A Formal Approach to Development of Network Protocols: Theory and Application to a Wireless Standard [*]

M. Ümit Uyar
Electrical Engineering Dept.
The City College of the City University of New York, NY, USA

Mariusz A. Fecko
Applied Research Area, Telcordia Technologies, Inc.,
Morristown, New Jersey, USA

Ali Y. Duale
Engineering Systems Test
IBM Corporation, Poughkeepsie, NY

Paul D. Amer, Adarshpal S. Sethi
Computer and Information Sciences Dept.
University of Delaware, Newark, DE

Abstract

This paper presents the research effort to formally specify, develop and test a complex real-life protocol for mobile network radios (MIL-STD 188-220). As a result, the team of researchers from the University of Delaware and the City University of the City College of New York, collaborating with scientists from CECOM (an R&D facility of the U.S. Army) and the U.S. Army Research Laboratory, have helped advance the state-of-the-art in the design, development, and testing of wireless communications protocols. Estelle is used both as the formal specification language for MIL-STD 188-220 and the source to automatically generate conformance test sequences. The formal test generation effort identified several theoretical problems for wireless communication protocols (possibly applicable to network protocols in general): (1) the timing constraint problem, (2) the controllability problem, (3) inconsistency detection and elimination problem and (4) the conflicting timers problem. Based on the collaborative research results, two software packages were written to generate conformance test sequences for MIL-STD 188-220. These packages helped generate tests for MIL-STD 188-220's Data Link Types 1 and 4 services that were realizable without timer interruptions while providing a 200% increase in test coverage. The test cases have been delivered and are being used by a CECOM conformance testing facility. *Keywords:* conformance testing, Estelle, formal description technique, formal specification, MIL-STD 188-220, protocol specification, test case generation

1 Introduction

Complexity of the wireless protocols used in MIL-STD 188-220, being developed for mobile combat network radios [21], necessitated that a formal approach be taken in protocol specification,

[*]This work supported by the US ARO (DAAH04-94-G-0093), and prepared through collaborative participation in the Advanced Telecommunications/Info Dist'n Research Program (ATIRP) Consortium sponsored by the US Army Research Lab under Fed Lab Program, Cooperative Agreement DAAL01-96-2-0002.

development and testing. Estelle was chosen as the formal specification language to define the protocols in MIL-STD 188-220, from which the conformance tests were automatically generated.

Let us first provide the following data to help the reader realize the magnitude of size and complexity of the wireless protocols used in 188-220. The Datalink and Network layer specifications consist of 69 and 19 documents, respectively, describing the architecture, interfaces, EFSM, and state table of each module. The Datalink layer specification is accompanied by three Estelle source code files (for Datalink classes A, B, and C) with approximately 1,600, 8,700, and 2,400 lines of code, respectively. The Estelle source code for the Network layer has 7,150 lines of code, defining 34 states and 370 transitions in 7 EFSMs (for details, consult [87]).

Automatic test generation from Estelle specifications presented various theoretical problems defined as follows:

- *Timing constraint problem:* During testing, if active timers were not taken into account when the tests were generated, these timers can disrupt the test sequences, thereby failing correct implementations or worse, passing incorrect ones. For accurate testing, timers must be incorporated as constraints into the extended FSM (EFSM) model of an Estelle specification.
- *Controllability problem:* Test sequence generation is limited by the controllability of an Implementation Under Test (IUT) [8]. Testers may not have direct access to all interface(s) in which the IUT accepts inputs. Typically, the interfaces with upper layers, or with timers are difficult or impossible to access during real testing conditions. In this case, some inputs cannot be directly applied; the interactions involving such interfaces may render some portions of the protocol untestable, and may introduce non-determinism and/or race conditions during testing.
- *Inconsistency detection and elimination problem:* Infeasible test sequences may be generated unless possible conflicts among the protocol's variables used in the actions and the conditions are avoided.
- *Conflicting timers problem:* Infeasible test sequences may result from a protocol's variables modeling multiple timers that may be running simultaneously.

The team of researchers and scientists that participated in this research and development effort are from the University of Delaware (UD), the City College of the City University of New York (CCNY), the Army Research Laboratory (ARL), US Army Communications-Electronics Command (CECOM), and the Joint Combat Net Radio Working Group (CNR-WG). As a result of this collaboration, the synergistic framework to develop C^4I (Command, Control, Communications, Computers, and Intelligence) systems with the help of formal methods serves as a model for future U.S. Department of Defense networking standards development [23].

Based on the solutions to these theoretical problems, two software packages, called *efsm2fsm-rcpt*, and (2) INDEEL, have been developed to automatically generate test cases from the EFSM models of Estelle specifications. The sizes of the resulting FSMs derived from the Estelle specifications range from 48 to 303 states, and from 119 to 925 transitions. The corresponding test sequences range from 145 to 2,803 test steps. These tests are free of interruptions due to unexpected timeouts while their coverage of the number of testable transitions increased from approximately 200 to over 700 by utilizing multiple interfaces without controllability conflicts.

