MODELING AND VERIFICATION OF A BIT-STUFFING PROTOCOL USING COMMUNICATING FINITE STATE MACHINES

Mohamed G. Gouda

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

TR-84-32 October 1984

Abstract

We define a bit-stuffing procedure of two communicating finite state machines M and N; the procedure is similar to that used in the standard HDLC protocol. Machine M continuously sends a stream of bits to N via an unbounded, FIFO channel. At some instants, M stuffs extra zero bits into the data bit stream to prevent the current bit pattern from looking like a special pattern that signals the end of the data stream. Machine N receives all the bits sent by M, detects the stuffed zero bits, and removes them from the data stream. We verify the safety and liveness properties of this network by constructing and examining a finite representation of its infinite reachability graph.

Table of Contents

1. Introduction

We consider a system of two processes called "sender" and "receiver:" Continuously, the sender sends a stream of binary bits to the receiver. When the sender has no useful data bits to send, it merely sends bits of ones. When the sender has some useful data bits, it sends them preceded by a special pattern "01110," and followed by the same pattern. This special pattern is called the *flag*. The function of these two flags is to inform the receiver that all the bits in between the two flags are useful data. To achieve this function, the sent data bits between the two flags should not contain the special pattern "01110." Therefore, as M sends the data bits, it keeps track of the number of ones which follow the last sent zero. Whenever this number becomes two, the sender inserts an additional zero into the data bit stream, forcing the sent pattern to become "0110." On the other side, whenever the receiver receives the pattern "0110" in the data bit stream, it recognizes that the last zero in this pattern was not in the original data stream and so discards it. In this note, we formally model this system as a network of two communicating finite state machines [1, 2, 6], and verify the correctness of our model.

The technique of stuffing extra zero bits into the bit stream at one end, and detecting and removing them from the data stream at the other end is called *bit-stuffing* [7]. It has been employed in the standard HDLC protocol, and its many versions such as SDLC, LAP, and ACCP [7].

Following the introduction, this note is organized as follows. First, networks of communicating finite state machines are briefly presented in Section 2. Then in Section 3, we discuss how to model the above system of sender and receiver using a network of two communicating finite state machines. Verification of this model is discussed in Section 4, and concluding remarks are in Section 5.

2. Network of Communicating Finite State Machines

A *communicating finite state machine* M is a labeled directed graph with two types of edges, namely *sending* and *receiving edges*. A sending (or receiving) edge is labeled -g (or +g, respectively) for some *message* g in a finite set G of messages. For convenience, we assume that each node in M has at least one outgoing edge. A node in M whose outgoing edges are all sending (or all receiving) edges is called a *sending* (or *receiving*) *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. [M,N] denotes the network consisting of machines M and N connected by two unbounded FIFO channels in opposite directions.

A *state* of a 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 a 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 a 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, labeled -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, labeled -g, from w to w' in N, and s'=[v,w',x.g,y].

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

iv. e is a receiving edge, labeled +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 a network [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 for i=1,...,r-1, s_{i+1} follows s_i over some edge e_i in M or N. A state of a network [M,N] is said to be *reachable* iff it is reachable from the initial state of [M,N].

A 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 state [v,w,x,y] of a network [M,N] is an *unspecified reception state for* M iff $x=g_1.g_2 \cdots .g_k$ ($k\geq 1$), and v is a receiving node and none of its outgoing edges is labeled $+g_1$. A state [v,w,x,y] is an *unspecified reception state for* N iff $y=g_1.g_2 \cdots .g_k$ ($k\geq 1$), and w is a receiving node and none of its outgoing edges is labeled $+g_1$. If no reachable state of [M,N] is an unspecified reception state for M or N, then the communication of [M,N] is said to be *free from unspecified receptions.*

3. Modeling the Bit-Stuffing Protocol

We model the system defined in Section 1 as a network [M,N], where M models the sender, N models the receiver, and the set G of messages sent from M to N is {0,1}. (0 denotes a zero bit, and 1 denotes a one bit.)

Figure 1 shows the sender M. Each edge in M is a sending edge where exactly one bit is sent, and so each edge is labeled with -0 or -1. For convenience, we add a one-letter "comment" beside the label of each edge to describe the sent bit at this edge:

- L indicates that the sent bit is a filling; i.e. it is sent only because the sender has no useful data bits to send.
- F indicates that the sent bit is part of a flag.
- D indicates that the sent bit is part of the useful data.
- S indicates that the sent bit is a "stuffed" extra zero bit.

