Protocol Specification, Testing, and Verification, IV
Y. Yemini, R. Strom, and S. Yemini (Editors)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1985

ON CONSTRUCTING MULTI-PHASE COMMUNICATION PROTOCOLS

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

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

ABSTRACT

Many real-life protocols can be observed to go through different phases performing a distinct function in each phase. We present a multi-phase model for such protocols. A phase is formally defined to be a network of communicating finite state machines with certain desirable correctness properties; these include proper termination, and freedom from deadlocks and unspecified receptions. A multi-function protocol is constructed by first constructing separate phases to perform its different functions. We show how to connect these phases together to implement the multi-function protocol such that the resulting network of communicating finite state machines is also a phase (i.e. it possesses the desirable properties defined for phases). We also present a sufficient condition for the communication of a multi-phase protocol to be bounded given that its constituent phases have bounded communications.

The modularity inherent in multi-phase protocols facilitates not only their construction but also their understanding and modification. We found an abundance of real-life protocols that can be constructed as multi-phase protocols. Three fairly large examples are presented herein: (1) a version of IBM's BSC protocol for data link control, (2) a high-level session control protocol modeled after one in IBM's Systems Network Architecture, and (3) a token ring network protocol.

1. Introduction

A layered communications architecture facilitates the construction of networking software in a modular fashion. Nevertheless each protocol layer is a set of complex parallel programs. Several distinct functions can usually be identified among the tasks designated for a protocol layer to perform. For example, a full-duplex data link control protocol may be thought of as having at least three functions: connection management and simultaneous one-way data transfers in opposite directions. In both the analysis and construction of protocols, however, it is preferable to think about the individual functions of a multi-function protocol one at a time. In fact, most protocol analyses

[†] Work supported in part by National Science Foundation Grants ECS78-01803 and No. ECS83-04734, and in part by a research grant from IBM Corporation.

published in the literature have been illustrated with single-function protocols. For examples, both the alternating-bit protocol and Stenning's protocol that have been extensively analyzed in Bochmann [4], Stenning [30], and Hailpern and Owicki [16] are concerned with a one-way data transfer function only. The protocol analyses in Kurose and Yemini [18] and Razouk [26] are concerned with the connection management function only. Of interest to us are methods for reducing the analysis/construction of a multi-function protocol to the analyses/construction of smaller single-function protocols.

Suppose we are given a multi-function protocol to analyze. Lam and Shankar [19,20] presented a method for constructing "image protocols" for each function of the multi-function protocol. An image protocol is an abstraction of the original protocol but is specified like any real protocol. It is constructed to preserve all safety and liveness properties of the original protocol concerning one of its functions. Thus, their method reduces the analysis of a multi-function protocol to the analyses of several smaller single-function protocols. An application of their method to verify a version of the HDLC protocol is presented in [28,29].

This paper is concerned with the construction of a multi-function protocol from a composition of single-function protocols. In general, this is a difficult problem. However, many real-life protocols can be observed to go through different phases of behavior. In particular, these protocols go through their phases one at a time with a distinct function performed in each phase. For protocols characterized by this model of multi-phase behavior, the following three-step methodology for constructing a multi-function protocol is proposed:

i. Divide the protocol's functionality into separate functions.

ii. For each function, construct and verify a phase to perform this function. A phase is a network of communicating finite state machines that satisfies certain desirable general properties (including proper termination, and freedom from deadlocks and unspecified receptions) to be defined.

iii. Connect individual phases together to form the required protocol. The resulting protocol should satisfy the same general properties of proper termination, and freedom from deadlocks and unspecified receptions as the individual phases.

Step i of the above methodology is straightforward; the protocol's functions can often be divided quite naturally. For example, a half-duplex data link control protocol such as IBM's BSC protocol can be divided into three functions [17,21]: a call setup function, a data transfer function, and a call clear function.

