# ALGORITHMIC VERIFICATION OF ISDN NETWORK LAYER PROTOCOL

**M. Ümit Uyar, Aleta Lapone, and Krishan K. Sabnani**

*The authors are members of technical staff at AT&T Bell Laboratories.* ***M. Ümit Uyar*** *is with the Network Interfaces and Standards Department at Holmdel, New Jersey. His interests include formal testing and verification of communication protocols, expert systems, and parallel processing. He received a B.S. from Istanbul Teknik Üniversitesi, Turkey, and an M.S. and Ph.D. from Cornell University, all in electrical engineering. He joined AT&T in 1986.* ***Aleta M. Lapone*** *is with the Distributed Systems Research Department at Murray Hill, New Jersey, where she develops techniques for protocol analysis and verification. She has a B.S. in computer science and mathematics from Montclair State College and an M.S. in computer science from Rutgers University. She joined AT&T in 1979.* ***Krishan Sabnani*** *is also with*
(continued on page 31)

This paper presents an algorithmic procedure for checking logical correctness of communication protocols and discusses its application to the Q.931 Integrated Services Digital Network (ISDN) network layer protocol. A protocol is specified as a collection of communicating finite-state machines (FSMs). The procedure described here consists of two steps. In the first step, the FSMs in the protocol are composed into a global FSM using the *incremental composition and reduction* (ICR) method. This method minimizes the state explosion problem by reducing the number of states in the global FSM by approximately one to two orders of magnitude while maintaining its observational equivalence. The second step checks whether the behavior of the service FSM, an FSM that models the services expected from the protocol, is a subset of the global FSM's behavior. A software tool, called APROVE (A Protocol Verifier), implements this procedure. We present the formal specification of Q.931 as a collection of 14 communicating FSMs. The results of its verification using APROVE are also reported. Several cases of incompleteness in the English language specification of Q.931 were detected in this exercise.



## Introduction

Communication protocols are becoming increasingly important with the proliferation of computer networks and the services provided by them. For the proper operation of such networks, the communication protocols must be free of logical errors. English language descriptions of protocols, however, are usually ambiguous and often incomplete. One way of eliminating these problems is to *formally specify* and *verify* communication protocols.[1,2]

**Panel 1. Acronyms Used in This Paper**

| | |
|---|---|
| APROVE | A Protocol Verifier |
| BRI | basic rate interface |
| CPE | customer premises equipment |
| CCITT | International Telephone and Telegraph Consultative Committee |
| CSP | Communicating Sequential Processes |
| FSM | finite-state machine |
| ICR | incremental composition and reduction |
| I/O | input/output |
| LAPD | link access procedures for D channel |
| PSL | Protocol Specification Language |
| SDL | Specification Description Language |

Verifying a protocol means checking whether it has the desired *safety* and *liveness* properties. The safety properties of a protocol typically include detection of deadlocks and functional errors in its design. The liveness properties address the detection of livelocks, in which no progress toward providing the desired services takes place. In other words, the safety properties prove that "bad things will not happen" (for example, deadlocks will be absent) and the liveness properties ensure that "good things will happen" (for example, a message will eventually be sent).

In this paper, an algorithmic procedure is presented for checking the safety properties of a protocol, which is specified as a collection of finite-state machines (FSMs) interacting with each other using interprocess input/output operations proposed by Hoare in the language Communicating Sequential Processes (CSP).[3] (Panel 1 lists acronyms used in the paper.) This procedure has been implemented in the software tool called APROVE (A Protocol Verifier), which includes the previously developed software package called PAV.[4] As a case study, the formal specification of the ISDN network layer protocol is presented. This protocol is based on CCITT Recommendation Q.931[5] and will be referred to as Q.931 throughout the paper. The results of the verification of Q.931 using APROVE are also presented. Here, we will not focus on checking of liveness properties, which is dealt with elsewhere.[6,7]

The authors have done a complete demonstration of the verification procedure. Only a portion of their work is described here.

In the past, safety properties have typically been checked by using program verification techniques.[1,3] These techniques usually require some rather insightful proofs that are hard to automate. In contrast, our approach is based on an algorithmic technique and, therefore, can be easily automated as described in the following steps:

- *Incremental composition and reduction of FSMs.* Two FSMs are composed into one machine. This composed machine is then reduced while its observational equivalence is maintained with the original machine.[8] *Observational equivalence* of two machines means that an external observer cannot notice any difference in their external behavior. This term is formally defined in Reference 9. The resulting reduced machine is then composed with a third FSM. This procedure is continued until all the FSMs of a protocol are composed and reduced into one global FSM. A potential problem in computing global FSM using traditional reachability computation procedures is that it can become too large to be computed, commonly known as the *state explosion problem.* The incremental composition and reduction (ICR) method given here keeps the state explosion problem under control and allows verification of complex protocols such as Q.931. By using the ICR method, the number of states in the composed machine for Q.931 is reduced from about $10^5$ to about $10^3$. At the end of this step, the specification is verified for the absence of deadlocks and functional errors.
- *Verifying the services expected from a protocol.* The services too are modeled as an FSM, called the *service*

*FSM*. Typically, every protocol has a service specification which can be used to construct its service FSM. If a service specification is not provided, the service FSM should be defined by the user on the basis of the English definition of the protocol services. Errors in the protocol behavior will lead the service FSM into its *error* state representing an undesirable behavior. The global FSM produced by the ICR method is composed with the service FSM. The protocol delivers the expected services if the resultant FSM does not enter into its *error* state.

The ISDN basic rate interface (BRI) network layer protocol covers the call control procedures for establishing, maintaining, and clearing circuit-switched voice connections. From the English description given in Reference 5, a model consisting of 14 communicating FSMs is developed. This model formally defines the interactions among the user-network interfaces at the originating and terminating ends, B channel, dial tone, announcements, and timers. Furthermore, two FSMs model the services expected from Q.931 by the calling and called users.

Verification of the formal specification of Q.931 by using APROVE shows that the specification does not have any major inconsistencies or logical errors and delivers the expected services to the users. However, during the analysis, it was observed that the English specification given in Reference 10 is very coarse. It does not provide enough details on how to handle race conditions, such as network and user interfaces trying to clear a call simultaneously. We present a formal description of Q.931 that is free of such errors. Although the shortcomings of an English specification can potentially lead to deadlocks in the implementations, they are usually overcome by the expertise of developers during the implementation phase. However, future protocols should be formally specified and verified to avoid these problems.