Figure 2 shows the receiver N. Each edge in N is a receiving edge, where exactly one bit is received, and so each edge is labeled with +0 or +1. For convenience, we add a comment beside the label of each edge to describe how N recognizes the received bit at this edge: The labels L, F, D, and S indicate respectively that N recognizes the received bit as a filling, part of a flag, part of the useful data, or as a stuffed bit. The label X indicates that N cannot determine at this instant whether the received bit is part of a flag or part of the useful data; this determination will be made at some later instant as indicated by the added comments. For example, the edge from node 6 to node 7 is labeled with "+0" and with the comment "X", and the self-loop at node 7 is labeled with "+0" and with the comment "X and the last 1 bit is D." Thus, if N receives two successive zero bits starting from node 6, then it will not be able to determine before receiving the second zero, whether the first received zero is part of a flag or part of the useful data. But after receiving the second zero, N determines that the first zero is part of the useful data. (However, it will not yet be able to determine for the second zero bit.)

4. Verification of the Bit-Stuffing Protocol

The network [M,N], where M and N are defined in Figures 1 and 2 respectively, has an *infinite* number of reachable states. Hence, its communication cannot be verified by generating and examining all its reachable states. Instead, we generate and examine only those states that are reachable by forcing the two machines to progress in equal speeds. Since machine M only sends, and the other machine N only receives, then forcing M and N to progress in equal speeds will cause the network [M,N] to reach only states of the form [v,w,E,E], where E denotes the empty string. The set of all such reachable states is finite, and can be represented by the finite directed graph R in Figure 3. (R is called [3, 8] the *fair reachability graph* of network [M,N].)

In R, each vertex corresponds to a reachable state of the form [v,w,E,E], and each arc corresponds to a transition from such a state to the next. Thus, each arc in R corresponds to two directed edges one in machine M and one in machine N. For example, the arc from vertex [8,8,E,E] to vertex [10,7,E,E] corresponds to the two directed edges, one is from node 8 to node 10 in M, and the other is from node 8 to node 7 in N. For convenience, we label each arc in R with the concatenation of the labels of the two corresponding edges in M and N. Next we discuss how to use R to verify the safety and liveness properties of this network.

