ALGORITHMIC GENERATION OF PROTOCOL CONFORMANCE TESTS

Anton T. Dahbura, Krishan K. Sabnani, and M. Ümit Uyar

With the recent expansion of data communications networks, computers and terminals from different manufacturers must be interconnected. Before being connected to the network, these interacting elements should be tested to ensure that they conform to the network's protocol specifications. We describe an algorithmic method (which is based on unique input/output sequences and Rural Postman tours) for generating conformance-test sequences that require minimum cost (i.e., run time) and completely cover the state transitions defined by the protocol specification. The technique, which has been implemented as the *POSTMAN* software package, has been widely used in AT&T Bell Laboratories to generate protocol-conformance tests. This approach can be used to generate tests for valid, inopportune, and illegal messages; for error recovery; and for window-flow-control procedures (see Sherif and Uyar[1]).

Anton T. Dahbura *and* ***Krishan K. Sabnani*** *are members of technical staff in the Distributed Systems Research Department at AT&T Bell Laboratories in Murray Hill, New Jersey.* ***M. Ümit Uyar*** *is a member of technical staff in the Network Interfaces and Standards Department at AT&T Bell Laboratories in Holmdel, New Jersey. Mr. Dahbura's interests are fault-tolerant computing, formal methods for protocol design and analysis, and algorithmic graph theory. He joined Bell Laboratories in 1983 and has a Ph.D. in electrical engineering from The Johns Hopkins University. Mr. Sabnani works on communications protocols. He joined Bell Laboratories in 1981 and has a B.Tech. in electrical engineering from the Indian Institute of Technology, in New Delhi, India, and a Ph.D. in electrical engineering from Columbia*
(continued on page 118)

Introduction

Technological advances in computer communications have resulted in the development in several countries of public data networks with access protocols, such as X.25 and ISDN. [The X.25 packet-switching protocol is a standard recommended by the International Telegraph and Telephone Consultative Committee (CCITT). ISDN is the Integrated Services Digital Network. Panel 1 defines terms and acronyms used in this paper.]

Communications among systems connected by such networks can be reliably ensured only if:

- Before any new equipment is connected to such a public data network, the protocols implemented in the equipment conform to the network-protocol specifications; and
- The new equipment will not interfere with the network's operation.

Such protocol-conformance testing has become integral to the design

Panel 1. Acronyms and Terms

BRI	basic-rate interface
CCITT	International Telegraph and Telephone Consultative Committee
DTE	data terminating equipment
E	set of edges
E_C	composition of $(v_i, v_j; L_l) \in E$ and UIO_j
FSM	finite-state machine
G	directed graph; graphical representation of an FSM and equivalent to a state-transition table
I	set of inputs
I/O	input/output
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
IUT	implementation under test
LAPD	CCITT recommended link-access protocol on the D channel
OSI	Open Systems Interconnection (model)
PRI	primary-rate interface
PSL	Protocol Specification Language
state	stable condition in which the protocol rests until a stimulus (an input) is applied
tour	a sequence of state transitions
TTCN	Tree Tabular Combined Notation; ISO and CCITT format to represent the tests for communications protocols
UIO	unique input/output (sequences)
V	set of vertices of a directed graph
X.25	CCITT recommended packet-switching protocol at link and packet layers

of a communications system.

Two key issues exist in conformance testing of complex protocols:

- The conformance tests must be complete in their coverage of all aspects of the protocol to be tested.
- The time required to run such tests in a testing laboratory must not be unacceptably long.

Satisfying these conflicting constraints is a difficult problem. But by using formal methods, one can automatically generate conformance tests that require minimum time to run, and cover all modeled features of the protocol under test.

In this paper, we describe such an algorithmic technique for generating conformance tests and the application of this technique to several standard protocols. This technique generates test sequences that are about one third the size of those produced by manual ad hoc methods. In addition, these test sequences have excellent fault coverage. We have implemented this technique in the POSTMAN package. Testing organizations at AT&T Bell Laboratories, a division of AT&T, have been using this software tool to generate conformance tests for several protocols, including the:

- ISDN network-layer protocols for the basic-rate interface (BRI) and primary-rate interface (PRI)
- X.25 packet-layer protocol
- CCITT link-access procedures for D channel (LAPD).

Testing Protocol Behavior. During conformance testing of a protocol implementation, an external tester applies a sequence of inputs (e.g., messages or packets) to the implementation and verifies that the implementation behaves as its specification describes.

What complicates this exercise are limitations on the controllability and observability of the protocol implementation. Because of limited controllability, the implementation usually cannot be put directly into a desired state but, instead, requires several additional state transitions. Unless efficient solutions are found, this limitation can force test sequences to have infeasibly large numbers of state transitions. Limited observability prevents the external tester from directly observing the state of the protocol implementation. However, such observations are critical for a test to detect errors, because even the simplest protocol may admit an astronomical number of different input sequences. Thus, these limitations

make the conformance-testing problem combinatorially challenging.

A protocol can be specified as a *deterministic finite-state machine* (FSM),[2] described as follows. The *state* of a protocol is defined as a stable condition in which the protocol rests until a stimulus—called an *input*—is applied. When an input is applied, an FSM generates a response—called an *output* (which may be null)—and moves into a new state (which may be the same as the previous state), where it remains until the next input. The purpose of conformance testing is to check whether the implementation of an FSM behaves as defined by the specification.

Test-Sequence Generation. Of the formal conformance-testing techniques in the literature,[2–14] those that use FSMs to model protocols typically generate a set of input sequences that will force the FSM implementation to undergo all specified transitions. These techniques can be classified as the:

- Transition-tour method[3–5]
- Distinguishing-sequence method[2,7–10]
- Characterizing-sequences method (also known as the W-method)[2,9–11]
- Unique input/output (UIO) sequences method[12–14] (see Panel 2).

These techniques all assume the so-called *black-box* approach, where the external tester can observe only the outputs generated by the implementation (on receipt of inputs).

The *transition-tour* method generates a sequence of state transitions (called a *tour*) that exercises every state transition of the implementation,[3–5] but does not address the observability problem described above. An optimal tour can be generated by the technique described by Uyar and Dahbura,[5] which is based on a graph-theoretic concept called the *Chinese Postman problem*;[15] see Panel 2. (An *optimal tour* is a tour that covers every state transition of an FSM specification, and that requires a minimum number of state transitions.)

Certain protocols have a special message to determine the state of the protocol (e.g., the *status message* defined in the ISDN network-layer protocol).[16] For these protocols, the length of the tour can also be minimized by using the technique described by Uyar and Dahbura.[5] In this technique, the observability problem is solved by the protocol specification.