The remainder of this paper defines the terms used, describes the algorithmic procedure for checking the safety properties, introduces our software tool APROVE, gives a formal specification of Q.931 and the results of its formal verification, and presents concluding remarks.

#### Definitions

A protocol is modeled as a collection of communicating FSMs. The interaction between the FSMs is specified by input/output operations similar to those used in the programming language CSP.[11] In an FSM $F_i$, a message $m$ sent to another FSM $F_j$ is an *output operation* for $F_i$ and is denoted as $F_j$ ! $m$ (send $m$ to $F_j$). In $F_j$, the same operation is defined as an *input operation* and is denoted as $F_i$ ? $m$ (receive $m$ from $F_i$). $F_j$ ? $m$ in $F_i$ and $F_i$ ! $m$ in $F_j$ are called *matching operations*. Matching operations are executed simultaneously within their corresponding FSMs and, therefore, they can synchronize FSMs. In other words, if an FSM wants to execute a matching output operation, it must wait until the corresponding FSM is ready to execute the matching input operation, and vice versa. For example, $F_i$ cannot execute $F_j$ ! $m$ unless $F_j$ executes $F_i$ ? $m$. An *internal operation* is an unobservable operation. While doing an internal operation, an FSM makes a state transition without interacting with any other FSM.

In this paper, an FSM is represented as a directed graph $G(V, E)$ where $V$ and $E$ are the set of vertices and edges, respectively. In $G$, a vertex corresponds to a state of the FSM and an edge between two vertices represents a state transition. Each edge is labeled by an input/output (I/O) operation or an internal operation that causes the state transition. For an edge directed from $v_i$ toward $v_j$, the vertices $v_i$ and $v_j$ are called the *tail* and *head* vertices (or states) of that edge, respectively. An edge labeled by $a * b$ denotes an I/O operation $a$ followed by an I/O operation $b$ required for that state transition. An edge labeled by $a + b$ is an abbreviation for two edges, labeled $a$ and $b$, connecting the same two

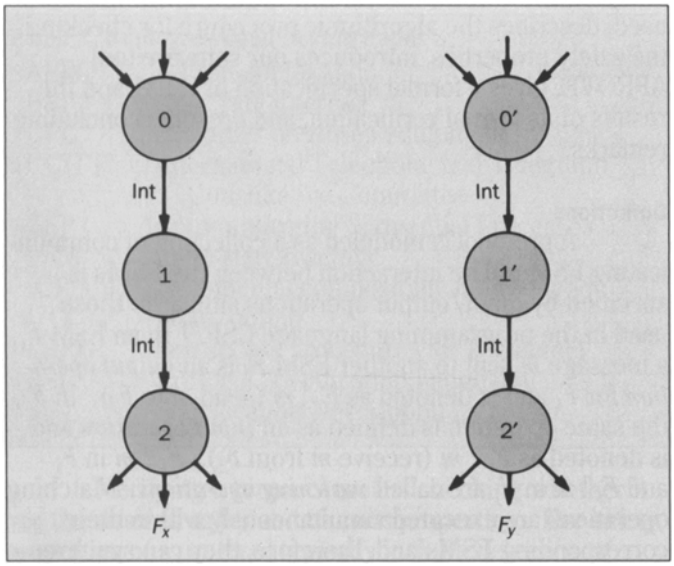



**Figure 1. Sections of two FSMs, $F_x$ and $F_y$.**

states. The services expected from the protocol are also specified as an FSM (the service FSM), which must have an *error* state, representing any undesirable behavior of the protocol.

We can build an FSM $F_1 \# F_2$ corresponding to the joint behavior of two FSMs $F_1$ and $F_2$. The machine $F_1 \# F_2$ is called the *reachable FSM* or the *composition* of $F_1$ and $F_2$. This operation of constructing $F_1 \# F_2$ is called the *reachability computation* or *composing*. This computation is done by computing *reachable global states*.[1,12,13] A *global state* for $F_1 \# F_2$ is defined as a two-tuple $(s_1, s_2)$, where $s_1$ is the current state of $F_1$ and $s_2$ is the current state of $F_2$.

For modeling a protocol as a collection of FSMs, we use a divide-and-conquer approach. The protocol being modeled is broken into components such that each can be easily modeled as an FSM. For example, Q.931 is modeled as a collection of 14 FSMs as explained later in this paper. Once a protocol is formally specified as a collection of communicating FSMs, its safety properties can be checked by an algorithmic procedure given in the next section.

#### Our Algorithmic Procedure for Protocol Verification

We first build a reachable global FSM using the ICR method described below. Reduction of states is done incrementally while the reduced machine's observational equivalence with the original machine is maintained. The final reduced machine is called the *reduced global machine*. Each new pair of machines chosen for composition should have considerable interaction with each other. Suppose a protocol consists of four FSMs, called $A$, $B$, $C$, and $D$. Some possible orders of composition are $(((A \# B) \# C) \# D)$, $(A \# B) \# (C \# D)$, and $((A \# B) \# D) \# C$. There is no quantitative way of ranking these orderings. But a good ordering can be chosen by inspecting the list of I/O operations. Any two FSMs chosen for composition should have a large number of matching I/O operations. For verifying safety properties of a protocol, we check whether its service FSM contains its reduced global FSM using an algorithm described in "Checking for Containment," below.

**Incremental Composition and Reduction Method.** The ICR method consists of two steps. During the first step, called *composition*, the reachable states and transitions of $F_1 \# F_2$ are computed by using an algorithm given in Reference 9. In the second step, called *reduction*, some states of $F_1 \# F_2$ are removed while the machine's observational equivalence is preserved.