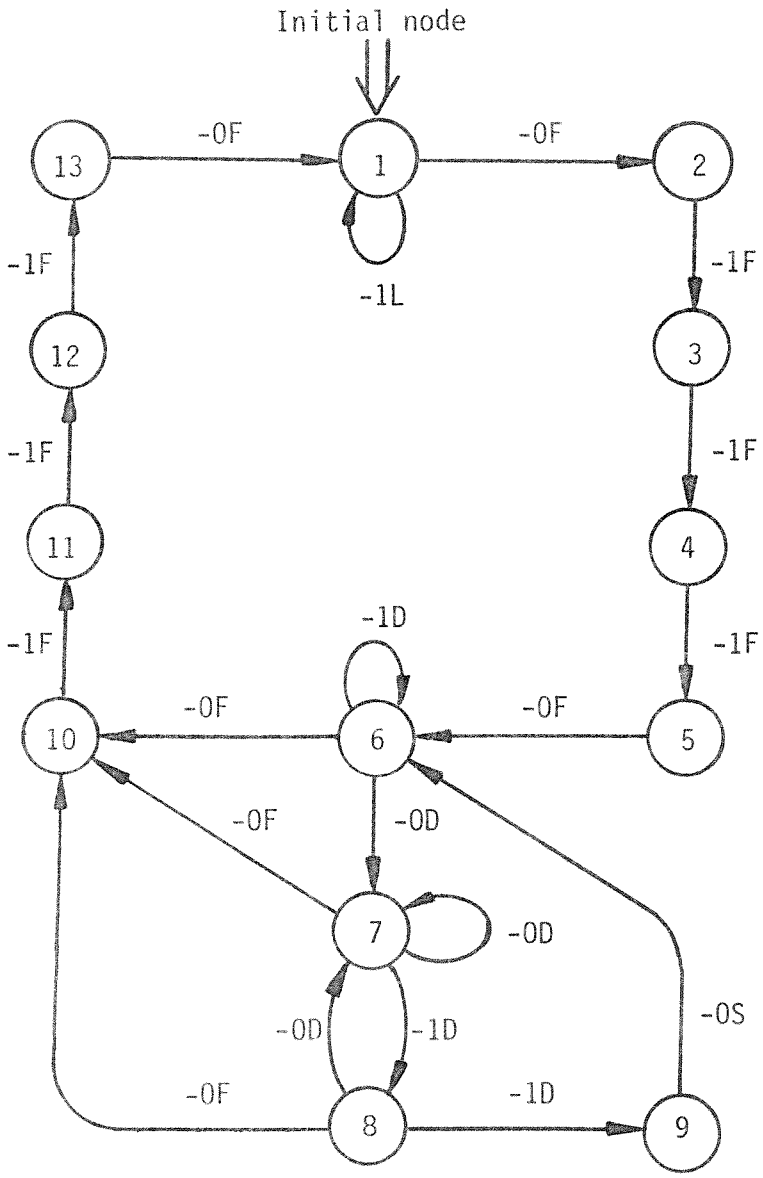

Figure 1. Sender M

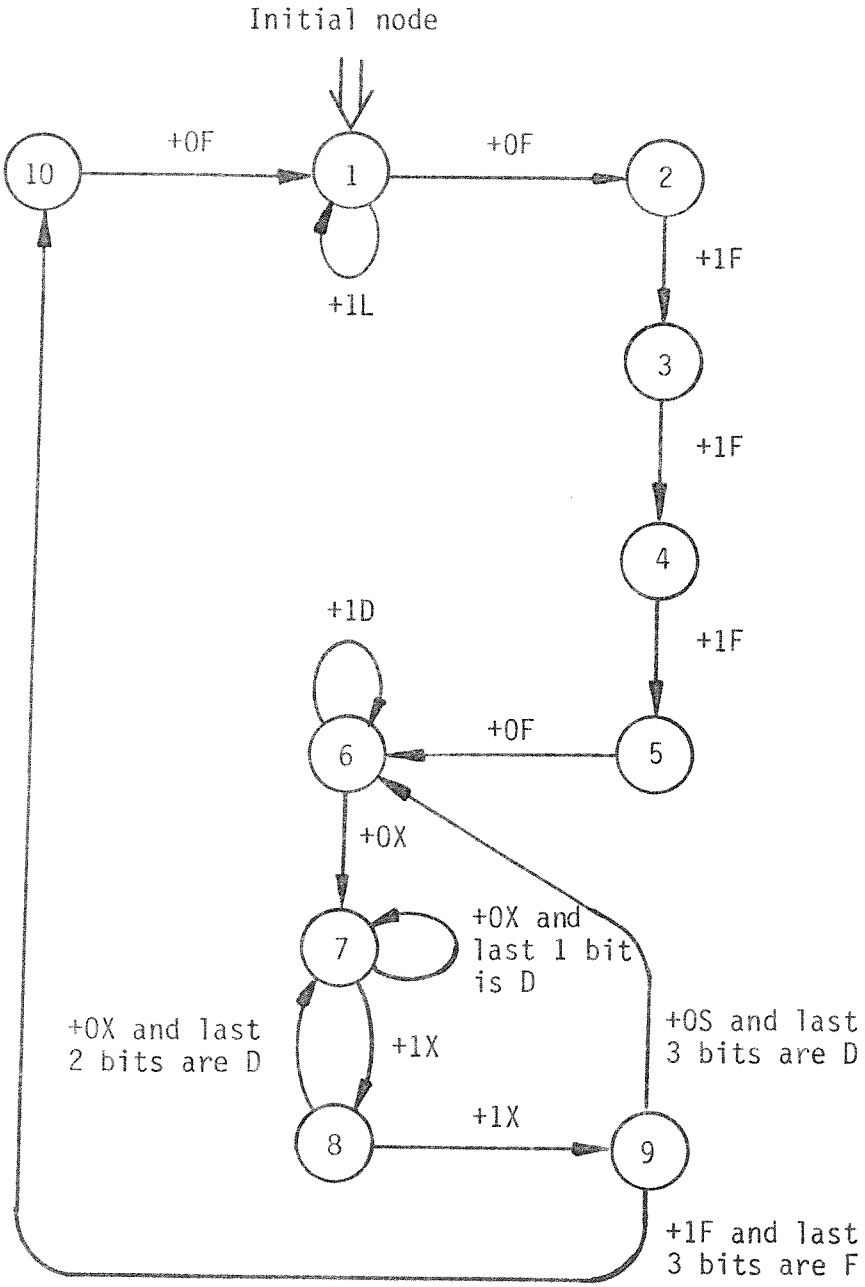

