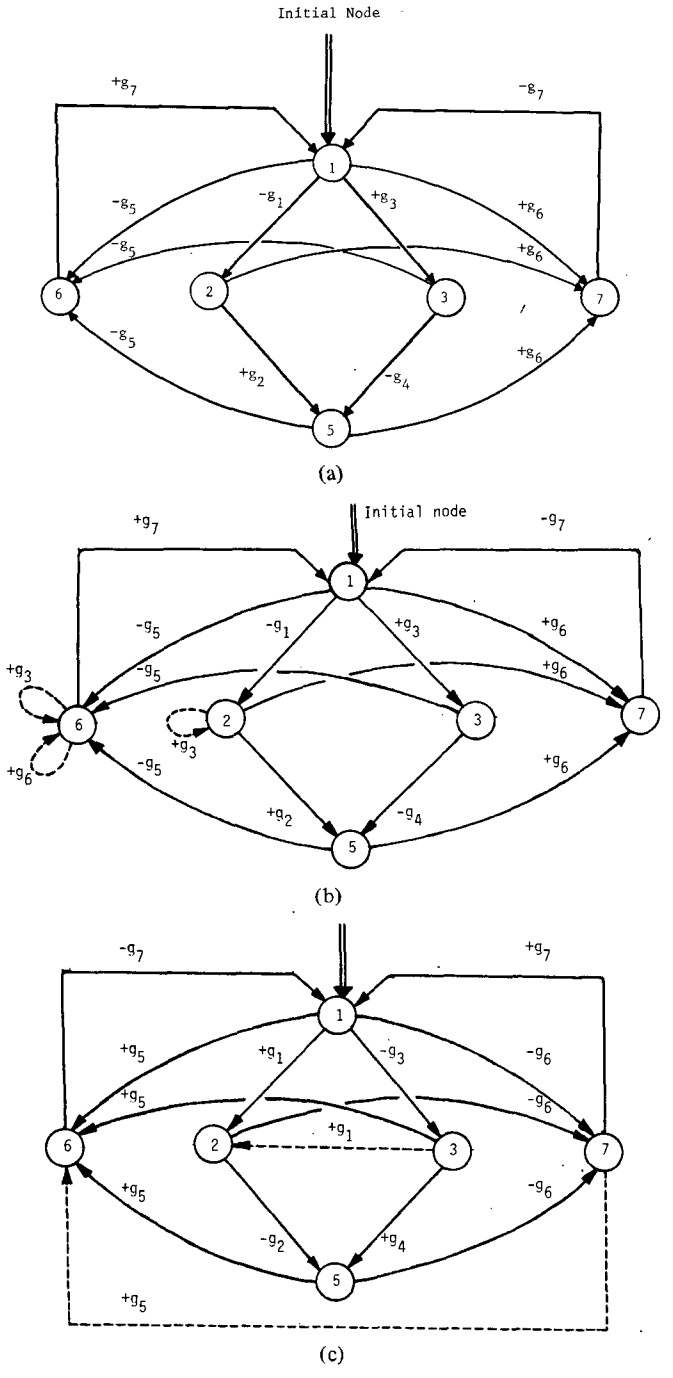

Fig. 7. Synthesizing a call establishment/clear protocol. [Notation: $-g$ = send (g), $+g$ = receive (g).] (a) M. (b) M'. (c) N'.

the constructed network M' and N' with that of the original network $\bar{M}$ and $\bar{N}$ yields the following observations.

1) For each of the "important" sequences of types 1, 2, and 3 for $\bar{M}$ and $\bar{N}$, there is an identical counter sequence for M' and N'.

2) For the sequence of type 4 for $\bar{M}$ and $\bar{N}$, there is no identical counter sequence for M' and N'. Instead, there is an "equivalent" sequence, namely:

$\langle M'$ sends g_5, N' sends g_6, M' receives g_6, N' receives g_5,

M' sends g_7, N' receives $g_7\rangle$

This sequence is equivalent to the type 4 sequence since in both sequences, the two machines try to close the connection and succeed.

3) For the sequence of type 5 for $\bar{M}$ and $\bar{N}$, there is no corresponding sequence for M' and N'.

From 3), the functionality of the constructed network M' and N' is a "proper subset" of the functionality of $\bar{M}$ and $\bar{N}$. After trying for some time, we feel that achieving all the functionality of $\bar{M}$ and $\bar{N}$, using our synthesis methodology, is impossible so long as the constructed machines M' and N' are expected to exchange only seven types of messages.

VI. Concluding Remarks

We have presented a two-algorithm synthesis methodology. The first algorithm takes one communicating machine M, and constructs two communicating machines M' and N' such that 1) M' is constructed from M by adding receiving edges to it, and 2) the communication between M' and N' satisfies some required progress properties. The second algorithm computes the smallest possible capacities for the two channels between M' and N'. It is straightforward to show that each algorithm requires a time of $O(st)$ where s is the number of nodes in the given machine M and t is the number of edges in M. The efficiency of these algorithms is the major advantage of our synthesis methodology.

The communication between the two constructed machines M' and N' has a fixed pattern. The communication proceeds in harmony until a loss of synchronization occurs at two dual mixed nodes in M' and N'. When a loss of synchronization is detected by both machines (not necessarily at the same time), then one machine (a loser) stops its current progress and rejoins the second machine, while the second machine (a winner) discards all the messages sent by the first machine during the loss of synchronization. Then, a harmonious communication between the two machines is resumed. This fixed pattern of communication is the major disadvantage of our synthesis methodology. For example, the methodology cannot synthesize the two communicating machines in Fig. 6(a) and (b) since their communication does not follow the above pattern. Instead, the methodology can synthesize the two functionally similar machines in Fig. 7 whose communication follows the above pattern.