During the reduction step, some states in the composed machine $F_1 \# F_2$ can be removed, provided that their incoming and outgoing edges satisfy certain conditions given in three rules described in this section. This step uses an approach similar to that developed by Milner and Hoare[8,14] and to the computation of projections.[15,16]

Composed global FSMs have typically several sections consisting of only internal transitions. If we compose two machines with internal edges, those

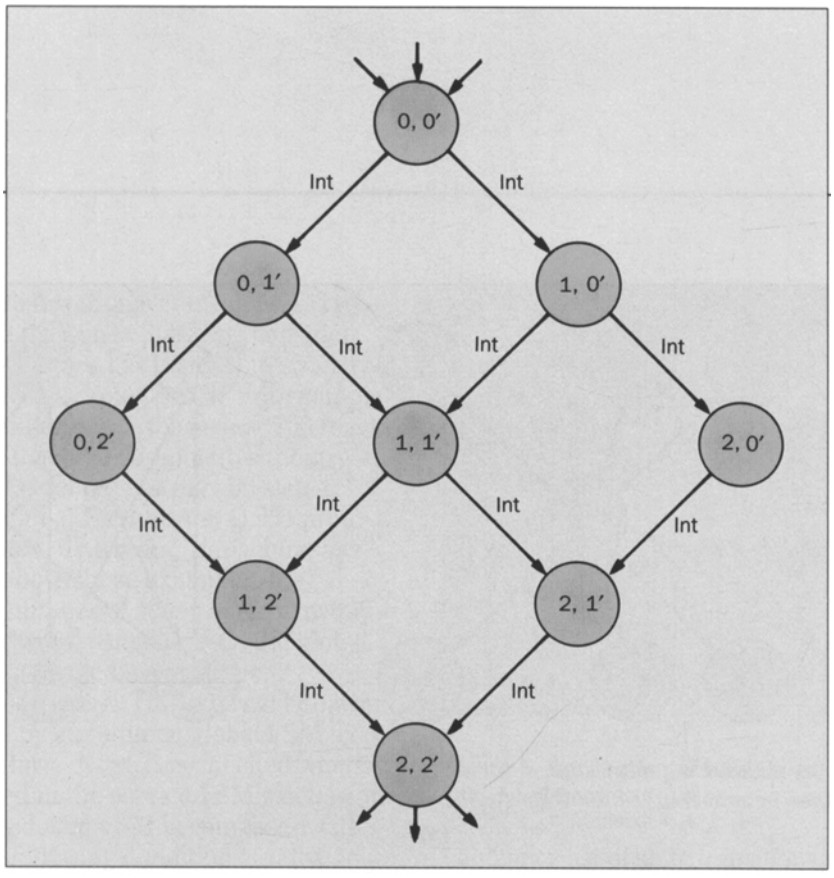



**Figure 2. Section of the FSM $F_x$ # $F_y$.**



sections that contain only internal edges may be very large. For example, consider such sections of two FSMs $F_x$ and $F_y$ as shown in Figure 1. Suppose the state (0, 0′) is reachable in the composition $F_x$ # $F_y$. Then, the section of $F_x$ # $F_y$ that covers the composition of the two sections of Figure 1 has 9 states, as shown in Figure 2. Since the sections consisting of only internal edges do not interact with one another, the number of states in the corresponding section of the composed machine is equal to the product of states in the sections being composed. If we were to reduce the machines $F_x$ and $F_y$ using the three rules given below, the corresponding sections can be merged into a single state:

Rule 1. A state $s_n$ for which all outgoing edges are internal can be removed while observational equivalence of the reduced machine with $F$ is maintained. Let the set of head states of the outgoing edges from $s_n$ be *SIO*, and the set of tail states of the incoming edges be *SI*. Here is the procedure for reduction:

- Remove $s_n$ and all its incoming and outgoing edges.
- For each $s_{io} \in SIO$ and each $s_i \in SI$, add an edge from $s_i$ to $s_{io}$ with the label of the incoming edge from $s_{in}$ to $s_n$ in the original FSM $F$ (Figure 3).

Rule 2. A state $s_m$ for which all incoming edges are internal can be removed while observational equivalence of the reduced machine with $F$ is maintained. Suppose the set of the tail states of incoming edges for $s_m$ is *SII*. Let the set of head states of the outgoing edges be *SO*. The procedure for reduction is as follows:

- Remove $s_m$ and all its incoming and outgoing edges.
- For each $s_{ii} \in SII$ and each $s_o \in SO$, add an edge from $s_{ii}$ to $s_o$ with the same label as that of the outgoing edge from $s_m$ to $s_o$ in $F$ (Figure 4).

Rule 3. Suppose two states $s_1$ and $s_2$ are connected by multiple internal edges. These multiple edges can be replaced by a single internal edge as shown in Figure 5.