Figure 2. Receiver N

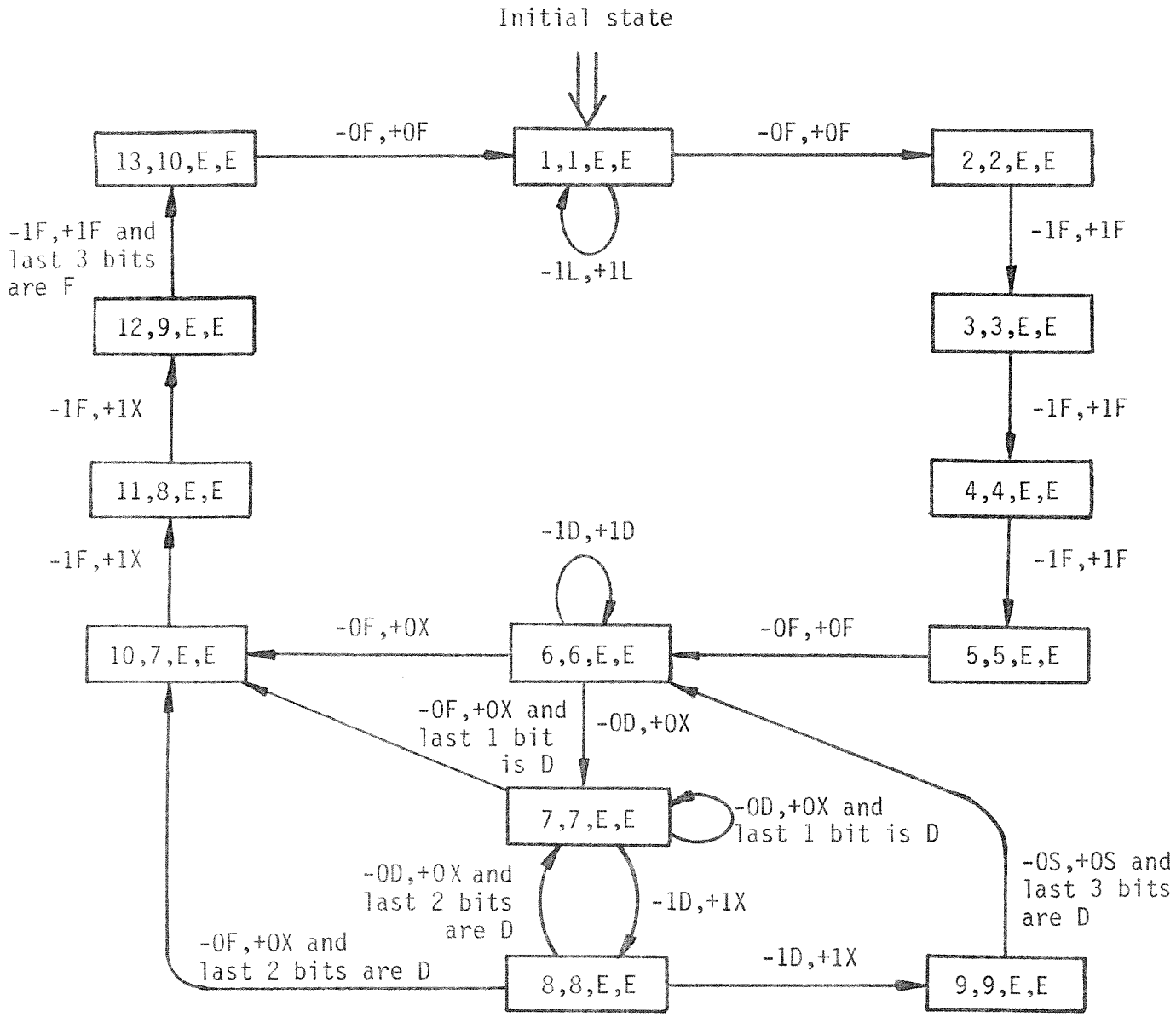

Figure 3. Fair reachability graph of network [M,N].

