AN EXERCISE IN CONSTRUCTING MULTI-PHASE COMMUNICATION PROTOCOLS

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

Department of Computer Sciences
University of Texas at Austin

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 discuss 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). A high-level session control protocol modeled after one in IBM's Systems Network Architecture is discussed, and constructed as a multi-phase 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. 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. Given a multi-function protocol, its analysis can be reduced to the analyses of several smaller single-function protocols, called image protocols, using the method of projections [LAM 81, LAM 82, SHAN 83].

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:

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

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 [IBM 70, LAM 83]: a call setup function, a data transfer function, and a call clear function.

For step ii of the methodology, 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 [BOCH 78, RUBI 82, YU 82, YU 83, GOUD 82c], program proving methods [GOOD 77, HAIL 80, MISR 81, MISR 82], or symbolic execution [BRAN 78]. 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 [BOCH 80, ZAFI 80, MERL 83, GOUD 81].

Step iii of the methodology has received little attention so far, although it is agreed in [RAZO 80, WEST 78] that many errors in a protocol are caused by improper connections between different phases of the protocol.

In this paper, 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 one realistic protocol can be constructed (and understood) in this fashion. Details of our theory and two other examples can be found in [CHOW 83].

This paper is organized as follows. In Section 2, the model of communicating finite state machines is presented. The concept of phases is formally defined in Section 3. The construction of a protocol by connecting phases together is 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 present a multi-phase protocol example, namely a high-level session control protocol modeled after IBM's SNA specification for LU-LU session management [CYPS 78]. Concluding remarks are in Section 6.

2. Networks of Communicating Finite State Machines

For the sake of conciseness, we present our theory (Sections 2, 3 and 4) using networks of two communicating finite state machines. However, our results can be extended in a straightforward manner to networks of n communicating finite state machines [CHOW 83].

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:

i. 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.

ii. e is a sending edge, labelled -g, form w to w' in N, and s'=[v,w',x.g,y].

iii. e is a receiving edge, labelled +g, from v to v' in M, and s'=[v',w,x',y], where x=g.x'.

iv. 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:

i. 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.

ii. 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:

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

ii. $y=g_1.g_2. \dots .g_k$ $(k\geq 1)$, 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 [BRAN 83, GOUD 82b]. 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 [CHOW 83], we discuss a technique, based on the concept of closed covers in [GOUD 82a], 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 technique is discussed in detail 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

i. 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.

ii. 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 N. It is proved in [CHOW 83] that the composite network $<p_1,C,p_2>$ is also a phase whose exit set is $(S_1 \cup S_2) - C$.

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

i. 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.

ii. 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. It is proved in [CHOW 83] that the composite network $<p,C>$ is also a phase whose exit set is S − C.

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 composite network $<p_{1,2},\{(5,5)\}>$ 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. A Session Protocol Example

The concept of phases can be extended in a straightforward manner to networks with r ($r \geq 2$) communicating finite state machines. As an example, we construct in this section a high-level session control protocol modeled after IBM's SNA specification for LU-LU session management [CYPS 78]. It is a multi-phase network of three communicating finite state machines. We discuss next how to extend earlier definitions to the case of networks with three communicating machines.

Consider a network of three communicating finite state machines. Each machine in the network can communicate with each of the other machines by exchanging messages over two, one-directional unbounded FIFO channels. In order to uniquely specify which machine is intended by each sending (receiving) operation, a sending (receiving) edge is now labelled -g/M (+g/M), where M is the machine to (from) which the message g is sent (received).

Let M_1, M_2 and M_3 be three communicating finite state machines with the same set G of messages; the triple (M_1,M_2,M_3) is called a *network* of the three machines. A *state* of the network (M_1,M_2,M_3) is a 3x3 matrix s_{ij}, where the component s_{ii} is a node in M_i,i=1,2,3, and the component s_{ij} ($i \neq j$) is a string over the messages in G representing the contents of the channel from machine M_i to M_j. The definitions of reachable states, proper termination, freedom from deadlocks and unspecified receptions, boundedness and phases can now be extended in a straightforward manner to networks with three communicating finite state machines.

Next, we discuss the basic phases of the session protocol, where each phase is a network of three machines. Later, we discuss how to connect these phases to construct the protocol.