It is shown in Reference 9 that the above rules

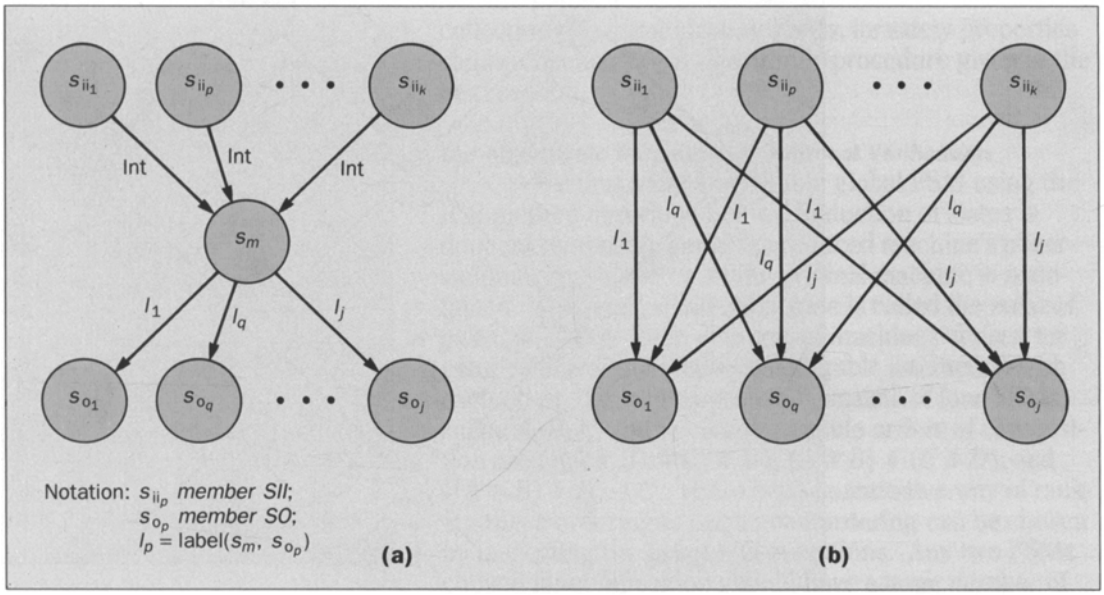



**Figure 3. Application of Rule 1. (a) State $s_n$ with the properties given in Rule 1. (b) Reduced section.**



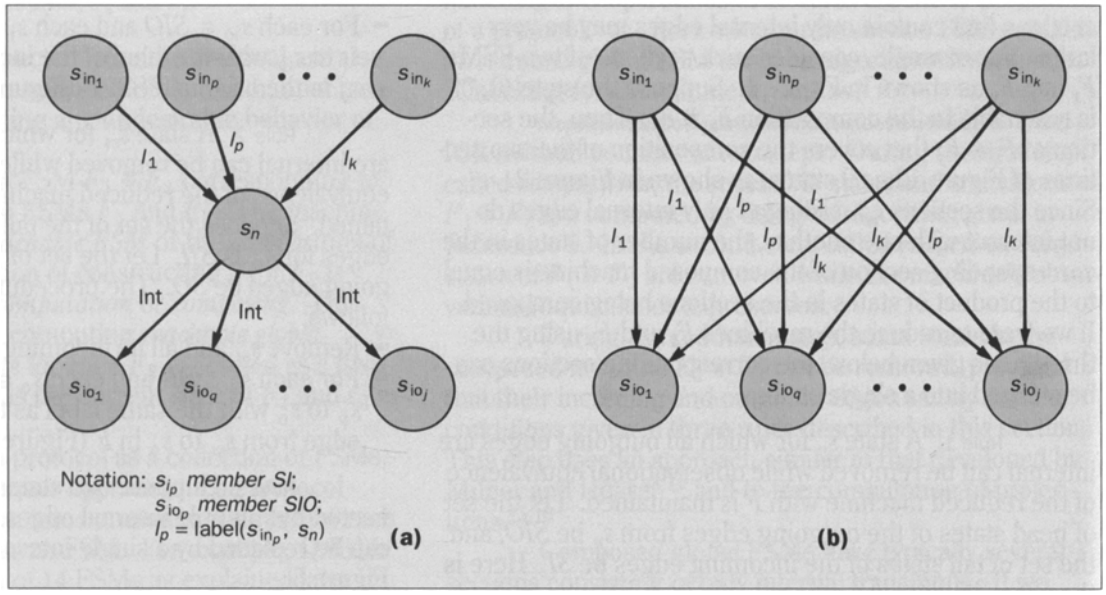



**Figure 4. Application of Rule 2. (a) State $s_m$ with the properties given in Rule 2. (b) Reduced section.**

for reduction maintain observational equivalence. Typically, the composition of machines that have considerable interaction with each other has many states satisfying the requirements given in Rules 1 to 3. For example, a version of ABP used in Reference 6 has over 700 reachable states in the global machine computed without reduction. The reduced machine has only 60 states. Without reduction, the global FSM for the Q.931 protocol has over 100,000 states. The reduced machine has less than 1000 states. In both these examples, the reduced machine has a number of states approximately one to two orders of magnitude smaller than the global FSM computed without intermediate reduction.

**Verifying the Service Delivery.** The external behavior of a protocol is captured by its reduced global FSM, computed by the ICR procedure. Next, it is checked whether this behavior is contained in the service FSM's behavior. In this step, the global reduced FSM is composed with the service FSM. If the following condition is satisfied, the protocol provides the expected services to its users: the resultant FSM obtained by composing the reduced global FSM and the service FSM does not have a state in which the service FSM enters into its *error* state.

The authors have developed a formal definition and proof of correctness for the above procedure. Another equivalent procedure is given in Reference 9, which checks whether the behavior of an FSM is completely contained in another FSM's behavior.

### A Protocol Verifier

The software package APROVE implements the procedures described above. It requires a machine-readable description of the component FSMs of the protocol. The input language is Protocol Specification Language (PSL).[4] In PSL, each FSM of a protocol is described as a process. Each process description has two parts: *declaration* and *body*. Process declarations contain a process name, a list of states, a list of I/O operations, and the initial state. The process body

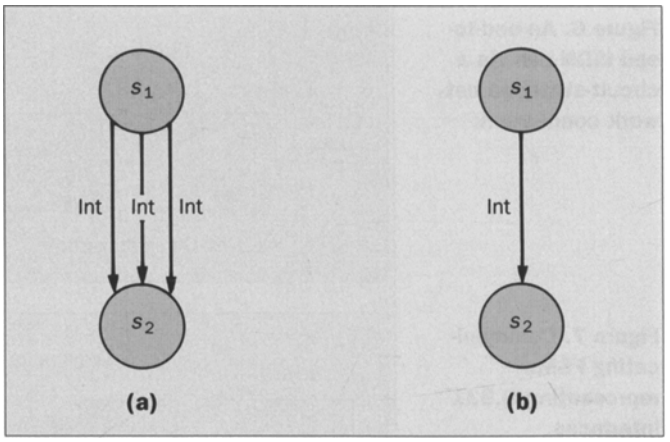



**Figure 5. Application of Rule 3. (a) A section of an FSM satisfying Rule 3. (b) Reduced section.**

contains a list of state transitions with the conditions that cause or are caused by the state transition (i.e., the I/O operations associated with the state transition). Each state transition has the following form:

current state : next state WHEN condition

APROVE first checks the syntactic correctness of the protocol specification: the state names and the I/O operations in each FSM must be consistent, and for each I/O operation there must be a matching pair in the corresponding FSM. Then, the FSMs representing the protocol are composed into a single FSM by the ICR method. Finally, this reduced global FSM is composed with the service FSM to check whether the protocol delivers the expected services to its users. As noted earlier, a service FSM models the services that the protocol is expected to provide to its users. Errors in the protocol behavior will lead this service FSM into its error state. APROVE indicates whether every transition of the service FSM is exercised by the protocol.

