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

ON THE DECIDABILITY OF LIVELOCK DETECTION IN NETWORKS OF COMMUNICATING FINITE STATE MACHINES

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

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

ABSTRACT

A livelock arises in a communication protocol if the communicating entities in the protocol keep on exchanging messages while no "useful work" is being done. We investigate this phenomenon in networks of communicating finite state machines. In particular, we show that it is undecidable in general whether the communication of any such network can reach a livelock. We also identify some special classes of networks for which livelock detection is decidable.

1. Introduction

The model of communicating finite state machines is useful in the specification [4,27,28], analysis [2,5,6,9,11,12,14,19,20,23,25,26,27,28,29,31,32,33,34] and synthesis [3,7,10,13,22] of communication protocols. The procedure for modeling and analyzing a communication protocol using this model typically proceeds as follows:

- First, the protocol is defined as a network of communicating finite state machines: Each machine in the network has a finite number of states and state transitions (called nodes and edges respectively in this paper). Each state transition of a machine is accompanied by the sending of a message into a channel or receiving of a message from a channel. Channels are assumed to be directional from one machine to another and FIFO. In general, a network may have an arbitrary topology.

- Second, the network defined is analyzed to ensure that its communication satisfies some nice properties such as boundedness [33], freedom from deadlocks [25,26,31,32] and freedom from unspecified receptions [11].

Examples of some realistic protocols that can be modeled and analyzed using this procedure include: the alternating-bit protocol [1], the Binary Synchronous protocol [24], and the call establishment/clearing procedures in X.21 [23,31] and X.25 [12,24].

† Work supported by National Science Foundation Grant No. ECS83-04734

One property that should be satisfied by networks of communicating finite state machines modeling real protocols, is freedom from livelocks. A livelock occurs when the machines in a network keep exchanging messages but no "useful work" is being done. The characterization of livelocks in distributed systems and communication protocols was considered by Hajek [16] and Lai [18]. Sherman and Rudin [29] characterized livelocks in networks of communicating finite state machines and pointed out the importance of detecting livelocks in such networks in protocol validation. In this paper we extend the work of Sherman and Rudin by showing that the livelock detection problem is undecidable for a general network of communicating finite state machines. We also list some special classes of networks for which the problem is decidable. Efficient algorithms for livelock detection in such networks are presented in [15].

This paper is organized as follows: Networks of communicating finite state machines are presented formally in Section 2. The concept of livelocks in such networks is defined in Section 3. In Section 4, we define the livelock detection problem and show that it is undecidable for general networks of communicating finite state machines. Finally in Section 5, we identify special network classes for which livelock detection is decidable.

2. Networks of Communicating Finite State Machines

A *communicating finite state machine* M is a labelled directed 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*) *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. Let (M,N) denote the network consisting of machines M and N connected by two FIFO channels in opposite directions.

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 denotes 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 be free from deadlocks and unspecified receptions, and to be bounded.

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:

- $x=g_1.g_2.\ \cdots\ .g_k$ ($k\geq 1$), and v is a receiving node and none of its outgoing edges is labelled $+g_1$.
- $y=g_1.g_2.\ \cdots\ .g_k$ ($k\geq 1$), 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 input channel of machine M (N) in network (M,N) is said to be *bounded by* K iff for every reachable state [v,w,x,y] of (M,N), $|x|\leq K$ ($|y|\leq K$). A channel in (M,N) is said to be *bounded* iff it is *bounded by* K, for some nonnegative integer K. The communication of (M,N) is said to be bounded (bounded by K) iff each of the two channels in (M,N) is bounded (bounded by K).

3. Livelocks

A *marked network* is a triple (M,N,m), where (M,N) is a network of two communicating finite state machines M and N, and m is a function, called the *marking* of the network. m assigns to each edge in M or N either the value "p" or the value "n". Let e be an edge in machine M or N. If m(e)=p then e is called a *progress edge*, otherwise m(e)=n, and e is called a *nonprogress edge.*

Let (M,N,m) be a marked network and let C and D be two directed cycles in M and N respectively. The pair (C,D) is called a *livelock* in (M,N,m), iff the following three conditions are satisfied:

i. All the edges of cycle C ,in M, are nonprogress.

ii. All the edges of cycle D ,in N, are nonprogress.

iii. There exists a sequence $(s_1,...,s_r)$ of reachable states of network (M,N) such that the following two conditions hold:

   a. For i=1,...,r-1, state s_{i+1} follows s_i over an edge e_i in M or N. Also state s_1 follows s_r over an edge e_r in M or N.

   b. The set of edges $\{e_1,e_2,...,e_r\}$ constitutes the two cycles C and D.

This sequence $(s_1,...,s_r)$ is called a *nonprogress cycle* for the livelock (C,D). (Notice that a livelock may have more than one nonprogress cycle.)