Panel 2. Transition Tours and UIO Sequences

Chinese Postman problem: to find a tour of a graph that starts and ends at the same vertex, is of minimum cost, and traverses each edge of the graph *at least* once.

Rural Postman problem: to find a tour of a graph that starts and ends at the same vertex, is of minimum cost, and traverses each of a specified *subset* of edges of the graph at least once.

UIO sequence (for state s_i): a sequence of inputs such that, if the input sequence is applied to the FSM when the FSM is in state s_i, the resulting output sequence could not have been produced by the FSM when the FSM is in any other state.

The remaining three methods emphasize the observability problem. In the *distinguishing-sequence* method, an input sequence is found for a protocol such that the outputs that the implementation generates will identify the protocol's state. The requirement for this method is a fully specified protocol, which may be too strong for most protocols. The *characterizing-sequences* method defines a *set* of input sequences for a subset of states in which the resulting set of output sequences distinguishes each state from the other states.

In both of these methods, the current state of the implementation is assumed to be unknown, but the methods generate sequences that can determine the current state. In other words, both the distinguishing and characterizing sequences answer the question: *What is the current state of the implementation?* In the *UIO-sequences* method, however, this question is relaxed and

Table I. State-Transition Table for ISDN BRI D-Channel Signaling Protocol (Network Side, Originating End)

State	UI_1?								NI_2?		
	rel	setup	info	connack	setup_bad	info_last	disc	info_bad	netalert	netclear	netconn
N0 null	UI_1! relcom N0	UI_1! setupack N1	—	—	UI_1! relcom N0	—	—	—	—	—	—
N1 dialtone	UI_1! relcom N0	—	UI_1! info N2	—	—	UI_1! info•callproc N3	UI_1! rel N19	UI_1! prog•disc N12	—	—	—
N2 overlap send	UI_1! relcom N0	—	null N2	—	—	UI_1! callproc N3	UI_1! rel N19	UI_1! prog•disc N12	—	—	—
N3 outcall proc	UI_1! relcom N0	—	—	—	—	—	UI_1! rel N19	—	UI_1! alert N4	UI_1! disc N12	—
N4 call deliv	UI_1! relcom N0	—	—	—	—	—	UI_1! rel N19	—	—	UI_1! disc N12	UI_1! conn N10
N10 active	UI_1! relcom N0	—	—	null N10	—	—	UI_1! rel N19	—	—	UI_1! disc N12	
N12 disc ind	UI_1! relcom N0	—	—	—	—	—	UI_1! rel N19	—	—	—	—
N19 rel req	—	—	—	—	—	—	null N19	—	—	—	—

becomes: *Is the implementation currently in state x?* The answer to this question generally yields much shorter sequences than the other two methods.

Most FSMs do not have a distinguishing sequence. Aho et al. describe [14] a UIO method that reduces the length of the sequences generated. This method is based on a more general form of the Chinese Postman problem, called the *Rural Postman problem*; see Panel 2. This paper describes in detail the approach based on UIO sequences and the Rural Postman problem. We also briefly discuss features of a software package, called POSTMAN, that implements this technique.

Standards organizations, such as the International Organization for Standardization (ISO) and CCITT, have defined basic concepts related to conformance testing.[17] However, the issues covered by the conformance-

Table I. continued

T302? timeout	T305? timeout	T308? timeout	State
—	—	—	N0 null
UI_1! prog•disc N12	—	—	N1 dialtone
UI_1! prog•disc N12	—	—	N2 overlap send
—	—	—	N3 outcall proc
—	—	—	N4 call deliv
—	—	—	N10 active
—	UI_1! rel N19	—	N12 disc ind
—	—	UI_1! rel N0	N19 rel req

testing standards do not include the generation of tests, because neither the manufacturers nor the major testing organizations have agreed on a formal test-generation method. As a case study, we apply the UIO sequence-Rural Postman method to conformance testing of X.25 destination terminal equipment (DTE) for the packet-layer protocol.[18] We show that the efficiency and effectiveness of tests defined in this draft international standard can be significantly improved by using the formal methodology we describe.

Approach to Test Generation

We now provide a detailed description of the UIO sequence-Rural Postman approach to test generation. State-transition tables and the graphical representation of an FSM are key components of this approach.

State-Transition Tables. A communications protocol can be modeled as a *finite-state machine* (FSM)[1] and is typically described using a *state-transition table.* The rows of the table correspond to the set $S = \{s_1, \cdots, s_n\}$ of stable *states* where the protocol FSM rests, until an external event or message (an *input* from the set I that corresponds to a column of the table) is applied to it. An input causes the protocol FSM to produce a predetermined *output* from the set O and to transition into a predetermined next state. This output and next state correspond to the appropriate entry in the table.

For example, Table I gives the state-transition table for a simplified version of the ISDN BRI D-channel signaling protocol (network-interface side, originating end).[16] The notation used to describe the input and output operations for each transition is based on the Protocol Specification Language (PSL)[13] and is interpreted as follows:

$$A?m_i / B!m_j$$

represents a state transition in FSM C caused by an input message m_i received from a different FSM A, which produces an output message m_j to a different FSM B.

For example, the entry in row N0, column `UI_1?setup_bad` of Table I corresponds to receiving a `setup_bad` message from FSM `UI_1` (the user-interface FSM) in state N0, sending a `relcom` message to `UI_1`, and remaining in state N0. The network interface at the originating end also communicates with `NI_2` (the network-interface FSM at the terminating end) and three

timer FSMs: T302, T305, and T308.

Null outputs are acceptable, and the protocol FSM is permitted to remain in the same state on receiving an input. However, it is assumed that, when the FSM is in a given state and a given input is applied, the subsequent output and next state are predictable; i.e., the FSM is said to be *deterministic.* Also assumed is that it is possible to reach any specified state from any other state via some sequence of valid inputs; i.e., the FSM is said to be *strongly connected.*

Finally, if each entry of the state-transition table contains a specified output and next state, then the entity is said to be *fully specified.* Otherwise, it is said to be *partially specified.* In the test-sequence-generation approach to be described here, the entity can be either fully or partially specified.

Graphical Representation of an FSM. A *directed graph* $G = (V, E)$ is a representation of an FSM and is equivalent to a state-transition table. The set $V = \{v_1, \cdots, v_n\}$ of vertices corresponds to the set of specified states of the FSM. A *directed edge* from vertex v_i to vertex v_j in E corresponds to a transition from state s_i to state s_j in the FSM.

Each edge in G is labeled by an input operation $a_k \in I$ and a corresponding output operation $o_l \in O$. Thus, an edge in E from v_i to v_j has the label, a_k/o_l, if and only if FSM M, in state s_i, on receiving input a_k, produces output o_l and moves into state s_j. Because there may be more than one transition from state v_i to v_j with different input and output operations, there are multiple edges in G. Therefore, an edge in G is specified by a triple $(v_i, v_j; L)$, where $L \equiv a_k/o_l$.