Session Establishment Phase

Consider the three communicating finite state machines L_1,M_1 and N_1 in Figure 2; they model the IBM LU-LU session establishment procedure in a single domain [CYPS 78]: L_1 models the primary logical unit, M_1 models the session controller, and N_1 models the secondary logical unit. The exchanged messages have the following meanings:

INIT	denotes a "request to set up a session" message.
RSP	denotes a "positive response" message.
NRSP	denotes a "negative response" message.
CINIT	denotes a "control initiate" message.
BIND	denotes a "bind session" message.
BINDF	denotes a "bind failure" message.
SST	denotes a "session started" message.
SDT	denotes a "start data traffic" message.

Starting from node 1, if L_1 (or N_1) wants to setup a session with N_1 (or L_1), it sends an INIT message to the controller M_1. M_1 may reject the request by sending back an NRSP message, that contains the rejection reasons. If M_1 accepts the request, it sends back an RSP message to the sender (L_1 or N_1), then sends a CINIT message to the primary L_1. On receiving CINIT, the primary L_1 may reject the controller's request by sending an NRSP message back to M_1. Or it may accept the request by sending an RSP message back to M_1 followed by a BIND message, that contains the proposed session parameters, to machine N_1. There are two possibilities:

i. *Machine N_1 accepts the session parameters proposed by L_1:* In this case, N_1 sends an RSP message to L_1. L_1 then notifies M_1 about the success of the session setup by sending an SST message. After L_1 gets an RSP message from M_1, L_1 sends an SDT message to N_1. When N_1 is ready for the data transfer, it sends an RSP message back to L_1 and the data transfer between L_1 and N_1 begins without the intervening of M_1.

ii. *Machine N_1 does not accept the session parameters proposed by L_1:* In this case, N_1 sends an NRSP message to L_1. L_1 then notifies M_1 about the failure of the session setup by sending a BINDF message, and L_1, M_1 and N_1 return to their initial nodes.

Using state exploration, it can be shown that network (L_1,M_1,N_1) is a phase whose exit set is {(14,13,7)}.

Data Transfer Phase

Consider the three communicating finite state machines L_2, M_2, and N_2 in Figure 3; they model a simplified half-duplex flip-flop data transfer procedure in the IBM session protocol: L_2 models the primary logical unit, M_2 models the session controller, and N_2 models the secondary logical unit. The exchanged messages have the following meanings:

D	denotes a data message.
RSP	denotes a "positive response" message.
SHUT	denotes a "shutdown" message.
SHUTC	denotes a "shutdown complete" message.

Since machine M_2 is not involved in the data transfer procedure, it is modeled as a single node without any outgoing edges, as shown in Figure 3.

Starting from their initial nodes, L_2 and N_2 take turns to send data and wait for a response. When the primary L_2 wants to terminate the data transfer, it can send at node 1 a SHUT message to N_2. In this case N_2 completes its session processing, then sends a SHUTC message to L_2 and each of L_2 and N_2 reaches its final node. It is straightforward to show that (L_2,M_2,N_2) is a phase whose exit set is {(6,1,6)}.

Session Termination Phase

Consider the three communicating finite state machines L_3, M_3 and N_3 in Figure 4; they model a simplified version of the session termination procedure: L_3 models the primary logical unit, M_3 models the session controller, and N_3 models the secondary logical unit. The exchanged messages have the following meanings:

TERM	denotes a "request to terminate a session" message.
RSP	denotes a "positive response" message.
CTERM	denotes a "control terminate" message.
UNBIND	denotes a "unbind session" message.
SE	denotes a "session ended" message.

Starting from its initial node, if L_1 (or N_1) wants to terminate the current session, it sends a TERM message to M_3. When M_3 receives the TERM message, it first responds with an RSP message, and then sends a CTERM message to the primary logical unit L_3. L_3 responds with an RSP message and then sends an UNBIND message to notify N_3 to end the session. On receiving the response to the UNBIND message from N_3, L_3 sends an SE message to notify the controller M_3 that the session between L_3 and N_3 has been ended . M_3 replies with an RSP message and the session termination procedure is then complete.

Note that the three small loops at nodes 4, 8 and 9 of M_3 are dealing with the message collision situations where both L_3 and N_3 concurrently send TERM messages to M_3.