4.1. Proving Safety Properties

The safety properties of this network can be stated as follows:

A. No reachable state of [M,N] is a deadlock or an unspecified reception state. (This safety property is needed later to prove the liveness properties of the network.)

B. For i = 1, 2, . . . , if machine N recognizes that its i^{th} received bit is L, F, D, or S, then the i^{th} sent bit by machine M is L, F, D, or S respectively.

Proving A is straightforward: Since machine M has no receiving nodes, then no reachable state of [M,N] is a deadlock or an unspecified reception state for M. Also from [3, 5], since no state in R, and no state that follows a state in R is an unspecified reception state for N, then no reachable state of [M,N] is an unspecified reception state for N.

Assertion B can be proved by checking that the label *l* of each arc c in R satisfies the following two conditions:

i. If *l* contains the term "+i K," where i = 0 or 1, and K = L, F, D, or S, then *l* must also have the term "-i K" with the same i and K.

ii. If *l* contains the term "the last i bits are K," where i = 1, 2, or 3 and K = F or D, then for every arc c' in R from which c can be reached by a directed path of at most i arcs, the label of c' must contain the term "-j K," for the same K, and for some j = 0 or 1.

4.2. Proving Liveness Properties

The liveness properties of this network can be stated as follows:

C. For i = 1, 2, . . . , machine N will receive the i^{th} bit.

D. For i = 1, 2, . . . , machine N will recognize that the i^{th} received bit is L, F, D, or S.

Assertion C follows immediately from the safety property A and the *weak fairness assumption* [4] that "if a machine *can* progress infinitely often, then it *will* progress infinitely often," where progress in this context means receiving a message.

Assertion D can be proved by checking that the label *l* of each arc c in R satisfies the following condition: If *l* contains the term "+j X," where j = 0 or 1, then in each directed path that starts with c there exists an arc c' such that (a) c' is at most i arcs away from c, and (b) the label of c' contains the term "the last i bits are K," for some K = F or D.

4.3. Proving Total Correctness

Combining the safety and liveness properties B, C, and D, we have established the following: For i = 1, 2, . . . , machine N will receive the i^{th} bit, and will recognize whether this bit is L, F, D, or S; moreover this recognition is in agreement with machine M's intention when it has sent this i^{th} bit.

5. Concluding Remarks

The stuffing-bit protocol discussed in this note is different from, and in some sense *more efficient*, than the one discussed in [7]. The protocol in [7] requires that a zero bit be stuffed after every two successive ones in the data stream regardless of whether or not this pair is preceded by a zero. Therefore, if the data stream consists of n bits of all ones, then the protocol in [7] requires to stuff extra n/2 zero bits, while our protocol requires to stuff only one zero bit, irrespective of the value of n.

The flag of the HDLC protocol is 01111110; it is three bits longer than our flag. Nevertheless, it is straightforward to modify machines M and N according to this longer flag. (Hint: the modified machine M has 22 nodes.)

References

[1] G. V. Bochmann, "Finite State Description of Communication Protocols," *Computer Networks*, Vol. 2, pp. 361-371, 1978.

[2] D. Brand and P. Zafiropulo, "On Communicating Finite State Machines," *JACM*, Vol. 30, pp. 323-342, April 1983.

[3] M. G. Gouda and J. Y. Han, "Protocol Validation by Fair Progress State Exploration," Technical Report 85-31, Dept. of Computer Sciences, Univ. of Texas at Austin, October 1984. (Submitted for journal publication.)

[4] M. G. Gouda and C. K. Chang, "Proving Liveness for Networks of Communicating Finite State Machines," Technical Report 84-4, Dept. of Computer Sciences, Univ. of Texas at Austin, February 1984. (Submitted for journal publication.)

[5] J. Rubin and C. H. West, "An Improved Protocol Validation Technique," *Computer Networks*, Vol. 6, April 1982.

[6] C. A. Sunshine, "Formal Techniques for Protocol Specification and Verification," *Computer*, Vol. 12, No. 9, pp. 20-27, September 1979.

[7] A. S. Tanenbaum, *Computer Networks*, Prentice-Hall Inc., Englewood Cliffs, pp. 167-170, 1984.

[8] Y. T. Yu and M. G. Gouda, "Unboundedness Detection for a Class of Communicating Finite State Machines," *Information Processing Letters*, Vol. 17, pp. 235-240, December 1983.