A cost can also be associated with an edge in G and corresponds to the time taken to realize the corresponding transition in M. Finally, the *initial state* s_0 of an FSM is the state that M is in immediately after power-up.

For example, Figure 1 shows a graphical representation of a simplified version of the ISDN BRI D-channel signaling protocol (network-interface side, originating end), whose state-transition table is given in Table I. The initial state is state N0. This FSM is partially specified, deterministic, and strongly connected. Furthermore, in this example, we assume all edges have unit cost.

Transition-Based Approach. The approach taken for algorithmically checking the conformance of a protocol implementation to its specification is to test the implementation for the correctness of every specified transition of M—that is, of every edge $(v_i, v_j; a_k/o_l)$ in E.

The procedure for testing a specified transition from state s_i to state s_j with input/output pair a_k/o_l takes place in three steps:

A1. The FSM implementation is put into state s_i.
A2. Input a_k is applied, and the output is checked to verify that it is o_l, as expected.
A3. The new state of the FSM implementation is checked to verify that it is s_j, as expected.

Henceforth, the subsequence of inputs for testing edge $(v_i, v_j; a_k/o_l)$ is denoted $TEST(v_i, v_j; a_k/o_l)$, and consists of input a_k followed by the sequence of inputs necessary to realize step A3.

In general, $TEST(v_i, v_j; a_k/o_l)$ is nontrivial to realize for two reasons:

- Because of the limited controllability of the FSM implementation, it is not possible to put the implementation of M into state s_i in step A1 of the above procedure without realizing several transitions (n transitions in the worst case).
- Because of the limited observability of the FSM implementation, it is not possible to verify directly that the implementation of M is in state s_j in step A3 of the above procedure.

In the past, several techniques have been designed for determining the state of an FSM when a sequence of inputs is first applied (henceforth called the *start* state of the FSM). These techniques involve applying a sequence of inputs and observing the outputs that are generated.[2,4,10,11,19–21] For example, a distinguishing sequence[10,19] is a sequence of inputs that produces a distinct sequence of outputs for each start state. Therefore, by applying a distinguishing sequence, it is possible to identify uniquely the start state of the FSM based on the

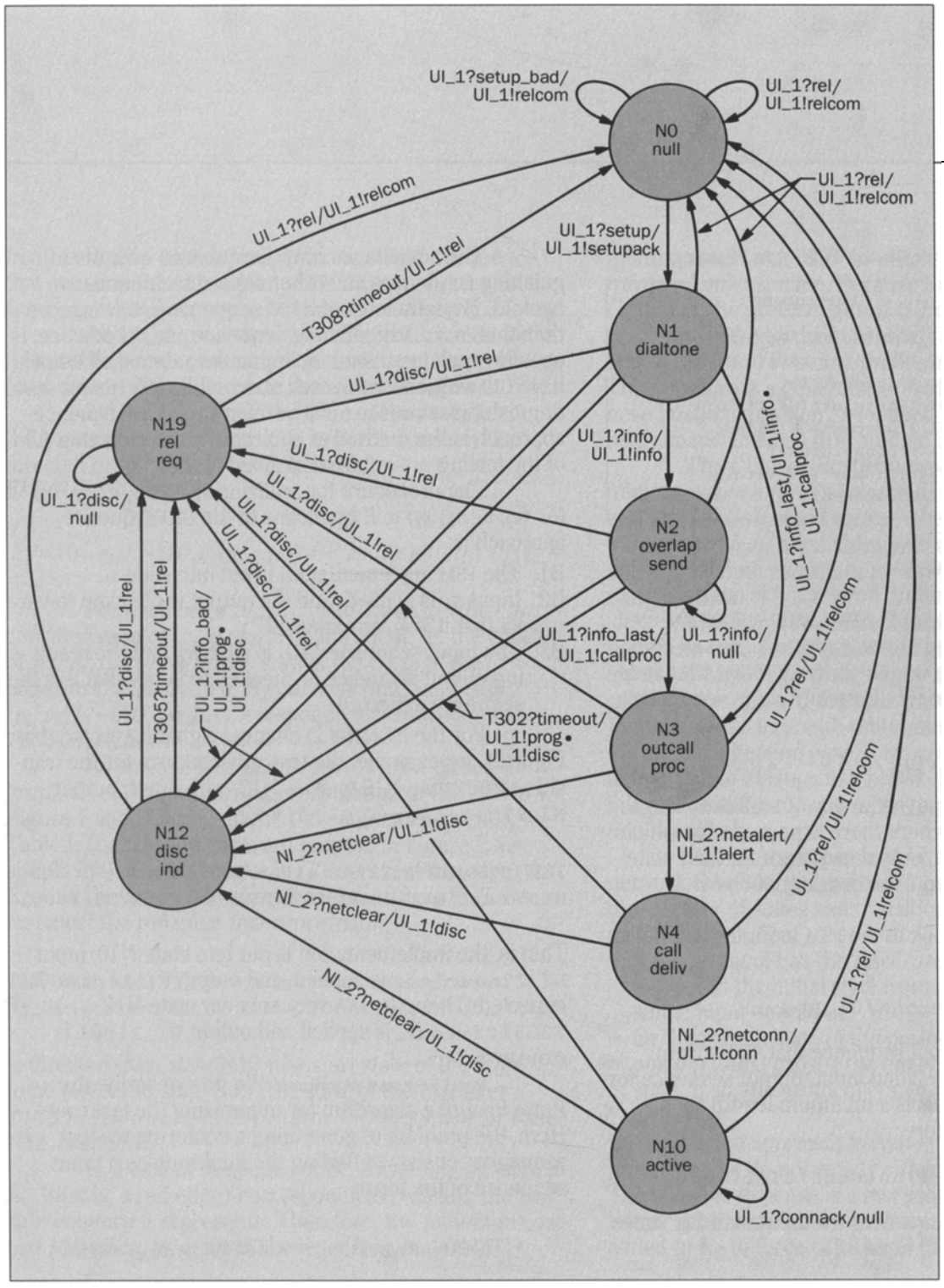

Figure 1. The directed graph *G*; FSM graphical representation of the D-channel signaling protocol for the ISDN basic-rate interface (network-interface side, originating end). Each arrowed line is a directed edge that represents the transition from one state to another. The I/O message sequence for the transition appears next to each edge.

Table II. UIO Sequences for States of FSM in Figure 1

State	UIO sequence
N0	`UI_1?setup_bad/UI_1!relcom`
N1	`UI_1?info/UI_1!info`
N2	`UI_1?info/null`
N3	`NI_2?netalert/UI_1!alert`
N4	`NI_2?netconn/UI_1!conn`
N10	`UI_1?connack/null`
N12	`T305?timeout/UI_1!rel`
N19	`UI_1?disc/null`

output sequence observed.