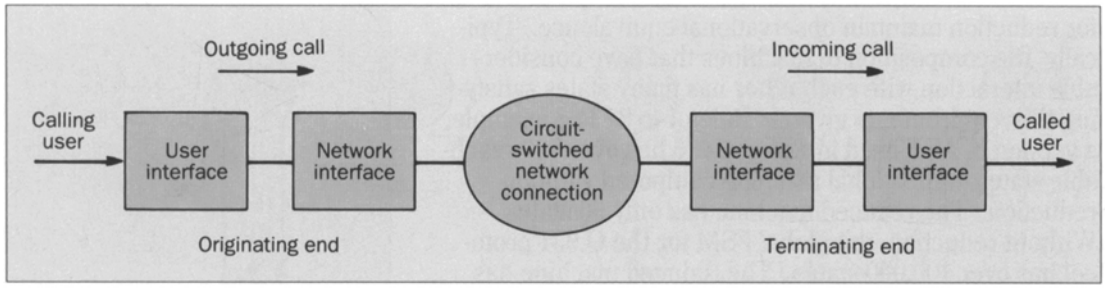



**Figure 6. An end-to-end ISDN call via a circuit-switched network connection.**

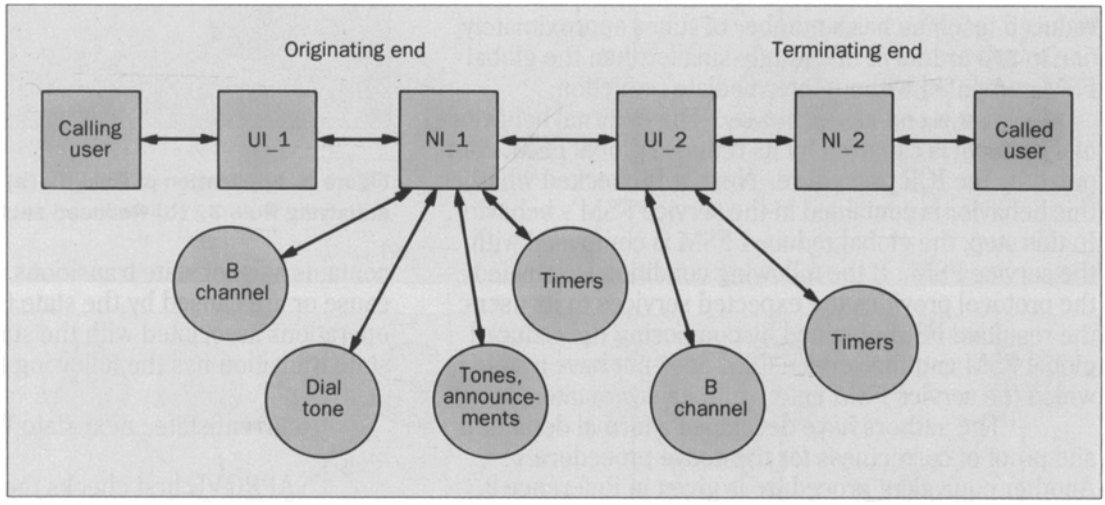



**Figure 7. Communicating FSMs representing Q.931 interfaces.**



### Formal Specification of Q.931

The ISDN BRI network layer protocol Q.931 establishes, maintains, and clears circuit-switched voice connections. It uses the services provided by LAPD.[5] The user originating a call will be referred to as the *calling user* and the one who is receiving the call as the *called user*. A basic circuit-switched voice connection is called an *outgoing call* with respect to the calling user. It is called an *incoming call* with respect to the called user. When a calling user originates a call, an outgoing call is initiated by the user interface at the calling user end (Figure 6). In order to complete the call, the network interface at the originating end informs the network interface at the terminating end, which in turn initiates the incoming call to the user interface of the called user.

Q.931 defines the user and network interfaces for outgoing and incoming calls. It defines the messages that are exchanged between the network and user interfaces at the originating and the terminating ends. The two network interfaces interact with each other using another protocol such as the Common Channel Signaling Protocol System 7.[17]

Q.931 is modeled as a set of communicating FSMs based on the English description and the

**Panel 2. Transition Table and Specification for UI_1**

The state transition table for the user interface at the originating end appears on pp. 26 and 27. The formal specification of UI_1 is as follows.

```
PROCESS UI_1;
STATES 0, 00, 0a, 0aa, 0d, 0dd, 0e, 0ee, 0o, 0oo, 0p, 0r
  1, 2, 2a, 2b, 2c, 2d, 3, 3a, 3b, 4, 4a, 10, 10c,
  11, 11a, 11c, 11cc, 11d, 11e, 11f, 12, 12a;

IO
  Calling_user?conreq_1,
  Calling_user?addr_1,
  Calling_user?cleareq_1,

  Calling_user!starthearing_1,
  Calling_user!toneoff_1,
  Calling_user!stophearing_1,
  Calling_user!dialtone_1,
  Calling_user!reorder_1,
  Calling user!ringback_1,
  Calling_user!no_dial_tone1,

  NI_1?setupack, NI_1?callproc, NI_1?alert,
  NI_1?conn, NI_1?disc, NI_1?rel,
  NI_1?relcom, NI_1?prog, NI_1?info,

  NI_1!setup, NI_1!disc, NI_1!rel,
  NI_1!relcom, NI_1!info, NI_1!connack;

INITIAL STATE 0;

TRANSITIONS

0: 1 WHEN Calling_user?conreq_1 * NI_1!setup,
0: 0a WHEN NI_1?setupack,
0: 0a WHEN NI_1?alert,
0: 0a WHEN NI_1?conn,
0: 0d WHEN NI_1?disc,
0: 0e WHEN NI_1?rel,
0: 0a WHEN NI_1?prog,
0: 0a WHEN NI_1?callproc,
0: 0 WHEN NI_1?info,
0: 0 WHEN NI_1?relcomm,
              . . .
              . . .
              . . .
```

Note: Only the first 10 transitions are given here.

**State Transition Table for User Interface at Outgoing End**

| Call State \ Message | SETUP ACK | CALL PROC | ALERT | CONN | DISC | REL |
|---|---|---|---|---|---|---|
| Null (U0) | Echo CRV; DISC → U0 | Echo CRV; DISC → U0 | Echo CRV; DISC → U0 | Echo CRV; DISC → U0 | Echo CRV; DISC → U0 | Echo CRV; REL COM → U0 |
| Call init (U1) | Condition 1: True — Cut B channel through to user; DIAL TONE → U2<br>Condition 1: False — REORDER, DISC; NO DIAL TONE → U1 | Condition 1: True — Cut B channel through to user → U3<br>Condition 1: False — REORDER, DISC → U0 | | | — → U12 | REL COM; Release CRV; NO DIAL TONE → U0 |
| Overlap sending (U2) | | — → U3 | RINGBACK → U4 | CONN ACK (optional); START HEARING → U10 | — → U12 | REL COM; Release B channel, CRV; REORDER → U0 |
| Outgoing call proc (U3) | | | RINGBACK → U4 | CONN ACK (optional); START HEARING → U10 | — → U12 | REL COM; Release B channel, CRV; REORDER → U0 |
| Call delivered (U4) | | | | CONN ACK (optional); START HEARING → U10 | — → U12 | REL COM; Release B channel, CRV; REORDER → U0 |
| Active (U10) | | | | | STOP HEARING → U12 | REL COM; Release B channel, CRV; STOP HEARING → U0 |
| Disconnect request (U11) | | | | | — → U11 | REL COM; Release B channel, CRV → U0 |
| Disconnect indication (U12) | | | | | | REL COM; Release B channel, CRV → U0 |