For step ii of the methodolgy, there are two basic approaches. In the first approach, each phase is constructed based on the designer's knowledge and experience. It is then verified using available verification techniques, e.g. efficient reachability analysis in Bochmann [4], Rubin and West [27], Yu and Gouda [32,33], and Gouda and Yu [15], program proving methods in Good [10], Hailpern and Owicki [16], and Misra and Chandy [23,24], or symbolic execution in Brand and Joyner [2]. If an error is found in a phase, the phase is modified and the verification repeated, and so on until a provably correct phase is obtained. In the second approach, each phase is constructed using some constructive design rules that automatically result in correct phases. See for example Bochmann and Sunshine [5], Zafiropulo et al [34], Merlin and Bochmann [22], and Gouda and Yu [14].

Step iii of the methodology has received little attention so far, although it is agreed in Razouk and Estrin [25] and West and Zafiropulo [31] that many errors in a protocol are caused by improper connections between different phases of the protocol.

In this and other papers [7,8], we formally characterize the concept of a phase, and present a methodology to connect the different phases of a protocol to yield a protocol that satisfies the general correctness properties of proper termination, freedom from deadlocks and unspecified receptions, and also boundedness. We demonstrate how some realistic protocols can be constructed (and understood) in this fashion.

For simplicity, the discussion in this paper is carried out using the model of communicating finite state machines, although the results can be extended to other models as well. (The model of communicating finite state machines has been used successfully to model and analyze many existing protocols. See for instance Bochmann [4], West and Zafiropulo [31], Razouk and Estrin [25], and Gouda [12].)

This paper is organized as follows. In Section 2, the model of communicating finite state machines is presented. The concept of phases is briefly defined in Section 3. The construction of a protocol by connecting phases together is briefly discussed in Section 4; the construction method guarantees that the resulting multi-phase protocol terminates properly and is free from deadlocks and unspecified receptions. In Section 5, we outline a sufficient condition for boundedness of multi-phase protocols. The advantages of our construction methodology are discussed in Section 6. In Section 7, we list three multi-phase communication protocols that can be constructed by our methodology.

2. Networks of Communicating Finite State Machines

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

Let M and N be two communicating finite state machines with the same set G of messages; the pair (M,N) is called a *network* of M and N.

A *state* of network (M,N) is a four-tuple [v,w,x,y], where v and w are two nodes in M and N respectively, and x and y are two strings over the messages in G. Informally, a state [v,w,x,y] means that the executions of M and N have reached nodes v and w respectively, while the input channels of M and N store the strings x and y respectively.

The *initial state* of network (M,N) is $[v_0,w_0,E,E]$ where v_0 and w_0 are the initial nodes in M and N respectively, and E is the empty string.

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

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

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

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

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

The communication of a network (M,N) is said to *terminate properly* iff the following two conditions are satisfied:

- For any reachable state [v,w,x,y] of (M,N), if v is a final node of M, then x must be the empty string and there must be a directed path of all receiving edges from node w to a final node w' in N, where the string y is received.
- For any reachable state [v,w,x,y] of (M,N), if w is a final node of N, then y must be the empty string and there must be a directed path of all receiving edges from node v to a final node v' in M, where the string x is received.

A reachable state [v,w,E,E] of (M,N) is called a *proper terminating state* iff both node v and w are final nodes.

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

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

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

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

The communication of a network (M,N) is said to be ***bounded by*** K, where K is a nonnegative integer, iff for every reachable state [v,w,x,y] of (M,N), $|x| \leq K$ and $|y| \leq K$ where $|x|$ is the number of messages in string x. The communication is said to be ***bounded*** iff it is bounded by some nonnegative integer K; otherwise it is ***unbounded***.

3. Phases

Let M and N be two communicating finite state machines. The network (M,N) is called ***safe*** iff its communication terminates properly and is free from deadlocks and unspecified receptions.

Let (M,N) be a safe network, and let v and w be two final nodes in machines M and N respectively. The node pair (v,w) is called an ***exit node pair*** of (M,N) iff the state [v,w,E,E] of (M,N) is reachable.