For protocol testing, however, it is enough to verify that the start state of the FSM is the state expected. If the start state is not what was expected, then an error has been detected. One way to verify an FSM's start state is to use a UIO sequence.[12,13] Formally, a UIO sequence for state s_i,

$$UIO_i = (a_{k_1}/o_{l_1}) \bullet (a_{k_2}/o_{l_2}) \bullet \cdots \bullet (a_{k_r}/o_{l_r}),$$

is a specified input/output sequence of minimum length and with start state s_i, where there is no $s_j \neq s_i$ for which UIO_i is a specified input/output sequence for start state s_j. (The bullet • represents the concatenation of input/output pairs.)

For example, look at the FSM in Figure 1. The input/output sequence,

`NI_2?netconn/UI_1!conn,`

is a specified input/output sequence that starts from state N4, and is not a specified input/output sequence for any other start state. This is a minimum-length such sequence for N4; therefore,

`UIO(N4) = NI_2?netconn/UI_1!conn.`

Table II gives the UIO sequences for the remaining states of the FSM represented in Figure 1.

The advantages of UIO sequences over distinguishing sequences and other related techniques are twofold. First, the cost of a UIO sequence is never more than that of a distinguishing sequence and, in practice, is usually much less. Second, in practice, almost all FSMs have UIO sequences for each state, while few have a distinguishing sequence.[2,10] Therefore, the UIO sequence approach is the method of choice for executing step A3 of the testing procedure described earlier.

The procedure for realizing $TEST(v_i, v_j; a_k/o_l)$ for $(v_i, v_j; a_k/o_l) \in E$ by means of the UIO-sequence approach is:

B1. The FSM implementation is put into state s_i.
B2. Input a_k is applied, and the output is checked to verify that it is o_l, as expected.
B3. The input sequence UIO_j is applied, and the resulting output sequence is checked to verify that it is the sequence expected.

For the ISDN BRI D-channel signaling protocol in Figure 1, for example, the test subsequence for the transition from state N10 to state N12, with input/output `NI_2?netclear/UI_1!disc` is:

$TEST$(`N10,N12;NI_2?netclear/UI_1!disc`) =
`NI_2?netclear/UI_1!disc•T305?timeout/UI_1!rel.`

That is, the implementation is put into state N10, input `NI_2?netclear` is applied, and output `UI_1!disc` is expected. Then, the UIO sequence for state N12, `T305?timeout`, is applied and output `UI_1!rel` is expected.

Rural Postman Algorithm. We now describe the Rural Postman algorithm for minimizing the test cost. Here, the problem of generating a conformance-test sequence consists of finding the minimum-cost input sequence of the form:

$$\cdots \bullet [TEST(v_{i_1}, v_{j_1}; L_1)] \bullet \cdots \bullet [TEST(v_{i_{|E|}}, v_{j_{|E|}}; L_{|E|})] \bullet \cdots$$

In other words, find the minimum-cost input sequence that contains the subsequence $TEST(v_i, v_j; L_k)$ for each $(v_i, v_j; L_k)$ that is a member of the set of edges E.

Aho et al. presented[14] an efficient solution for this problem under certain conditions that protocols often satisfy. This solution is described as follows.

Let $TAIL(UIO_j)$ be the state of the FSM after the UIO sequence for state s_j is applied. Consider the directed graph $G' = (V', E')$, for which $V' \equiv V$ and $E' \equiv E \cup E_C$, where:

$$E_C = \{(v_i, v_k; L_l \bullet UIO_j) : (v_i, v_j; L_l) \in E \text{ and } TAIL(UIO_j) = v_k\}.$$

For each edge $(v_i, v_k; L_l \bullet UIO_j) \in E_C$ [the concatenation of $(v_i, v_j; L_l) \in E$ and UIO_j], let the cost of $(v_i, v_k; L_l \bullet UIO_j)$ be the sum of the costs of the edges in G that are in it. By construction, each edge $(v_i, v_k; L_l \bullet UIO_j) \in E_C$ corresponds to a subsequence $TEST(v_i, v_j; L_k)$.

As an example, Figure 2 shows the directed graph G' of the graphical representation of the FSM in Figure 1 and is based on the UIO sequences given in Table I. (Labels and costs are omitted in Figure 2 for simplicity.) The red edges in G' represent the edges in E_C, and the green edges represent the edges in E. For instance, the red edge that represents:

TEST(`N10,N12;NI_2?netclear/UI_1!disc) =`
`NI_2?netclear/UI_1!disc•T305?timeout/UI_1!rel`

is directed from state N10 (the start state of the edge to be tested) to state N19 (the state of the FSM after `T305?timeout/UI_1!rel`, the UIO sequence for state N12, is applied).

In a tour of the edges of G', traversing an edge in E_C (that is, a red edge) corresponds to realizing the test subsequence it represents. Therefore, the minimum-cost test sequence, as defined above, corresponds to a minimum-cost tour of G' in which each edge in E_C is traversed at least once. Because the original edges of G (that is, the green edges of G') may be traversed to reach one red edge from another, the problem is equivalent to the Rural Postman problem of graph theory.[22] This problem is a generalization of the so-called Chinese Postman problem, so named because the Chinese mathematician Kuan first studied it.[15]

The Chinese Postman problem is to find a minimum-cost tour of a (directed) graph, starting and ending at a designated vertex, where each edge is traversed *at least* once. Edmonds and Johnson gave[23] a polynomial-time algorithm for solving the Chinese Postman problem.

The Rural Postman problem is to find a minimum-cost tour of a directed graph, starting and ending at a designated vertex, where each edge *in a given subset of the edges* (E_C here) is traversed at least once. Finding such a tour with minimum cost is *NP-complete* for the most general case,[22] which means the Rural Postman problem belongs to a broad class of problems in discrete mathematics for which a polynomial-time solution is unlikely.

But as Aho et al. showed,[14] we can use a network-flow-based algorithm[24] (which is polynomial in the number of states and transitions of the FSM) to find an optimal solution:

- If the protocol has the *reset capability*; that is, if it can move into the initial state from any other state with a single input operation.
- Or, if for each state of the protocol, there is at least one input that causes the entity to remain in the same state. If so, the protocol is said to have the *self-loop property*.

Of course, the approach in Aho et al.[14] is still valid for protocols that do not satisfy either of these conditions. The possible expense is a test sequence that is slightly longer than an optimal tour, because extra edges must be added to E_C to force G' to meet the requisite conditions.