Message received from network interface at originating end

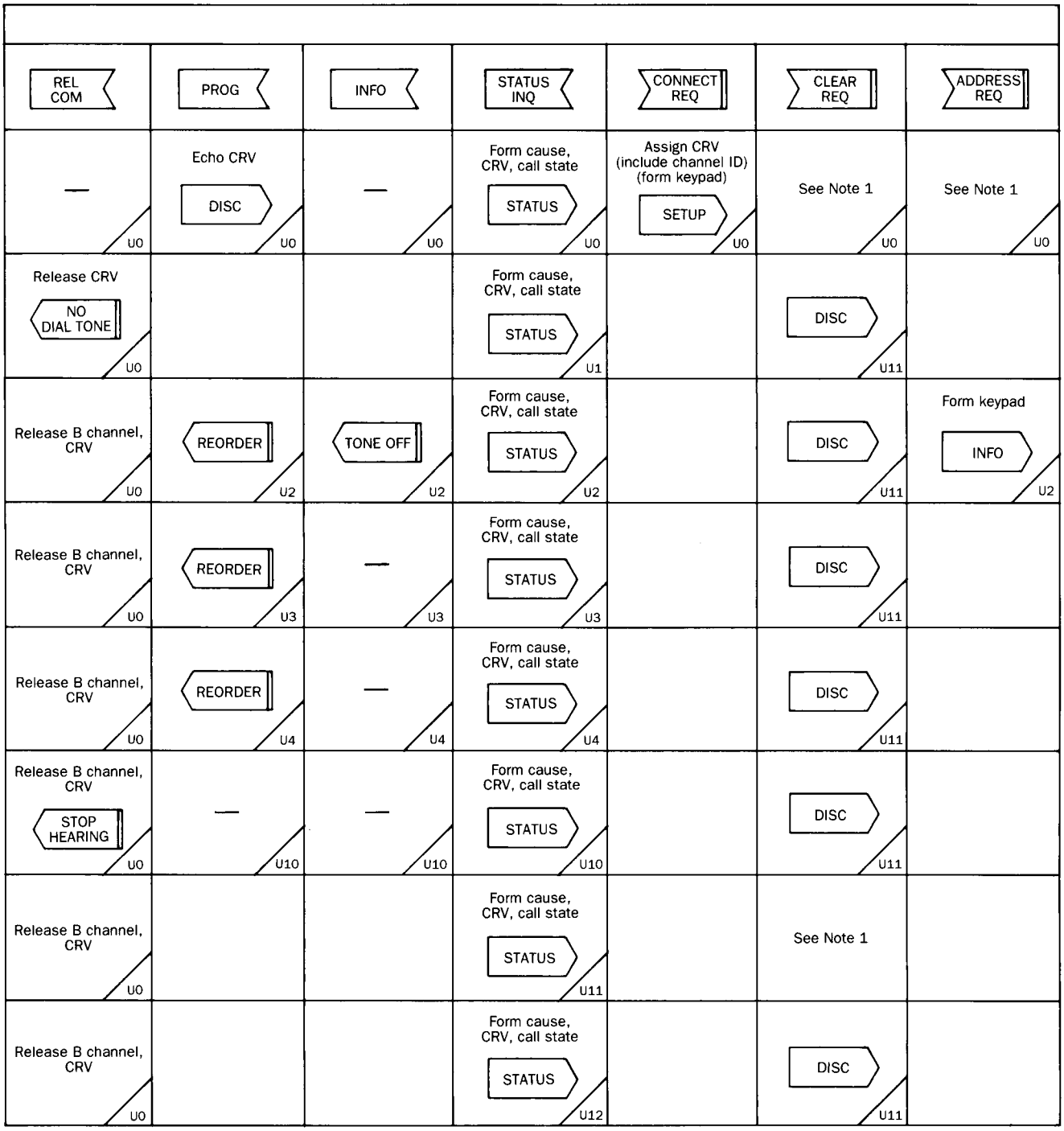
Message sent to network interface from originating end

Message received from calling user

Message sent to calling user

Condition 1: B channel is accepted

Note 1: implementation option


REL COM
PROG
INFO
STATUS INQ
CONNECT REQ
CLEAR REQ
ADDRESS REQ
—
U0
Echo CRV
DISC
U0
—
U0
Form cause, CRV, call state
STATUS
U0
Assign CRV (include channel ID) (form keypad)
SETUP
U0
See Note 1
U0
See Note 1
U0
Release CRV
NO DIAL TONE
U0
Form cause, CRV, call state
STATUS
U1
DISC
U11
Release B channel, CRV
U0
REORDER
U2
TONE OFF
U2
Form cause, CRV, call state
STATUS
U2
DISC
U11
Form keypad
INFO
U2
Release B channel, CRV
U0
REORDER
U3
—
U3
Form cause, CRV, call state
STATUS
U3
DISC
U11
Release B channel, CRV
U0
REORDER
U4
—
U4
Form cause, CRV, call state
STATUS
U4
DISC
U11
Release B channel, CRV
STOP HEARING
U0
—
U10
—
U10
Form cause, CRV, call state
STATUS
U10
DISC
U11
Release B channel, CRV
U0
Form cause, CRV, call state
STATUS
U11
See Note 1
Release B channel, CRV
U0
Form cause, CRV, call state
STATUS
U12
DISC
U11

Specification Description Language (SDL) diagrams in Reference 10. Fourteen communicating FSMs are defined, representing the user and network interfaces at the originating and terminating ends, B channel, dial tone, announcements, and timers (Figure 7). The PSL descriptions of these 14 FSMs formally define Q.931 and, therefore, constitute the formal specification of Q.931. The user interface at the originating end is modeled as one FSM, called UI_1. The network interface at the originating end is modeled as the FSM called NI_1, which has six satellite FSMs modeling the B channel, tones, announcements, and timers. The network and user interfaces at the terminating end are modeled by the FSMs called NI_2 and UI_2, respectively. NI_2 has four satellite FSMs corresponding to B channel and timers. Furthermore, two service FSMs (*calling_user* and *called_user*) model the services expected by users at the originating and terminating ends. Some of these FSMs are described briefly below.