The ***exit set*** of a safe network (M,N) is the set of all exit node pairs of (M,N).

A safe network (M,N) is called a ***phase*** iff every final node in M or N appears in exactly one exit node pair in the exit set of (M,N).

Consider the following problem. Is it decidable whether an arbitrary network is a phase? In general, this problem is undecidable as discussed in Brand and Zafiropulo [3], and in Gouda, Manning, and Yu [13]. However, the problem can be decided in some special cases: For instance, if the communication of the given network (M,N) is bounded, then the problem can be decided by generating and checking all the reachable states of (M,N). In [7], we discuss a technique, based on the concept of closed covers in Gouda [11], to verify that a given network is a phase even if the number of its reachable states is infinite.

4. Constructing Multi-Phase Networks

In this section we discuss a discipline to connect a number of phases together to construct a multi-phase network that is also a phase (thus guaranteeing that its communication terminates properly and is free from deadlocks and unspecified receptions). Phases are connected by joining the exit node pairs of one phase to the initial node pair of another phase, or the same phase. The validation of this discipline is proved in [7]. The technique is discussed next.

Let $p_1=(M_1,N_1)$ and $p_2=(M_2,N_2)$ be two phases, with exit sets S_1 and S_2 respectively, and let C be a subset of of S_1. We define a ***composite network*** of p_1, C, and p_2, denoted by $<p_1,C,p_2>$, to be the network (M,N) where

- M is the communicating finite state machine constructed (from M_1, C, and M_2) by joining all the final nodes of M_1 in C to the initial node of M_2. The initial node of M_1 becomes the initial node of M.
- N is the communicating finite state machine constructed (from N_1, C, and N_2) by joining all the final nodes of N_1 in C to the initial node of N_2. The initial node of N_1 becomes the initial node of N.

The two phases $p_1=(M_1,N_1)$ and $p_2=(M_2,N_2)$ are called the ***constituent phases*** of the composite network $<p_1,C,p_2>$. In this case, machines M_1 and M_2 are called the *constituent machines* of M, and machines N_1 and N_2 are called the *constituent machines* of

As an example, Figure 1a shows two phases $p_1=(M_1,N_1)$ and $p_2=(M_2,N_2)$. In phase p_1, the node pair (1,1) is its initial node pair and {(2,2),(3,3)} is its exit set. In phase p_2 the node pair (4,4) is its initial node pair and {(5,5)} is its exit set. By joining the exit node pair (2,2) of p_1 to the initial node pair (4,4), we have the composite network $p_{1,2}=<p_1,\{(2,2)\},p_2>$ shown in Figure 1b.

So far we have discussed how to connect one phase to another. Next, we discuss how to connect a phase to itself.

Let $p_1=(M_1,N_1)$ be a phase whose exit set is S_1, and let C be a subset of S_1. The *composite network* of p_1 and C, denoted $<p_1,C>$, is a network (M,N) where

- M is the communicating finite state machine constructed (from M_1 and C) by joining all the final nodes of M_1 in C to the initial node of M_1. The initial node of M_1 becomes the initial node of M.

- N is the communicating finite state machine constructed (from N_1 and C) by joining all the final nodes of N_1 in C to the initial node of N_1. The initial node of N_1 becomes the initial node of N.

Phase $p_1=(M_1,N_1)$ is called the *constituent phase* of the composite network $<p_1,C>=(M,N)$. In this case, machines M_1 and N_1 are called the *constituent machines* of M and N respectively.

For example, consider phase $p_{1,2}$ in Figure 1b, if we join the exit node pair (5,5) of $p_{1,2}$ to its initial node pair, then we get the composte phase $<p_{1,2},\{(5,5)\}>$ shown in Figure 1c.