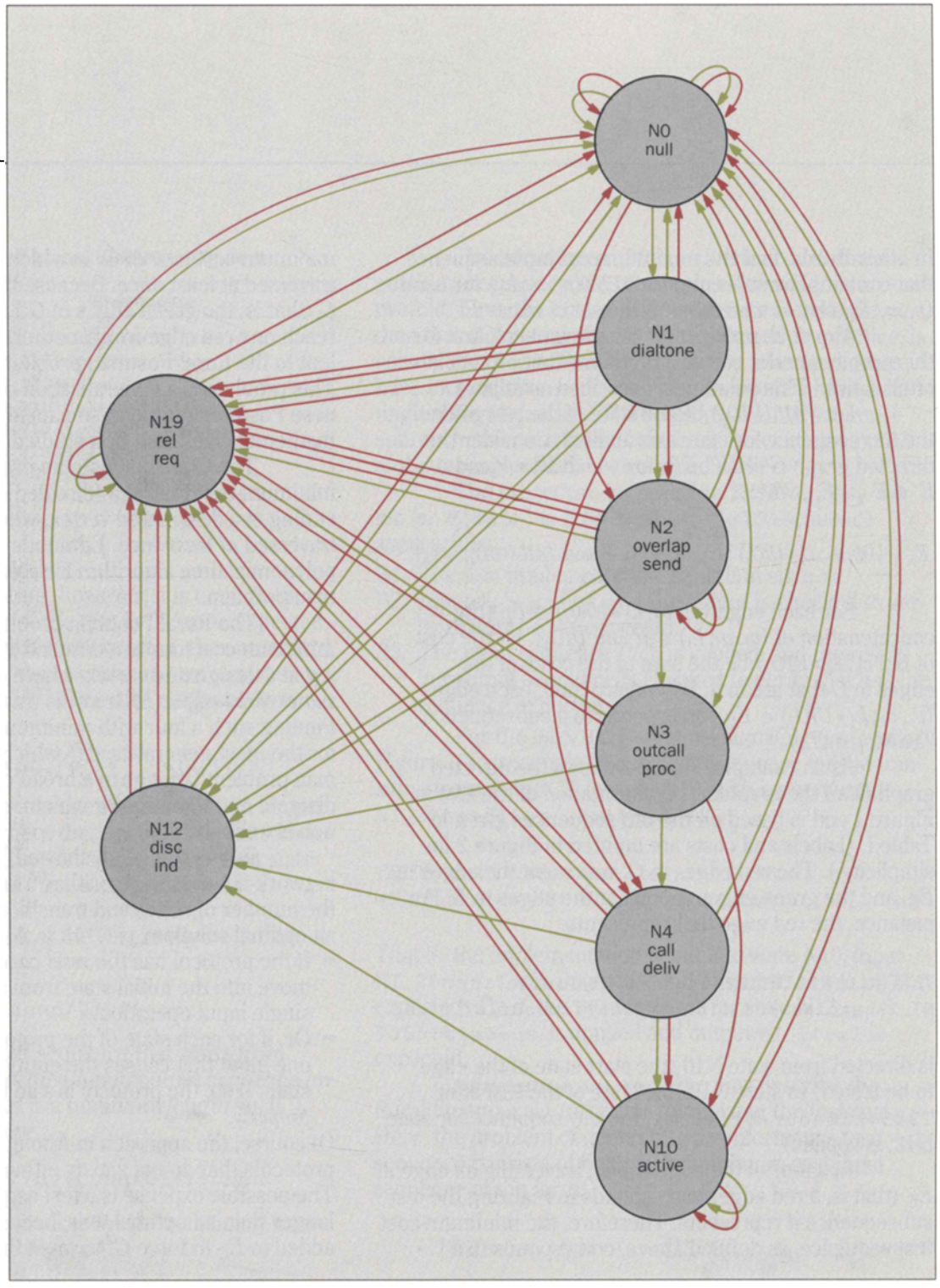

Figure 2. Directed graph G' for the graph G of Figure 1. The red edges are edges in the set E_C, while the green edges are edges in the set E.

In the ISDN BRI D-channel signaling protocol example, there is a transition from every state to state N0. Therefore, the protocol has the reset capability described above. Table III shows the resulting minimum-cost test sequence, which consists of 109 input/output pairs.

POSTMAN Test-Sequence-Generation Software System

The ideas described in the previous section are the basis of a software package, called POSTMAN, that was developed by the authors. POSTMAN can be used to generate test sequences for checking the conformance of a protocol implementation (or any other entity that can be modeled as a finite-state machine) to its specification.

Input-Specification Language. As input, POSTMAN accepts a description of the FSM. This input consists of a list of the FSM's transitions. For example, in the ISDN BRI D-channel signaling protocol in Figure 2, the transition from state N10 to state N12 is entered as:

```
N10 : N12 WHEN (NI_2?netclear,UI_1!disc) COST 1
```

If no cost is entered for a transition, then the transition is assumed to have unit cost. Each of the other transitions of the FSM is described in a similar fashion.

POSTMAN also has an optional feature that allows a user to input the FSM description interactively.

POSTMAN Features. We designed POSTMAN to be extremely flexible. For example, a user may specify any number of particular input sequences, called *scenarios*, that must appear in the test sequence. Scenarios are useful if a user wants to include particular sequences of transitions that would not normally appear in the POSTMAN-generated test sequence. The test sequence that POSTMAN produces is guaranteed to contain all the user-specified scenarios, and the overall cost of the test sequence is minimized.

POSTMAN is also able to print the resulting test sequence in either table format (i.e., the standard UNIX® system `tbl` format) or the Tree Tabular Combined Notation (TTCN),[17] a notation offered by the ISO and CCITT standards bodies to represent the tests for communications protocols.

POSTMAN has additional computational features that lead to the generation of even more compact test sequences. An important extension is the ability to *overlap* test subsequences. Because the UIO sequence portion of one test subsequence may coincide with the beginning of a second test subsequence, the overlapping results in a shorter test sequence. Finding an optimal overlap is currently an open research issue, so POSTMAN uses an approximation algorithm to perform overlapping.

POSTMAN can generate test sequences for any deterministic FSM in which any state can be reached from any other state, even if the FSM does not have the reset capability or the self-loop property. For such FSMs, POSTMAN uses an approximation algorithm to find a test sequence that is extremely compact, although not guaranteed to be of absolute minimum cost.

POSTMAN is widely used at AT&T Bell Laboratories to generate conformance tests for various communications protocols, including the BRI and PRI ISDN network-layer protocols, the ISDN LAPD, and the X.25 packet-layer protocol. Typically, the resulting test sequences are three times shorter than ones generated by manual ad hoc methods, even though the ad hoc test sequences may not cover every state transition.