It is straightforward to show that (L_3,M_3,N_3) is a phase whose exit set is $\{(11,12,5)\}$.

Constructing a Multi-Phase Session Protocol

The following equation sequence represents a multiphase session protocol that consists of the three phases defined earlier (namely, one session establishment phase, one data transfer phase and one session termination phase).

$p_1 \quad =(L_1,M_1,N_1)$
$p_2 \quad =(L_2,M_2,N_2)$
$p_3 \quad =(L_3,M_3,N_3)$
$p_{1,2} \quad =<p_1,C_1,p_2>$
$p_{1,2,3} \quad =<p_{1,2},C_2,p_3>$
$p \quad =<p_{1,2,3},C_3>$

where L_1, M_1 and N_1 are defined in Figure 2
L_2, M_2 and N_2 are defined in Figure 3
L_3, M_3 and N_3 are defined in Figure 4
$C_1=\{(14,13,7) \text{ in } p_1\}$
$C_2=\{(6,1,6) \text{ in } p_2\}$
$C_3=\{(11,12,5) \text{ in } p_3\}$
where (i,j,k) in $p_l=$ the node triple (i,j,k) in phase p_l.

The above multi-phase protocol is guaranteed to terminate properly and be free from deadlocks and unspecified receptions.

6. Concluding Remarks

We have proposed a construction methodology for large multi-phase communication protocols, and demonstrated that this methodology can be used to construct (and explain) some realistic protocols. The resulting protocols are guaranteed to terminate properly and be free from deadlocks and unspecified receptions. More results and examples about the construction of multi-phase protocols can be found in [CHOW 83].

Although the phase concept and the proposed methodology are discussed using communicating finite state machines, it is straightforward to extend the results to facilitate protocol construction using other models such as the extended state transition model of Bochmann et al [BOCH 82].

REFERENCES

[BOCH 78] Bochmann, G. V. "Finite state description of communication protocols," *Computer Networks*, Vol. 2, 1978, pp. 361-371.

[BOCH 80] Bochmann, G. V. and C. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. on Commun.*, April 1980, pp.624-631.

[BOCH 82] Bochmann, G. V. et al, "Experience with formal specifications using an extended state transition model," *IEEE Trans. on Commun.*, Dec. 1982, pp. 2506-2513.

[BRAN 78] Brand, D. and W. H. Joyner, Jr., "Verification of protocols using symbolic execution," *Comput. Networks*, vol. 2, Oct. 1978, pp.351-360.

[BRAN 83] Brand, D. and P. Zafiropulo, "On communicating finite-state machines," *JACM*, Vol. 30, No. 2, April 1983, pp. 323-342.

[CHOW 83] C. H. Chow, M. G. Gouda, and S. S. Lam, "A discipline for constructing multiphase communication protocols," TR-233, Dept. of Computer Sciences, Univ. of Texas at Austin, June 1983. Revised Oct. 1983. Submitted for publication.

[CYPS 78] Cypser, R. J., "Communications architecture for distributed systems," Reading, Mass., Addison-Wesley, 1978.

[GOOD 77] Good, D. I., "Constructing verified and reliable communications processing systems," *ACM Software Eng. Notes*, vol. 2, Oct. 1977, pp.8-13.

[GOUD 81] 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, Oct. 1983. To appear in *IEEE Trans. on Commun.*

[GOUD 82a] 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.*

[GOUD 82b] 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.

[GOUD 82c] 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.

[GOUD 83] 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-Holland, 1983, pp. 63-74.

[HAIL 80] Hailpern, B. T. and S. S. Owicki, "Verifying network protocols using temporal logic," In *Proceedings Trends and Applications 1980: Computer Network Protocols,* IEEE Computer Society, 1980, pp. 18-28.

[IBM 70] IBM Corp., "General Information--Binary Synchronous Communications," *Manual No. GA27-3004-2, 3rd. ed.*, Oct. 1970.

[LAM 81] Lam, S. S. and A. U. Shankar, "Protocol projections: a method for analyzing communication protocols," *Conf. Rec. National Telecommunications Conference*, Nov. 1981, New Orleans.

[LAM 82] 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.