The construction process of the multi-phase network p^* in Figure 1c from the two phases p_1 and p_2 in Figure 1a can be represented by the following sequence of equations:

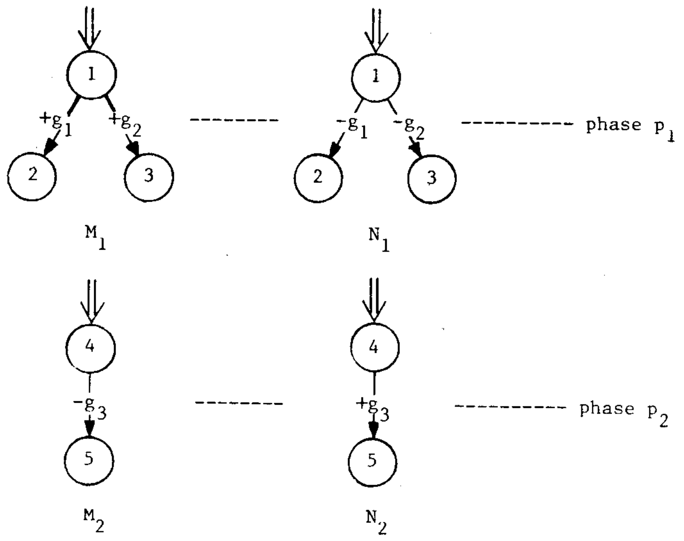
$$\begin{aligned} p_1 &=(M_1,N_1) \\ p_2 &=(M_2,N_2) \\ p_{1,2} &=<p_1,\{(2,2)\},p_2> \\ p^* &=<p_{1,2},\{(5,5)\}> \end{aligned}$$

This equation sequence clearly provides all the information needed to construct p^* from p_1 and p_2; moreover it is a more concise notation than the graphical representations in Figures 1b and 1c.

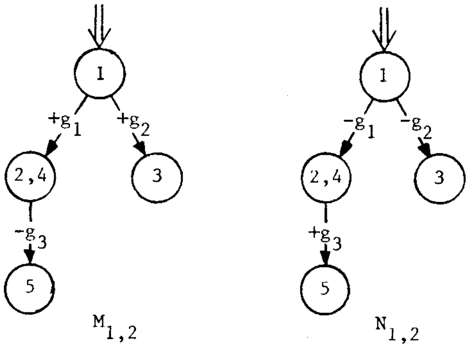
5. Boundedness of Multi-Phase Networks

In this section we present a sufficient condition for the communication of a composite network to be bounded provided that all its constituent phases are bounded. In order to state this sufficient condition, some definitions are in order.

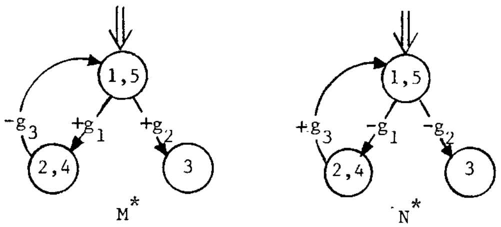
Let (M,N) be a composite network, i.e. both M and N consist of a finite number of



(a) Two phases $p_1 = (M_1,N_1)$ and $p_2 = (M_2,N_2)$



(b) The composite phase $p_{1,2} = < p_1, \{(2,2)\}, p_2 >$



(c) The composite phase $p^* = < p_{1,2}, \{(5,5)\} >$

Figure 1. An example for constructing multi-phase networks.

constituent machines. Two constituent machines in M, or N, are called *disjoint* iff they share no directed edges.

Let (M,N) be a composite network, and let M_i be a-constituent machine in M. A final node in M_i is called a ***plus node*** iff all its incoming edges are receiving deges. A final node in M_i is called a ***minus node*** iff all its incoming edges are sending edges. A final node in M_i is called a ***zero node*** iff its incoming edges include both sending and receiving edges.