Section 2 of this paper presents a part of the Estelle specification of 188-220. A general approach adopted at UD and CCNY to test generation from an Estelle formal specification is described in

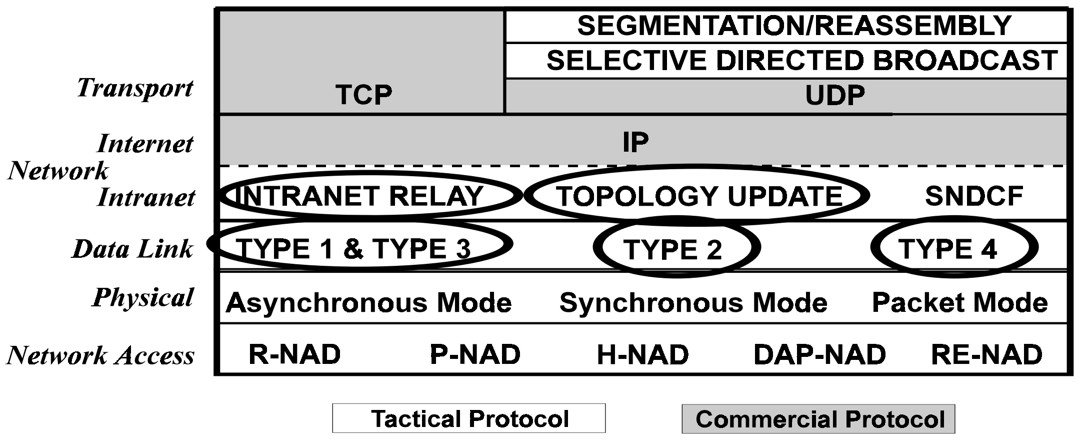


Figure 1: MIL-STD 188-220 Protocol Architecture. The circles indicate those parts of the protocol where FDTs were used during the development.

Section 3. This section also summarizes research results in test generation based on formal specifications. Section 4 presents *efsm2fsm-rcpt*, and INDEEL software systems. Section 5 summarizes our practical test generation results. Finally, Section 6 presents the authors' personal perspective on how the protocol development process is in general improved thanks to using formal methods.

2 Estelle Specification of MIL-STD 188-220

The Protocol Engineering Lab researchers at UD used Estelle to specify parts of the 188-220 protocol suite [3, 21, 14, 53]. 188-220, originally developed in 1993, evolved to 188-220A with substantial new functionality, including support for new radio technology and integration with Internet protocols (commercial IP, TCP, and UDP at the network and transport layers). Version 188-220B, whose architecture is depicted in Figure 1, describes the protocols needed to exchange messages using Combat Network Radio (CNR) as the transmission media. These protocols include the physical, data link and part of the network layer of the OSI model. The protocols apply to the interface between host systems and radio systems. Hosts usually include communications processors or modems that implement these lower layer protocols. The unshaded portions of Figure 1 indicate those protocols and extensions that were developed specifically for use with CNR.

MIL-STD-188-220 Datalink layer specifies several service types, each intended to handle different types of traffic with different quality of service (QoS) demands. A 188-220 station can actually process several different types of traffic simultaneously (and almost orthogonally). MIL-STD-188-220 Network Layer consists of Internet (IP) Layer, Subnetwork Dependent Convergence Function (SNDCF), and Intranet Layer. The Intranet Layer has been dedicated to routing intranet packets between a source and possibly multiple destinations within the same radio network. The Intranet Layer also accommodates the rapid exchange of topology and connectivity information—each node on the radio network needs to determine which nodes are on the network and how many hops away they are currently located.