Example (A Buffer Allocation Protocol): In a store-and-forward (SF) network, each SF node has a finite number of buffers to store received packets before forwarding them to another SF node in the network. A buffer is allocated for a packet when it is received and only released when a positive acknowledgement message of the packet is received from its next SF node. A store-and-forward livelock can occur in the absence of an effective buffer management strategy [17]. We illustrate such livelock, in the simple case of two SF nodes that communicate via a full duplex channel; each SF node has two buffers which can store at most two data packets.

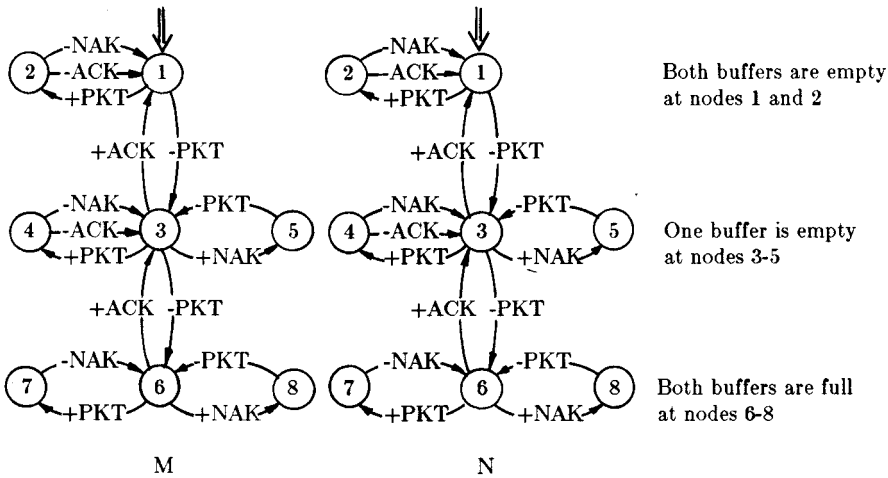

Figure 1. A buffer allocation protocol based upon contention.

Consider the two communicating finite state machines M and N in Figure 1. Each machine models a SF node with two buffers. The exchanged messages between M and N have the following meanings:

PKT denotes a data packet.
ACK denotes a positive acknowledgement.
NAK denotes a negative acknowledgement.

Starting from node 1, M can either receive a data packet or send out a data packet. After receiving a data packet, M sends back a ACK or NAK message to

machine N and returns to node 1. If M sends out a data packet, it reaches node 3. At node 3, one buffer is used to store a sent (and unacknowledged) packet, therefore only one buffer remains available. There are four possibilities:

- *M sends a second data packet to N:* M uses up the remaining buffer and reaches node 6.
- *M receives a data packet:* M stores the packet in the remaining buffer, sends back an ACK message to N, then releases the buffer.
- *M receives an ACK message:* M returns to node 1, since its two buffers are available now.
- *M receives an NAK message:* M enters node 5 and retransmits the data packet stored in the buffer.

At node 6, the two buffers in M are used to store the two sent and unacknowledged packets. If a new data packet from N is received, it has to be discarded, since there is no buffer available to store it, and M sends a NAK messsage back to N. If a NAK message is received, M retransmits the data packet stored in the buffer. If a ACK message is received, M returns to node 3 since one buffer becomes available. (It is straightforward to extend this model to the situation of n buffers. We simply repeat the structure in the second row of machine M, n-2 times and connect them together.)

One natural marking m for this protocol is as follows: All the edges labelled +ACK in M or N are marked progress, while all the other edges are marked non-progress. Let C_1 (D_1) be the directed cycle in M (N) that starts at node 6, goes to node 7, then returns to node 6. Also, let C_2 (D_2) be the directed cycle in M (N) that starts at node 6, goes to node 8, then returns to node 6. It is straightforward to show that (C_1,D_2) and (C_2,D_1) are livelocks of the marked network (M,N,m).

The livelock (C_1,D_2) represents the situation where the two machines have used up all their buffers for outgoing packets. Each expects the other to send a positive acknowledgement message. One way to prevent such livelocks is to require buffer reservation for each direction before any data transfer begins, or by separating packets into different priority classes based upon "hop count" [30].

□

4. Undecidability of the Livelock Detection Problem

The *livelock detection problem* can be stated as follows: Decide for any given marked network whether it is free from livelocks. The next theorem shows that this problem is undecidable for a general network of communicating finite state machines. (Actually, as the proof shows, livelock detection is undecidable even if we restrict the machines to those that send only two types of messages and have no mixed nodes.)

Theorem 1: It is undecidable whether any arbitrary marked network (M,N,m) can reach a livelock.

□

Proof:

We show that if the livelock detection problem can be decided, then the undecidable halting problem of Post machines can be decided.