Let (M,N) be a composite network, and assume that machine M consists of r ($r \geq 1$) mutually disjoint constituent machines $M_1,M_2,...,M_r$. An *abstract machine* $\widetilde{M}$ of M is a directed labelled graph constructed from M as follows:

- For each constituent machine M_i in M, add a node, also called M_i, to $\widetilde{M}$.
- If only plus nodes of a constituent machine M_i are joined with the initial node of a constituent machine M_j (M_j may be the same as M_i), then add a directed edge, labelled +, from node M_i to node M_j in $\widetilde{M}$.
- If only minus nodes of a constituent machine M_i are joined with the initial node of a constituent machine M_j (M_j may be the same as M_i), then add a directed edge, labelled -, from node M_i to node M_j in $\widetilde{M}$.
- If the nodes of a constituent machine M_i that are joined with the initial node of a constituent machine M_j include one zero node or include both plus nodes and minus nodes, then add two directed edges, one labelled -, the other labelled +, from node M_i to node M_j in $\widetilde{M}$.

A directed edge labelled + (-) in $\widetilde{M}$ is called a plus (minus) edge.

As an example, Figure 2 shows the abstract machine of the communicating machine M in Figure 1c.

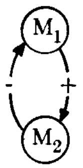


Figure 2. Abstract machine of machine M^* in Figure 1c.

Let (M,N) be a composite network whose constituent phases are all bounded; and let $\widetilde{M}$ be the abstract machine of M. The sufficient condition for the communication of (M,N) to be bounded is that each directed cycle in $\widetilde{M}$ must have at least one plus edge and one minus edge.

The abstract machine $\widetilde{M}$ in Figure 2 satisfies this condition, therefore the communication of the protocol example is bounded.

A correctness proof for the above sufficient condition is shown in [7].

6. Achieving Modularity

This paper proposes the following construction methodology for multi-phase communication protocols. First, construct the individual phases of the protocol independently, and ensure that each phase satisfies the desirable properties of phases as defined formally in section 3. Second, connect the resulting phases using the method discussed in Section 4. The result is a protocol that is guaranteed to terminate properly and to be free from deadlocks and unspecified receptions. Under some additional conditions, the resulting protocol is also bounded.

Advantages of this construction methodology are as follows:

- *Ease of Construction and Reasoning:* The methodology allows one to focus on only one phase of a complex protocol at a time. By ensuring that each phase satisfies some desirable properties, one is guaranteed that the phases can be later connected together to form a multi-phase protocol with the same desirable properties.

- *Parallel Construction and Verification:* Construction and verification of the different phases of a protocol can proceed independently and hopefully in parallel.

- *Flexibility for Modifying a Phase:* After constructing a protocol by connecting a number of phases together, it is possible to modify one of the phases without affecting the others. This is done by preserving the exit set in the modified phase. An example is shown in [7].

- *Flexibility for Rearranging Phases:* After constructing a protocol by connecting a number of phases together, it is possible to add more copies of the existing phases and rearrange the connections between phases to make the protocol satisfy some additional desirable properties such as fairness or robustness. An example is shown in [7].

- *Efficient Validation:* Many copies of the same phase may be used in a complicated protocol. (Examples are shown in [7].) The method of phases requires that such a phase is validated only once regardless of how many copies of it are used in the protocol. By constrast, validating the entire protocol will require validating each employed copy of such a phase.

7. Examples

We have applied this discipline to construct three multi-phase communication protocols [7,8]:

- *A version of IBM's BSC protocol for data link control [7]:* This protocol can be modeled as a network with five phases, a call setup phase, two data transfer phases, and two call clear phases.

- *A high-level session control protocol modeled after IBM's SNA specification for LU-LU session management [8,9]:* This example demonstrates that the concept of phases can be extended in a straightforward manner to networks with more than two communicating finite state machines. This protocol can be modeled as a network of three phases (a session establishment phase, a data transfer phase, and a session termination phase); each phase consists of three communicating finite state machines.

- *A token ring network protocol [7]:* This example demonstrates that the concept of phases can be extended to parameterized networks; i.e. networks with n communicating finite state machines, where n can take any positive integer value. This protocol can be modeled as a network with n identical phases; each phase consists of n communicating finite state machines.