[LAM 83] Lam, S. S., "Data link control procedures," in *Computer Communications*, Vol. 1, ed. W. Chou, Prentice-Hall, Englewood Cliffs, 1983, pp. 81-113.

[MERL 83] 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.

[MISR 81] Misra, J. and K. M. Chandy, "Proof of networks of processes," *IEEE Tran. on Software Eng.*, Vol. SE-7, No. 4, July 1981.

[MISR 82] Misra, J., K. M. Chandy and T. Smith, "Proving safety and liveness of communicating processes with examples," *Proc ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Aug. 1982, pp. 18-20.

[RAZO 80] Razouk, R. R. and G. Estrin, "Modeling and verification of communication protocols in SARA: The X.21 interface," *IEEE Trans. on Comput.*, vol. C-29, No. 12, Dec. 1980, pp. 1038-1052.

[RUBI 82] Rubin, J. and C. H. West, "An improved protocol validation technique," *Computer Networks*, April 1982.

[SHAN 83] 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.

[WEST 78] 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.

[YU 82] 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.

[YU 83] Yu, Y. T. and M. G. Gouda, "Unboundedness detection for a class of communicating finite-state machines," *Information Processing Letters*, Vol. 17, Dec. 1983, pp. 235-240.

[ZAFI 80] Zafiropulo, P. et al, "Towards analyzing and synthesizing protocols," *IEEE Trans. on Commun.*, vol. COM-28, No. 4, April 1980, pp. 651-661.

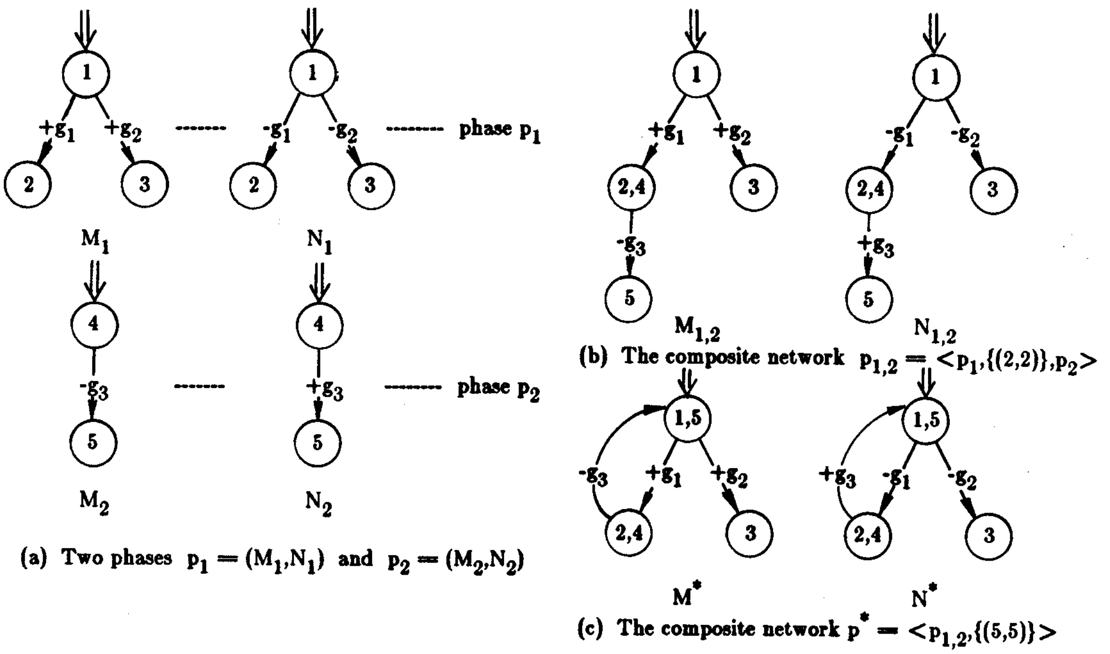

Figure 1. An example for constructing multi-phase networks.