For example, the number of state transitions in the POSTMAN-generated test sequence for the ISDN PRI network layer is reduced from 350 to 115. For the X.25 packet-layer protocol, the test sequence is reduced from over 800 transitions for the manual method to 444 for POSTMAN. This is true even though the manual method omits the state-verification step of every state-transition test, which significantly reduces the error-detection capability of the test.

Application to OSI Conformance Standards

Standards organizations, such as the ISO and CCITT, have developed basic conformance-testing

Table III. Minimum-Cost Test Sequence for Subset of X.25 Packet-Layer Protocol

Test sequence table				
Step	Current state	Next state	Message to IUT	Message from IUT
			Test body for 1_101 in p1	
1	p1	p1	restart.strt_dce	restart.strtc
2	p1	p4	call.call_1_lc	accept.ansr_1_lc
			Test body for 11_102 in p4	
3	p4	p4	data.d_1_lc	rr.rr_1_lc
4	p4	p4	data.d_1_lc	rr.rr_1_lc
			Test body for 7_101 in p4	
5	p4	p1	clear.clr_c1_lc	clearc.clrc_0_lc
6	p1	p4	call.call_1_lc	accept.ansr_1_lc
			Test body for 7_201 in p4	
7	p4	p6	call.call_s_lc	clear.clr_r1_lc
8	p6	p6	Elapse_T23	clear.clr_1
			Test body for 9_310 in p6	
9	p6	p6	resetc.rstc_1_lc	null
10	p6	p6	Elapse_T23	clear.clr_1
			Test body for 9_201 in p6	
11	p6	p6	error.err_5_lc	clear.clr_r1_lc
12	p6	p6	Elapse_T23	clear.clr_1
			Test body for 23_105 in p6	
13	p6	p6	Elapse_T23	clear.clr_1
14	p6	p6	Elapse_T23	clear.clr_1
15	p6	p1	clear.clr_c1_lc	null
			Test body for 1_201 in p1	
16	p1	p1	call.call_0	null
17	p1	p4	call.call_1_lc	accept.ansr_1_lc
			Test body for 7_301 in p4	
18	p4	p6	call.call_z_lc	clear.clr_r1_lc
19	p6	p6	Elapse_T23	clear.clr_1
			Test body for 9_101 in p6	
20	p6	p1	clear.clr_c1_lc	null
21	p1	p4	call.call_1_lc	accept.ansr_1_lc
22	p4	p1	clear.clr_c1_lc	clearc.clrc_0_lc
			Test body for 1_301 in p1	
23	p1	r2	restartc.strtc	restart.strt_dtea
24	r2	r2	Elapse_T20	restart.strtc

Table III. continued

Test sequence table				
Step	Current state	Next state	Message to IUT	Message from IUT
			Test body for 2_301 in r2	
25	r2	r2	rnr.rnr_0_1c	null
26	r2	r2	Elapse_T20	restart.strtc
			Test body for 2_207 in r2	
27	r2	r2	restart.strt_1o	restart.strt_nr
28	r2	r2	Elapse_T20	restart.strtc
			Test body for 23_101 in r2	
29	r2	r2	Elapse_T20	restart.strtc
30	r2	r2	Elapse_T20	restart.strtc
			Test body for 2_101 in r2	
31	r2	p1	restart.strt_dce	null
32	p1	p4	call.call_1_1c	accept.ansr_1_1c
33	p4	p1	clear.clr_c1_1c	clearc.clrc_0_1c
			Test body for 4_101 in p1	
34	p1	p4	call.call_1_1c	accept.ansr_1_1c
35	p4	p4	data.d_1_1c	rr.rr_1_1c
36	p4	p1	clear.clr_c1_1c	clearc.clrc_0_1c

NOTE: Total I/O messages = 36.

concepts for protocol implementations of the ISO's Open Systems Interconnection (OSI) reference model.[25] This effort has resulted in a draft standard[17] that defines the following concepts related to conformance testing of an OSI-protocol implementation in a test laboratory:

- Abstract-test-suite specification
- Abstract-test-suite architectures
- Test-representation languages
- Test realization
- Evaluation.

The OSI standards do not include test-generation techniques for a protocol implementation under test (IUT), because it is assumed that the tester has already developed a set of tests. In addition, the standards do not address the test-generation issue primarily because no agreement has been reached on a formal method. (See Linn[26] and Rayner[27] for descriptions of conformance testing standards.)

In this section, we show how the test-generation technique described earlier is applied to the OSI conformance standards. As a case study, we use a subset of the conformance-testing standard for the DTE side of the X.25 packet-layer protocol.[18]

OSI Abstract-Test-Suite Definitions. Interactions between a tester and an IUT during conformance testing are described in an *abstract test suite*. The representation of tests in an abstract test suite has a hierarchical structure that classifies the tests as follows:

Table IV. Test Purposes for Subset of X.25 Packet-Layer Protocol

Test body	Test-purpose definition	Type
1_101	Verify that DTE accepts restart indication at p1.	Valid
1_201	Verify that DTE restarts packets with LCI=0 in p1.	Illegal
1_301	Verify that restart confirm is restarted with the correct diagnostic code in p1.	Inopportune
4_101	Verify that DTE accepts a valid incoming call in p1.	Valid
2_101	Verify the r2 p1 transition via restart indication.	Valid
23_101	Verify T20 timer operation.	Valid
2_207	Verify that DTE restarts a long Restart at r2.	Illegal
2_301	Verify that DTE discards inopportune packets in r2.	Inopportune
7_101	Send DTE a clear with cause code 01 in p4.	Valid
11_102	Verify that DTE accepts data in p4.	Valid
7_201	Verify that DTE clears a short incoming packet in p4.	Illegal
7_301	Verify that DTE clears an incoming-call packet in p4.	Inopportune
9_101	Send DTE a clear with cause code 01 in p6.	Valid
23_105	Verify T23 timer operation.	Valid
9_201	Verify that DTE discards a short packet.	Illegal
9_310	Verify that DTE discards a reset-confirm packet in p6.	Inopportune

NOTE: The entry in the **Test body** column is the section number in ISO DIS 8882-3.[18]

- *Basic interconnection tests*—provide limited testing to ensure that the IUT can establish a basic interconnection, before thorough testing is performed.
- *Capability tests*—check that the IUT can provide the observable capabilities, based on the *static conformance requirements*. These requirements describe options, ranges of values for parameters and timers, and so on.
- *Behavior tests*—test the *dynamic conformance requirements* of an IUT. These requirements (and options) define the observable behavior of a protocol. The behavior tests include tests of the state transitions, timers, window-rotation mechanisms, and error conditions. Behavior tests constitute the major portion of protocol conformance tests.
- *Conformance-resolution tests*—provide definite diagnostic answers to specific requirements, such as previously identified situations that may cause incorrect behavior of an IUT. For example, these tests provide a yes/no answer about whether a particular feature, such as reset, is implemented in an IUT.