REFERENCES

1. Andrews, D. W. and G. D. Schultz, "A token-ring architecture for local area networks: an update," *Proc. Compcon Fall 82*, pp. 615-624.

2. Brand, D. and W. H. Joyner, Jr., "Verification of protocols using symbolic execution," *Comput. Networks*, vol. 2, Oct. 1978, pp.351-360.

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

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

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

6. Bochmann, G. V. et al, "Experience with formal specifications using an extended state transition model," *IEEE Trans. on Commun.*, Dec. 1982, pp. 2506-2513.

7. Chow, C. H., M. G. Gouda, and S. S. Lam, "A discipline for constructing multi-phase communication protocols," *TR-233*, Department of Computer Sciences, University of Texas at Austin, June 1983. Revised, Oct. 1983.

8. Chow, C. H., M. G. Gouda, and S. S. Lam, "An exercise in constructing multi-phase communication protocols," *SIGCOMM '84*, Montreal, Canada, June 1984.

9. Cypser, R. J., "Communications architecture for distributed systems," Reading, Mass., Addison-Wesley, 1978.

10. Good, D. I., "Constructing verified and reliable communications processing systems," *ACM Software Eng. Notes*, vol. 2, Oct. 1977, pp.8-13.

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

12. Gouda, M. G., "An example for constructing communicating machines by step-wise refinement," *Proc. 3rd IFIP Workshop on Protocol Specification,*

Testing, and Verification, edited by H. Rudin and C. H. West, North-Holand, 1983, pp. 63-74.

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

14. Gouda, M. G. and Y. T. Yu, "Synthesis of communicating machines with guaranteed progress", *TR-179,* Dept. of Computer Sciences, Univ. of Texas at Austin, June 1981. Revised Jan. 1983. Revised Oct. 1983. To appear in *IEEE Trans. on Commun.*

15. Gouda, M. G. and Y. T. Yu, "Protocol validation by maximal progress state exploration", *TR-211,* Dept. of Computer Sciences, Univ. of Texas at Austin, July 1982. To appear in *IEEE Trans. on Commun.* Jan. 1984.

16. Hailpern, B. T. and S. S. Owicki, "Verifing network protocols using temporal logic," In *Proceedings Trends and Applications 1980: Computer Network Protocols,* IEEE Computer Society, 1980, pp. 18-28.

17. IBM Corp., "General Information--Binary Synchronous Communications," *Manual No. GA27-3004-2, 3rd. ed.,* Oct. 1970.

18. Kurose, J. and Y. Yemini, "The specification and verification of a connection establishment protocol using Temporal Logic," *Proc. 2nd Int. Workshop on Protocol Specification, Testing and Verification,* May 1982. Idyllwild, CA.

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

20. Lam, S. S. and A. U. Shankar, "Protocol verification via projections," *TR-207,* Dept. of Computer Sciences, Univ. of Texas at Austin, August, 1982. To appear in *IEEE Trans. on Software Engineering.*

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

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

23. Misra, J. and K. M. Chandy, "Proof of networks of processes," *IEEE Tran. on Software Eng.,* Vol. SE-7, No. 4, July 1981.

24. Misra, J., K. M. Chandy and T. Smith, "Proving safety and liveness of communicating processes with examples," *Proc ACM SIGACT-SIGOPS Symposium on Principls of Distributed Computing,* Aug. 1982, pp. 18-20.

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

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

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

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

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

30. Stenning, N. V., "A data transfer protocol," *Computer Networks*, vol. 1, Sept. 1976, pp. 99-110.

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

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

33. Yu, Y. T. and M. G. Gouda, "Unboundedness detection for a class of communicating finite-state machines," *TR-181*, Dept. of Computer Sciences, Univ. of Texas at Austin, Jan. 1983. To appear in *Information Processing Letters*.

34. Zafiropulo, P. et al, "Towards analyzing and synthesizing protocols," *IEEE Trans. on Commun.*, vol. COM-28, No. 4, April 1980, pp. 651-661.