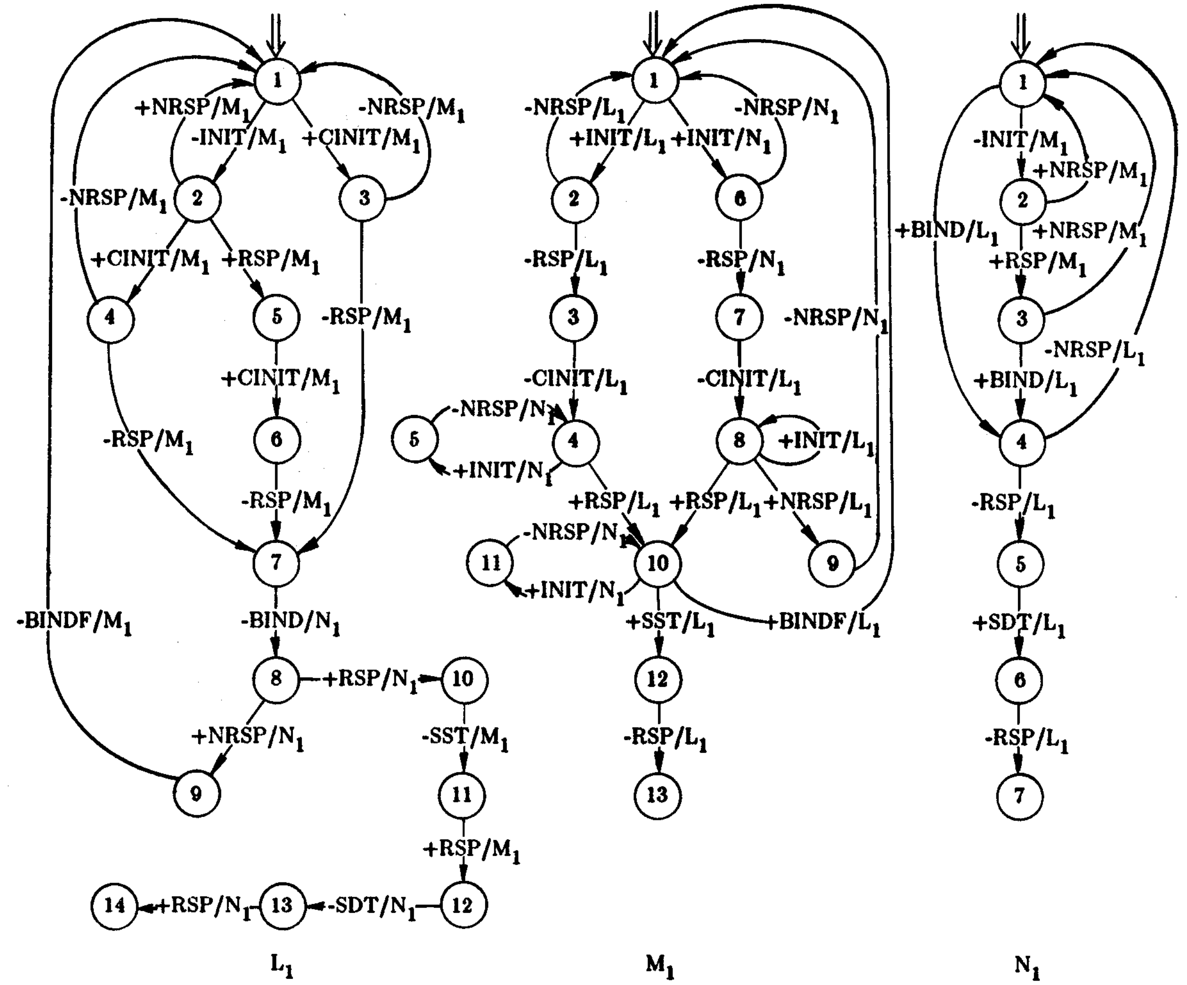

Figure 2. Establishment Phase of the Session Protocol.