The test-generation method presented in this paper can generate all these tests, if an FSM model that represents the IUT can be created.

Each test in an abstract test suite is based on a well-defined test purpose. A *test purpose* describes the objective of the test, as specifically as possible, in a high-level form (e.g., in English). For example, the expected behavior of an IUT on receipt of an input message in a given state defines a typical test purpose. Details of vari-

ous message exchanges between a tester and an IUT to realize a test purpose are described in a *test case*. Each test case consists of three components: test preamble, test body, and test postamble.

The *test preamble* defines the necessary steps (i.e., input/output messages or packets, etc.) required to put the IUT into the desired starting state of a test case. The preamble corresponds to step A1 of a state-transition test (defined earlier).

The *test body* consists of two logical parts. The first part defines the specific input/output messages sent to or received from an IUT to achieve the objective of the corresponding test purpose. The second part of a test body is the verification of the new state of an IUT after the first part is run. Therefore, the test body covers steps A2 and A3 of the state-transition test given earlier.

The *test postamble* consists of the steps needed to bring the IUT into a stable state after execution of the test body.

As mentioned before, the standards organizations have not adopted formal methodologies for test generation. Typically, this leads to the design of abstract-test-suite definitions that use the protocol's initial state as the stable state for starting the preambles and ending the postambles, and omit the verification routines within test bodies. This approach often produces test suites that require a very long time to run and are not as effective as they should be in detecting errors.

The test-generation technique we have described can be applied to an abstract suite that is defined according to the OSI conformance standards mentioned above. The directed graph that models an IUT can be constructed from the test purposes defined for the IUT. Each input or output message defined by a test purpose corresponds to an edge in the directed-graph representation of an IUT. The UIO sequences generated based on the directed-graph model can be used in the second part of each test body as verification procedures.

To minimize the run-time cost of an abstract-test suite, test bodies can be concatenated to form a tour. We then can define the preamble and postamble of each test case in such a way that they are executed conditionally. The condition for a preamble to be executed is: *during testing, the IUT fails the previous test body in the tour.* If so, the tour is broken and the IUT needs to be initialized to a certain state to continue the tour. Similarly, a postamble is executed only if running a test body results in a fail verdict.

Application of POSTMAN to the ISO DIS 8882-3. The ISO recommendation DIS 8882-3 is the standardized, abstract test suite for conformance testing of the X.25 packet-layer protocol.[18] In this section, we use a small subset of this abstract test suite to illustrate how POSTMAN can minimize the run-time cost and increase the effectiveness of test cases.

Table IV shows the test purposes that we selected for this example. (Definitions for the states and packets of the X.25 packet-layer protocol can be found in DIS 8882-3.[18]) Each test purpose defines an edge of the directed-graph representation of an IUT. In Panel 3, we give the PSL description of the directed graph that corresponds to these test purposes. The UIO sequences generated by POSTMAN for each state are:

```
p1 : call.call_1_1c/accept.ansr_1_1c
r2 : Elapse_T20/restart.strtc
p4 : data.d_1_1c/rr.rr_1_1c
p6 : Elapse_T23/clear.clr_1
```

(These sequences are based on the PSL description in the panel.)

Using these UIO sequences, POSTMAN generates a minimum-cost tour (Table III) that represents the order for running the test bodies. Each step is run consecutively, as indicated by the column called "Step" in the table. The steps that correspond to a test body (i.e., the input and output messages defined by the test purpose and the state-verification procedures) are grouped together.

For example, in Table III, step 1 represents the input and output messages to verify that the IUT accepts

Panel 3. Description of Directed Graph

This PSL description of a directed graph is based on the test purposes defined in Table IV for a subset of the X.25 packet-layer protocol.

```
/* FSM for ISO 8882-3 subset of X.25 Packet Layer Protocol (DTE Side) */
INITIAL_STATE p1;
OTHER_STATES r2, p4, p6;

TRANSITIONS

/* State transitions for state p1 */
  p1:p1  WHEN (restart.strt_dce, restart.strtc) /*8882-3 id. 1_101*/
  p1:p1  WHEN (call.call_0, null) /*8882-3 id. 1_201*/
  p1:r2  WHEN (restartc.strtc, restart.strt_dtea) /*8882-3 id. 1_301*/
  p1:p4  WHEN (call.call_1_lc, accept.ansr_1_lc), /*8882-3 id. 4_101*/
/* State transitions for state r2 */
  r2:p1  WHEN (restart.strt_dce, null) /*8882-3 id. 2_101*/
  r2:r2  WHEN (Elapse_T20, restart.strtc) /*8882-3 id. 23_101*/
  r2:r2  WHEN (restart.strt_lo, restart.strt_nr) /*8882-3 id. 2_207*/
  r2:r2  WHEN (rnr.rnr_0_lc, null), /*8882-3 id. 2_301*/
/* State transitions for state p4 */
  p4:p1  WHEN (clear.clr_c1_lc, clearc.clrc_0_lc) /*8882-3 id. 7_101*/
  p4:p4  WHEN (data.d_1_lc, rr.rr_1_lc) /*8882-3 id. 11_102*/
  p4:p6  WHEN (call.call_s_lc, clear.clr_r1_lc) /*8882-3 id. 7_201*/
  p4:p6  WHEN (call.call_z_lc, clear.clr_r1_lc), /*8882-3 id. 7_301*/
/* State transitions for state p6 */
  p6:p1  WHEN (clear.clr_c1_lc, null) /*8882-3 id. 9_101*/
  p6:p6  WHEN (Elapse_T23, clear.clr_1) /*8882-3 id. 23_105*/
  p6:p6  WHEN (error.err_5_lc, clear.clr_r1_lc) /*8882-3 id. 9_201*/
  p6:p6  WHEN (resetc.rstc_1_lc, null); /*8882-3 id. 9_310*/
```


a restart-indication packet, while the messages in step 2 verify that the next state of the IUT, on completion of step 1, is p1. Some test steps (e.g., steps 22, 33, and 36) are not grouped with a test body, but are used to link test bodies in the minimum-cost tour.

The postambles and preambles can be inserted before and after each test body, respectively, to be run conditionally in the test sequence of Table III. POSTMAN generates a compact form of this table that incorporates the conditional preambles and postambles (represented as `+preamble` and `+postamble`).

Summary

In this paper, we described an algorithmic procedure to generate conformance tests for protocol implementations. The technique is based on UIO sequences

and Rural Postman tours. We showed that the tests generated by this method are of minimum cost and cover all modeled aspects of a protocol implementation.