A *Post machine* P is a finite directed graph with a string z over the symbols in {0,1,#}. Each vertex in the directed graph is labelled with a statement. The arcs in the directed graph represent the execution order among statements. There are four types of statements (shown in Figure 2):

- The *start* statement: It is the first statement executed by the Post machine and has no effect on string z. See Figure 2a.
- *assignment* statements: These statements concatenate a symbol, namely 0, 1 or #, to the right of string z. See Figure 2b.
- *test* statements: These statements check the leftmost symbol of string z, namely head(z), and delete it after making decision. Since the test result has four possibilities, there are four outgoing arcs in each vertex labelled with the test statement. See Figure 2c.
- The *halt* statement: This statement is the last statement executed by the Post machine before halting. See Figure 2d.

Let P be any given Post machine. The computation of P can be simulated by a marked network (M,N,m) that is constructed as follows:

- Machines M and N exchange four types of messages, namely 0, 1, # and $.
- Machine M sends every message it receives from N.
- Machine N simulates the actions of the Post machine P. It is constructed by applying the following four transformation rules to the directed graph of P:
  a. The start statement is transformed to an arrow which indicates the initial node in N. See Figure 2e.
  b. An assignment statement z <-- z.g, where g is a symbol in {0,1,#}, is transformed to a sending node with an outgoing edge labelled -g in N. See Figure 2f.
  c. A test statement is transformed to a structure shown in Figure 2g. Informally, with this structure, machine N simulates the test of empty(z) by sending the special message $ to M, then it waits to receive a message from M. If it receives the same message $, it recognizes that z is empty. If it receives other message, it recognized that z is not empty. In this case, N removes the message $ then waits to receive the next message and depending on its type, the execution of N proceeds along one of the three outgoing edges.
  d. The halt statement is transformed to the structure shown in Figure 2h.
- The marking m of (M,N,m) is as follows: All the edges of machine M are marked nonprogress and all the edges of machine N are marked progress except those edges which form the cycle C in the structures corresponding to the halt statement.