**User Interface at the Originating End (UI_1).** The user interface at the originating end initiates, maintains, and clears the outgoing calls. This interface exchanges messages with the calling user and the network interface at the originating end.

The messages sent to the calling user by the user interface are not explicitly defined in Reference 10. However, the actions to be taken by the user interface with respect to the calling user are clearly stated. Explicitly defining communication with the user is deliberately avoided in the specification since it may be interpreted as an implementation requirement to the customer premises equipment (CPE) manufacturers. We have defined the following pseudo-messages between the user interface and the calling user based on the English description given in Reference 10: DIALTONE, NO_DIAL_TONE, TONEOFF, RINGBACK, START_HEARING, STOP_HEARING, and REORDER.

A state transition table of this user interface is given on pages 26 and 27. The left-most column of the state transition table represents the states of the user interface. The top row of the table lists the messages that are received from the network interface and the calling user. An entry in the table corresponds to the action taken by the user interface. The next state of the user interface is given at the lower right corner of each entry. For example, the table entry corresponding to state U1 (the second row of the table) and the message called RELease (the sixth column of the table) will be interpreted as follows: the user interface in state U1 will send a RELease COMplete message to the network interface if it receives a RELease message and moves into state U0.

If the action of the user interface depends on certain conditions, all possible actions are presented in the entry with a note explaining the condition. For example, the response of the user interface to a SETUP ACKnowledge message in state U0 depends on Condition 1. If Condition 1 is satisfied (various conditions are described in the table), the B channel will be cut through to the calling user, the dial tone will be provided, and the next state will be U2; otherwise, the user interface will send a DISConnect message to the network interface (dial tone is not provided to the calling user) and will move to state U11.

This interface is modeled as an FSM called UI_1 (the formal specification of Panel 2). Although the English specification given in Reference 10 defines nine states for this user interface (as in the panel), its formal model has 34 states. The reason for the additional states is that the English language specification is not precise enough, which causes some deadlocks in the protocol as explained below in "Results of Q.931 Analysis Using APROVE." The addition of new states to get an error-free specification is an iterative process as discussed below.

**Network Interface at the Originating End (NI_1).** The network interface at the originating end is responsible for continuing call establishment through the network toward the network interface at the terminating end. This interface also performs the call clearing procedures to release the B channel and other resources allocated to a

**Panel 3. Formal Specification of Service FSMs**

```
PROCESS Calling_user;
STATES 0,1,2,3,4,5,6,7,ERROR;

IO
  UI_1!conreq_1,UI_1!clearreq_1,addr_1,
  UI_1?starthearing_1, UI_1?toneoff_1,
  UI_1?stophearing_1, UI_1?dialtone_1,
  UI_1?no_dial-tone, UI_1?reorder_1,
  UI_1?ringback_1;

INITIAL STATE 0;

TRANSITIONS

0: 1 WHEN UI_1!conreq_1,
0: ERROR WHEN UI_1?starthearing_1,
0: ERROR WHEN UI_1?toneoff_1,
0: ERROR WHEN UI_1?stophearing_1,
0: ERROR WHEN UI_1?dialtone_1,
0: ERROR WHEN UI_1?reorder_1,
0: ERROR WHEN UI_1?ringback_1,
  . . .
  . . .
  . . .
```

Note: Only the first seven transitions are given here.

```
PROCESS Called_user;
STATES 0,1,2,3,4,ERROR;

IO
  UI_2!conreq_2,
  UI_2!clearreq_2, UI_2?ringing_2,
  UI_2?starthearing_2, UI_2?stophearing_2;

INITIAL STATE 0;

TRANSITIONS

0: 1 WHEN UI_2?ringing_2,
0: ERROR WHEN UI_2?starthearing_2,
0: ERROR WHEN UI_2?stophearing_2,

1: 2 WHEN UI_2!conreq_2,
1: 0 WHEN UI_2?stopringing_2,
1: ERROR WHEN UI_2?ringing_2,
1: ERROR WHEN UI_2?starthearing_2,
1: ERROR WHEN UI_2?stophearing_2,
  . . .
  . . .
  . . .
```

Note: Only the first eight transitions are given here.

call. There are three timers defined in Reference 10 for this interface, namely T302 for interdigit delays, and T305 and T308 for call clearing delays. This interface also provides dial tone, in-band or audible tones, and announcements to the user interface at the originating end.

The network interface exchanges messages with the user interface at the originating end, timers, B channel coordinating processors (both local to the network interface at the originating end), and the network interface at the terminating end.

This interface is modeled as the FSM NI_1. The number of states in this machine is 47. It is much larger than the number of states in the English specification (nine states) for the reasons given under "Results of Q.931 Analysis Using APROVE."

In order to model the B channel, tones, announcements, and timers in sufficient detail, six satellite FSMs are defined for NI_1. These FSMs do not communicate with any FSM except NI_1. These machines are called CONNECTOR_1, DIALTONE_1, ANNOUNCE_1, T302, T305, and T308, respectively.

**Calling and Called User Service FSMs.** The services offered to the calling and called users are described in Reference 10. Two service FSMs (*calling_user* and *called_user*) model these users (Panel 3). A state called *ERROR* is defined in each one of the service FSMs. Any undesirable behavior of the protocol drives at least one of these machines into its *ERROR* state.

### Results of Q.931 Analysis Using APROVE

Fourteen communicating FSMs given in the previous section were composed using the ICR method. The reduced global FSM has under 1000 states, while the global FSM without intermediate reduction has over 100,000 states. Several iterations of modification were required to identify and remove all possible deadlocks from the original specification. Initial versions of the formal specification had several deadlocks due to the lack of precision in the English language specification as discussed below.