It is useful to compare this synthesis methodology with the ZWRCB methodology [19] as they both share similar objectives.

1) The ZWRCB methodology supports a reasonably rich class of communication patterns, whereas our methodology supports one fixed communication pattern.

2) The ZWRCB methodology is based on generating and processing reachability trees to detect deadlocks and overflows. Therefore, it requires more execution time than our methodology.

3) The ZWRCB methodology is based on a trial-and-error principle, and so it can consume large amounts of execution time whenever the designer proceeds in erroneous directions. For instance, the designer may add some new sending transition to one of the two machines, and then later discover that this added transition will cause a deadlock, and so he removes it. By contrast, our methodology is deterministic, and so is not based on trial and error.

The discussion in this paper is limited to the case of two communicating finite-state machines. Extending the synthesis methodology to more than two machines is still an open problem that requires further research.

Acknowledgment

We are thankful to G. v. Bochmann, H. Rudin, and C. Sunshine for their helpful comments on an earlier version of this paper. We are also thankful to the referees whose suggestions have greatly improved the presentation.

REFERENCES

[1] G. v. Bochmann, "Finite state description of communication protocols," *Comput. Networks,* vol. 2, pp. 361–372, 1978.

[2] G. v. Bochmann and P. Merlin, "On the construction of communication protocols," in *Proc. 5th Int. Comput. Commun. Conf.,* Oct. 1980.

[3] G. v. Bochmann and C. Sunshine, "Use of formal methods in communication protocol design," *IEEE Trans. Commun.,* vol. COM-28, pp. 624–631, Apr. 1980.

[4] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *J. Ass. Comput. Mach.,* vol. 30, pp. 433–445, Apr. 1983.

[5] M. G. Gouda and W. Chu, "A finite state model for protocol processes and services," Dep. Comput. Sci., Univ. Texas at Austin, Tech. Rep. 198, Apr. 1982.

[6] M. G. Gouda and Y. T. Yu, "A methodology to design deadlock-free and bounded," presented at 1st Int. Workshop Protocol Testing and Verification, Brit. Nat. Phys. Labs, May 1981.

[7] ——,"Maximal progress state exploration," presented at ACM SIGCOMM '83, Univ. Texas at Austin, Mar. 1983.

[8] ——,"Protocol validation by maximal progress state exploration," *IEEE Trans Commun.,* vol. COM-32, pp. 94–97, Jan. 1984.

[9] P. Merlin and G. v. Bochmann, "On the construction of submodule specifications and communication protocols," *TOPLAS,* vol. 5, pp. 1–25, Jan. 1983.

[10] J. Rubin and C. H. West, "An improved protocol validation technique," *Comput. Networks,* vol. 6, Apr. 1982.

[11] H. Rudin and C. H. West, "A validation technique for tightly coupled protocols," *IEEE Trans. Comput.,* vol C-31, July 1982.

[12] C. A. Sunshine, "Formal modeling of communication protocols," Inform. Sci. Inst., Univ. Southern California, Los Angeles, Res. Rep. 81-89, Mar. 1981; also in *Computer Networks and Simulation II,* S. Schoemaker, Ed. New York: North-Holland, 1982.

[13] A. Tannenbaum, *Computer Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[14] C. H. West, "An automated technique of communication protocol validation," *IEEE Trans. Commun.,* vol. COM-26, pp. 1271–1275, Aug. 1978.

[15] Y. T. Yu, "Communicating finite state machines: Analysis and synthesis of communication protocols," Ph.D. dissertation, Dep. Comput. Sci., Univ. Texas at Austin, Jan. 1983.

[16] Y. T. Yu and M. G. Gouda, "Deadlock detection for a class of communicating finite-state machines," *IEEE Trans. Commun.,* vol. COM-30, pp. 2514–2518, Dec. 1982.

[17] ——,"Unboundedness detection for a class of communicating finite state machines," *Inform. Processing Lett.,* vol. 17, pp. 235–240, Dec. 1983.

[18] P. Zafiropulo, "Protocol validation by duologue-matrix analysis," *IEEE Trans. Commun.,* vol. COM-26, pp. 1187–1194, Aug. 1978.

[19] P. Zafiropulo, C. West, H. Rudin, D. D. Cowan, and D. Brand, "Towards analyzing and synthesizing protocols," *IEEE Trans. Commun.,* vol. COM-28, pp. 651–661, Apr. 1980.

★

Mohamed G. Gouda (S'76–M'77) received the B.Sc. degrees in engineering and mathematics from Cairo University, Cairo, Egypt, in 1968 and 1971, respectively, the M.A. degree in mathematics from York University, Toronto, Ont., Canada, in 1972, and the M.Math and Ph.D. degrees in computer science from the University of Waterloo, Waterloo, Ont., in 1973 and 1977, respectively.

From 1977 to 1980 he worked for the Honeywell Systems and Research Center and the Honeywell Corporate Technology Center, Minneapolis, MN. Since 1980 he has been an Assistant Professor in the Department of Computer Sciences at the University of Texas at Austin. His research interests include formal verification and synthesis of distributed systems and communication protocols.

★

Yao-Tin Yu was born in Taipei, Taiwan, R.O.C., in 1953. He received the B.S. degree in electrical engineering from the National Taiwan University in 1975 and the Ph.D. degree in computer science from the University of Texas at Austin in 1983.

Since 1983 he has been an Assistant Professor in the Department of Computer Science at the University of Iowa, Iowa City. His research interests include formal techniques for the analysis and synthesis of communication protocols.