This technique, as described here, can be applied directly to any abstract test suite that is defined by the standards organizations. To illustrate the use of the UIO sequence-Rural Postman approach, we used as a case study a subset of the conformance-testing standard for the X.25 packet-layer protocol.

The technique we described has been widely used in testing organizations at AT&T Bell Laboratories for conformance testing of various protocol implementations, such as the ISDN BRI and PRI network-layer and X.25 packet-layer protocols and the ISDN LAPD.

Acknowledgments

The conformance-testing approach described here and the POSTMAN software package would not have been possible without the valuable contributions of the following people (in alphabetical order): A. V. Aho, P. Campbell, A. Lapone, D. Lee, F. Lombardi (Texas A&M University), G. W. Mulcahy, K. Nakajima (University of Maryland), A. Sengupta (University of South Carolina), A. B. Sharma, M. H. Sherif, Y.-N. Shen (Texas A&M University), and C. Wu (Harvard University).

References

1. M. H. Sherif and M. U. Uyar, "Protocol Modeling for Conformance Testing: Case Study for the ISDN LAPD Protocol," *AT&T Technical Journal*, Vol. 69, No. 1, January/February 1990, pp. 60-83.
2. Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill Book Company, New York, 1978.
3. S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transition tours," *Proceedings of the 11th IEEE Fault-Tolerant Computing Symposium*, Portland, Maine, June 24 to 26, 1981, IEEE Computer Society Press, New York, 1981, pp. 238-243.
4. B. Sarikaya and G. V. Bochmann, "Some experience with test sequence generation," *Proceedings on Protocol Specification, Testing, and Verification*, Second IFIP WG 6.1 International Workshop, Idyllwild, California, May 17 to 20, 1982, C. Sunshine (ed.), North-Holland Publishing Co., Amsterdam, Holland, 1982, pp. 285-297.
5. M. U. Uyar and A. T. Dahbura, "Optimal Test Sequence Generation for Protocols: The Chinese Postman Algorithm Applied to Q.931," *Communications: broadening technology horizons*, Vol. 1, Proceedings of the IEEE Global Telecommunications Conference, Session 3.1, Globecom '86, Houston, Texas, December 1 to 4, 1986, IEEE Communications Society, New York, 1986, pp. 68-72.
6. H. Ural and R. L. Probert, "User-guided test sequence generation," *Protocol Specification, Testing, and Verification, III*, Proceedings of 3d International Workshop on Protocol Specification, Testing, and Verification, Ruschlikon, Switzerland, May 31 to June 2, 1983, pp. 421-436.
7. E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies*, Annals of Mathematical Studies No. 34, C. E. Shannon and J. A. McCarthy (eds.), Princeton University Press, Princeton, New Jersey, 1956, pp. 129-153.
8. A. Gill, "State-identification experiments in finite automata," *Information and Control*, Vol. 4, 1961, pp. 132-154.
9. A. Gill, *Introduction to the Theory of Finite-State Machines*, McGraw-Hill Book Company, New York, 1962.
10. F. C. Hennie, "Fault-detecting experiments for sequential circuits," *Proceedings of the 5th Annual Symposium on Switching Circuit Theory and Logical Design*, Princeton University Press, Princeton, New Jersey, November 1964, pp. 95-110.
11. T. S. Chow, "Testing software designs modeled by finite-state machines," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 3, March 1978, pp. 178-187.
12. K. K. Sabnani and A. T. Dahbura, "A new technique for generating protocol tests," *Proceedings of the 9th Data Communications Symposium*, Whistler Mountain, British Columbia, September 10 to 13, 1985, IEEE Computer Society Press, New York, September 1985, pp. 36-43.
13. K. K. Sabnani and A. T. Dahbura, "A Protocol Testing Procedure," *Computer Networks*, Vol. 15, No. 4, 1988, pp. 295-297.
14. A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and Chinese postman tours," *Protocol Specification, Testing, and Verification, VIII: Proceedings of the IFIP WG 6.1 Eighth International Symposium on Protocol Specification, Testing, and Verification*, Atlantic City, New Jersey, June 7 to 10, 1988, S. Aggarwal and K. Sabnani (eds.), North-Holland Publishing Co., Amsterdam, Holland, 1988, pp. 75-86.
15. M.-K. Kuan, "Graphic programming using odd or even points," *Chinese Mathematics*, Vol. 1, 1962, pp. 273-277.
16. AT&T, *5ESS® Switch, 5E4 Generic Program, ISDN Basic Rate Interface Specification*, Select code 5D5-900-301, AT&T Customer Information Center, Indianapolis, Indiana, September 1985.
17. "OSI conformance testing methodology and framework," ISO DIS 9646, Parts 1 through 4, International Organization for Standardization, Geneva, Switzerland, 1988.
18. "Information processing systems—X.25 DTE conformance

testing," ISO DIS 8882-3, Part 3, International Organization for Standardization, Geneva, Switzerland, 1988.
19. G. Gonenc, "A method for the design of fault detection experiments," *IEEE Transactions on Computers*, Vol. 19, No. 7, June 1980, pp. 551-558.
20. E. P. Hsieh, "Checking experiments for sequential machines," *IEEE Transactions on Computing*, Vol. C-20, No. 10, October 1971, pp. 1152-1166.
21. P. K. Lala, *Fault Tolerant and Fault Testable Hardware Design*, Prentice-Hall International, Englewood Cliffs, New Jersey, 1985.
22. J. K. Lenstra and A. H. G. Rinnooy Kan, "On general routing problems," *Networks*, Vol. 6, 1976, pp. 273-280.
23. J. Edmonds and E. L. Johnson, "Matching, Euler tours and the Chinese postman," *Mathematical Programming*, Vol. 5, 1973, pp. 88-124.
24. A. M. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, Cambridge, England, 1985.
25. Technical Committee ISO/TC 97, *Information processing systems: open systems interconnection, basic reference model*, Reference No. ISO 7498-1984 (E), International Organization for Standardization, Geneva, Switzerland, 1984.
26. R. J. Linn, Jr., "Conformance Evaluation Methodology and Protocol Testing," *IEEE Journal on Selected Areas in Communication*, Vol. 7, No. 7, September 1989, pp. 1143-1158.
27. D. Rayner, "OSI conformance testing," *Computer Networks and ISDN Systems*, Vol. 14, No. 1, 1988, pp. 79-98.

Biographies (continued)

University. Mr. Uyar's interests include formal testing and verification of communications protocols, expert systems, and parallel processing. He joined Bell Laboratories in 1986 and has a B.S. in electrical engineering from Istanbul Teknik Üniversitesi, Turkey, and both an M.S. and Ph.D. in electrical engineering from Cornell University.

(Manuscript received July 17, 1989)