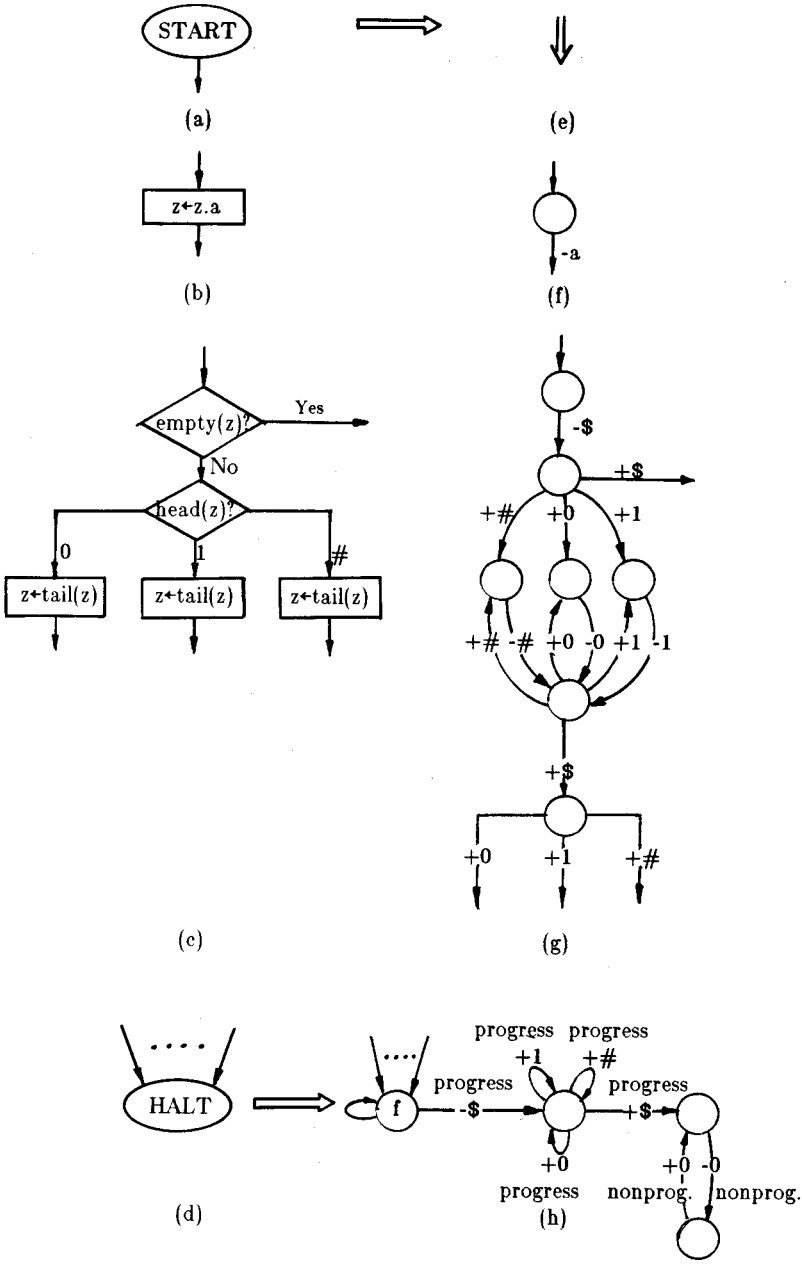

Figure 2. Reducing a Post machine to communicating finite state machines.

It is clear from this simulation that P reaches the halting statement iff the marked network (M,N,m) reaches a livelock. This completes the proof. (Notice that the machines in the simulated network exchange four types of messages, but these can be simulated

using only two types of messages. Notice also that the machines have no mixed nodes.) □

5. Classes of Networks with Decidable Livelock Detection Problem

In [15], we present efficient algorithms to detect livelocks in the following classes of networks:

- Networks of two machines with bounded communication.
- Networks of two machines where one of the two channels is bounded.
- Networks of two machines where one machine sends a single message type.
- Networks with any number of machines that have closed covers [9].

The algorithm for livelock detection for each of these special network classes is based on constructing an abstract representation of the reachability graph of a given network. Livelock detection is accomplished by an examination of this abstract representation (in a finite time) instead of the entire reachability graph. In each case, the constructed abstract representation is finite for any given network in its class, even if the reachability graph of this network is infinite (i.e. even if the communication of the given network is unbounded). For a complete discussion of these abstract representations and the specified algorithms, we refer the reader to [15].

REFERENCES

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

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

3. Bochmann, G. V. and C. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. on Commun.*, Vol. COM-28, No. 4, April 1980, pp.624-631.

4. Bochmann, G. V. et al, "Experience with formal specifications using an extended state transition model," *IEEE Trans. on Commun.*, Vol. COM-30, No. 12, Dec. 1982, pp. 2506-2513.

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

6. Chang, C. K. and M. G. Gouda, "Proving liveness for networks of communicating finite state machines," *TR-84-04*, Dept. of Computer Sciences, Univ. of Texas at Austin, Feb. 1984.

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

8. Chow, C. H., M. G. Gouda, and S. S. Lam, "An exercise in constructing multi-phase communication protocols," *Proc. of SIGCOMM'84 Symposium*,

June 1984.

9. 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.*

10. 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.

11. 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.

12. Gouda, M. G. and Y. T. Yu, "Protocol validation by maximal progress state exploration", *IEEE Trans. on Commun.* Vol. COM-32, No. 1, Jan. 1984, pp. 94-97.

13. Gouda, M. G. and Y. T. Yu, "Synthesis of communicating machines with guaranteed progress", To appear in *IEEE Trans. on Commun.*, July 1984.

14. Gouda, M. G. and L. E. Rosier, "Communicating finite state machines with priority channels," *Proc. of the Eleventh International Colloqium on Automata, Languages, and Programming (ICALP)*, 1984.

15. Gouda, M. G., C. H. Chow and S. S. Lam, "Livelock detection in networks of communicating finite state machines," *TR-84-10*, Dept. of Computer Sciences, Universty of Texas at Austin, March 1984.

16. Hajek, J. "Automatically verified data transfer protocols," *Proc. of 4th International conference on computer communication,* Kyoto, Sept. 1978, pp. 749-756.

17. Kahn, R. E. and W. R. Crowther, "Flow control in a resource-sharing computer network," *IEEE Trans. Commun.*, vol. COM-20, pp. 539-546, June 1972.

18. Lai, W. S., "Protocol traps in computer networks -- a catalog," *IEEE Trans. on Commun.*, vol. COM-30, No. 6, pp. 1434-1450, June 1982.

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," To appear in *IEEE Trans. on Software Engineering,* July 1984.

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. 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.

24. 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.

25. Rosier, L. E. and M. G. Gouda, "Deciding progress for a class of communicating finite state machines," *Proc. of Conference on Information Sciences and Systems*, Princeton University, 1984.

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

27. 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.

28. 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, Nov. 1983, pp. 331-368.

29. Sherman, M. and H. Rudin, "Using automated validation techniques to detect lockups in packet-switched networks," *IEEE Trans. on Commun.*, Vol. COM-30, No. 7, July 1982, pp. 1762-1767.

30. Tanenbaum, A. S., "Computer Networks," Prentice-Hall, Englewood Cliffs, N.J., 1981.

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.*, Vol. COM-30, No. 12, Dec. 1982, pp. 2514-2519.

33. 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.

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