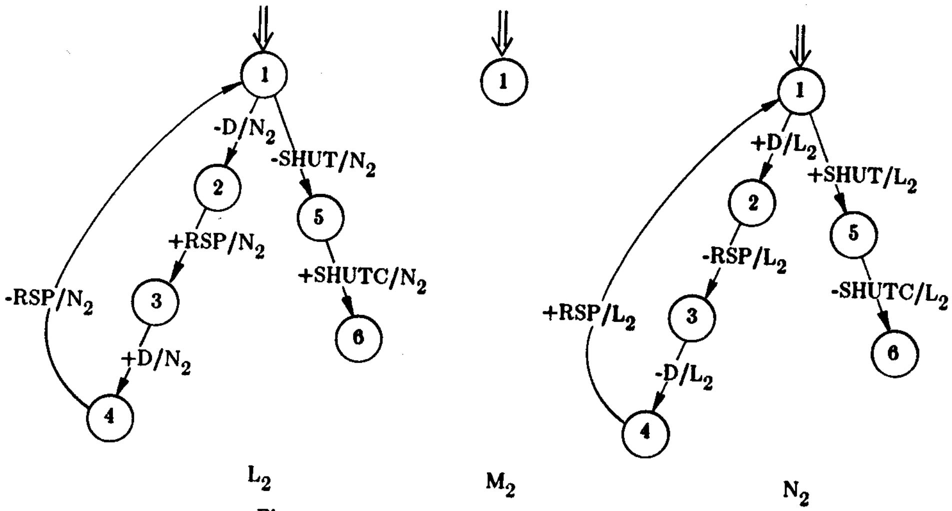

Figure 3. Data Transfer Phase of the Session Protocol.

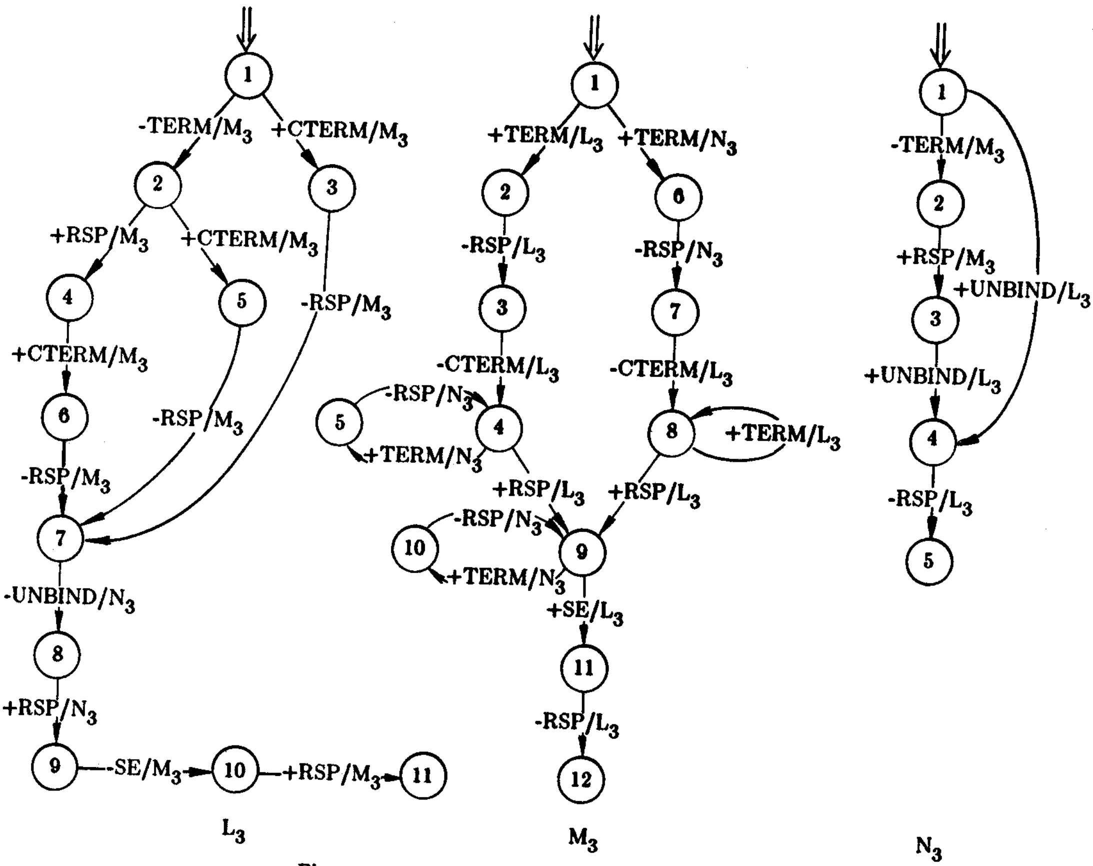

Figure 4. Termination Phase of the Session Protocol.