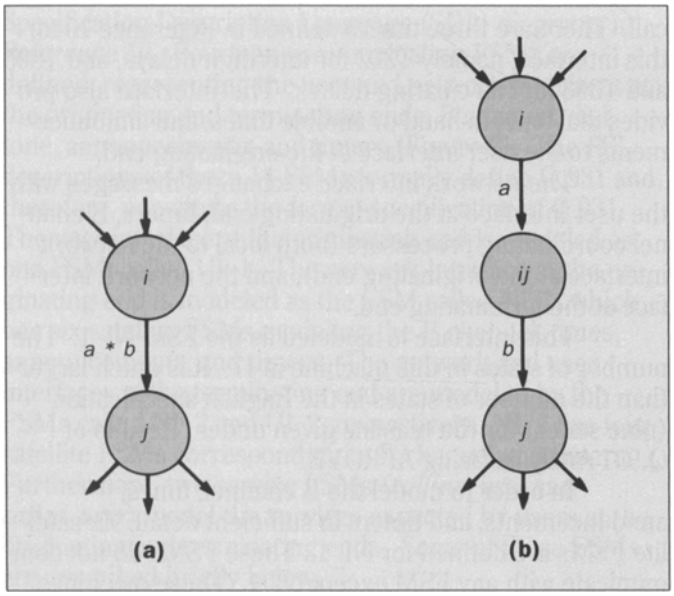



**Figure 8. (a) A transition with the label *a* * *b*. (b) Expanded transition.**

As noted, the English language description of Q.931 is very coarse. For example, several state entries in the table of Panel 2 consist of many actions. If some of these actions cannot be completed, the protocol can get into a deadlock. Consider a state transition for a process *P* from state *i* to state *j* which requires completion of two actions called *a* and *b*. This is modeled in the corresponding FSM as a state transition with the label *a* * *b* (Figure 8a). However, multiple actions imply that there is an intermediate state *ij* into which the FSM enters after action *a*. Suppose *b* requires an output operation which cannot be accepted by the intended receiver FSM. The machine will be locked in the intermediate state *ij*, resulting in a deadlock. This error can be removed by defining the FSM behavior in the intermediate state *ij*.

There were many race conditions detected by APROVE in the user interface during the user-initiated clearing procedures and in the network interface during the network-initiated clearing procedures. For example, suppose UI_1 is in the *call delivered* (U4) state, where the user is provided an audible ringback. If the user hangs up, UI_1 will send a *DISConnect* message to NI_1 and UI_1 will move into the *disconnect request* (U11) state. At this point, NI_1 may have already sent a *CONNect* message to UI_1 (i.e., the far end answered the call). UI_1 receives a *CONNect* message in state U11. The SDL diagrams given in Reference 10 cover this case by stating that the user response is an "option," which implies an inopportune (i.e., unexpected) message reception throughout the specification. The user response for such a message should be clearly stated in the specification and only the inopportune message receptions should be left unspecified as an "option" for the developer.

Another set of errors was caused by uncertainty of some of the state definitions. For example, if UI_1 receives a *PROGress* message at the *overlap sending* (U2) state, the only action expected from UI_1 is clearing the call. Therefore, UI_1 has to move to a new state where the only user action expected is clearing the call. However, the English specification indicates that UI_1 stays in state U2, which means that the user can continue dialing. Similar situations occur at states U3 and U4. The specification should be more explicit to handle such events which can lead to unnecessary message exchanges between the interfaces, mistakes in implementation and delays in clearing the call in the network (such as waiting for a timer to expire).

### Conclusions

Communication protocols are very complex, since they define rules of communication among multiple processes connected by unreliable channels. However, English language specifications of such protocols are typically incomplete, which may result in implementations with undesirable behavior. To eliminate

ambiguities and misinterpretations, communication protocols should be formally specified and verified. These problems are clearly illustrated in this paper by an example real-life protocol, Q.931. The software package APROVE can help protocol designers in correctly specifying and verifying communication protocols. Among other protocols that are verified by using APROVE are LAPD, ISO Transport Protocol (Class 4), and IEEE 802.3 protocol.

**References**


1. G. v. Bochmann and C. A. Sunshine, "A Survey of Formal Methods," *Computer Networks and Protocols*, P. E. Green (ed.), Plenum Press, New York, May 1983, pp. 561-578.
2. G. J. Holzmann, "On limits and possibilities of automated protocol analysis," *Protocol Specification, Testing, Verification, VIII*, Harry Rudin and Colin West (eds.), North-Holland, New York, 1987.
3. B. T. Hailpern and S. S. Owicki, "Modular Verification of Computer Communication Protocols," *IEEE Transactions on Communications*, Vol. COM-31, No. 1, January 1983, pp. 56-68.
4. K. K. Sabnani and A. Lapone, "PAV—Protocol Analyzer and Verifier," *Protocol Specification, Testing, and Verification, VI*, B. Sarikaya and G. v. Bochmann (eds.), North-Holland, New York, 1986.
5. *CCITT Red Book*, Vol. III, Fascicle III.5, Integrated Services Digital Network (ISDN) Recommendation Q.931 (Study Group XVIII), 1984.
6. K. K. Sabnani, "An Algorithmic Procedure for Protocol Verification," *IEEE Transactions on Communications*, Vol. 36, No. 8, August 1988, pp. 924-931.
7. K. K. Sabnani, P. Wolper, and A. Lapone, "An Algorithmic Procedure for Protocol Verification," *Proceedings of IEEE Global Communications Conference*, 1985, pp. 82-88.
8. R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, New York, 1980.
9. K. K. Sabnani, A. Lapone, and M. Ü. Uyar, "An Algorithmic Procedure for checking Safety Properties of Communication Protocols," *IEEE Transactions on Communications*, September 1989, pp. 940-948.
10. *AT&T 5ESS™ Switch Basic Rate Interface Specification*, 5E5 Generic Program, 5D5-900-311, AT&T, December 1987.
11. C. A. R. Hoare, "Communicating Sequential Processes," *Communications of the ACM*, Vol. 21, No. 8, August 1978, pp. 666-677.
12. P. Zafiropulo et al., "Protocol Analysis and Synthesis using a State Transition Model," *Computer Network Architectures and Protocols*, P. E. Green (ed.), Plenum Press, New York, May 1983, pp. 645-670.
13. C. H. West, "An Automated Technique of Communications Protocol Validation," *IEEE Transactions on Communications*, Vol. COM-26, No. 8, August 1978, pp. 1271-1275.
14. C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985, pp 111-117.
15. S. L. Lam and A. U. Shankar, "Protocol Verification via Projections," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4, July 1984, pp. 325-342.
16. B. Pehrson, "Abstraction by Structural Reduction," *Protocol Specification, Testing, Verification, III*, H. Rudin and C. West (eds.), North-Holland, New York, 1983.
17. F. Hlawa and A. Stoll, "Common Channel Signaling based on CCITT System 7," *Telephony*, February 9, 1981.




Biographies (continued)

*the Distributed Systems Research Department at Murray Hill. He works on communication protocols. He has a B.Tech. from the Indian Institute of Technology and a Ph.D. from Columbia University, both in electrical engineering. He joined AT&T in 1981.*

*(Manuscript received July 6, 1989)*