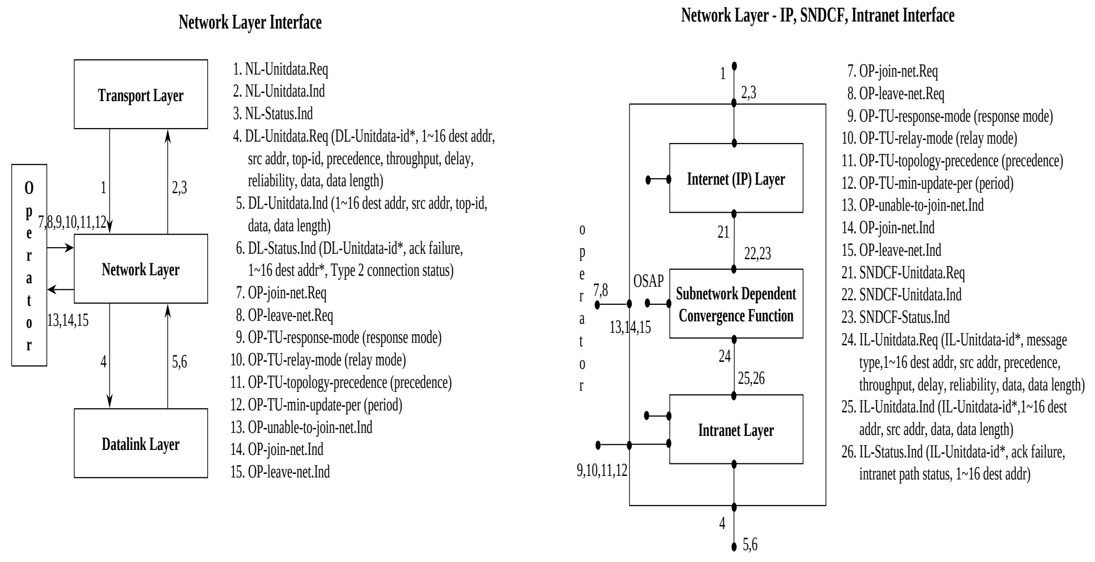

Network Layer Interface
Transport Layer
Network Layer
Datalink Layer
Operator
1
2,3
7,8,9,10,11,12
13,14,15
4
5,6
1. NL-Unitdata.Req
2. NL-Unitdata.Ind
3. NL-Status.Ind
4. DL-Unitdata.Req (DL-Unitdata-id*, 1~16 dest addr, src addr, top-id, precedence, throughput, delay, reliability, data, data length)
5. DL-Unitdata.Ind (1~16 dest addr, src addr, top-id, data, data length)
6. DL-Status.Ind (DL-Unitdata-id*, ack failure, 1~16 dest addr*, Type 2 connection status)
7. OP-join-net.Req
8. OP-leave-net.Req
9. OP-TU-response-mode (response mode)
10. OP-TU-relay-mode (relay mode)
11. OP-TU-topology-precedence (precedence)
12. OP-TU-min-update-per (period)
13. OP-unable-to-join-net.Ind
14. OP-join-net.Ind
15. OP-leave-net.Ind
Network Layer - IP, SNDCF, Intranet Interface
Internet (IP) Layer
Subnetwork Dependent Convergence Function
Intranet Layer
operator
OSAP
1
2,3
21
22,23
7,8
13,14,15
24
25,26
9,10,11,12
4
5,6
7. OP-join-net.Req
8. OP-leave-net.Req
9. OP-TU-response-mode (response mode)
10. OP-TU-relay-mode (relay mode)
11. OP-TU-topology-precedence (precedence)
12. OP-TU-min-update-per (period)
13. OP-unable-to-join-net.Ind
14. OP-join-net.Ind
15. OP-leave-net.Ind
21. SNDCF-Unitdata.Req
22. SNDCF-Unitdata.Ind
23. SNDCF-Status.Ind
24. IL-Unitdata.Req (IL-Unitdata-id*, message type,1~16 dest addr, src addr, precedence, throughput, delay, reliability, data, data length)
25. IL-Unitdata.Ind (IL-Unitdata-id*,1~16 dest addr, src addr, data, data length)
26. IL-Status.Ind (IL-Unitdata-id*, ack failure, intranet path status, 1~16 dest addr)

Figure 2: Network Layer Interface and Architecture

2.1 Intranet Layer Architecture

Figure 2 shows the interface and general architecture of the Network layer. The architecture represents the protocol stack at a single station, as well as an interface with “operator module” which can interact with several different layers in the stack. The operator module abstracts the link layer’s interactions with both a human operator and a system management process.[1]

Figure 3 shows the internal structure of the Intranet Layer. The two main Intranet Layer functionalities, Source Directed Relay (SDR) and Topology Update exchange (TU), were encapsulated in separate component modules of the Intranet Layer module. This simplifies the design of the FSMs that model the entire layer, and also allows for generating test cases for each functionality separately.

The SDR module receives *IL_Unitdata_Req* messages through *SNDCFSAP* interaction point. It starts/stops a varying number of *END_END_ACK* timers, one for each IP packet that has been sent but not yet acknowledged. The TU module interacts with the SDR module by notifying it of any topology changes that take place dynamically. The TU module communicates with two timers: *Topology_Update Timer* and *Topology_Update_Request Timer*. The former is started after a topology update message is sent by the station. According to 188-220A, a station is not allowed to send another topology update message until the timer expires. The latter performs the same role for topology update request messages.

Both SDR and TU modules can send and receive messages from the datalink layer through their *lower_mux* interaction points—the messages from the two modules are multiplexed by the parent Intranet Layer module. A peer operator or management component is connected directly to the Topology Update module and can set parameters that are relevant in topology update mechanism. Part of the diagram inside the dash-lined rectangular contains modules that handle XNP procedures:

[1]Note that the numbers in Figures 2 through 3 refer to interactions, and are consistent throughout the figures (e.g., number 12 refers to *OP-min-update-per* in all three figures).

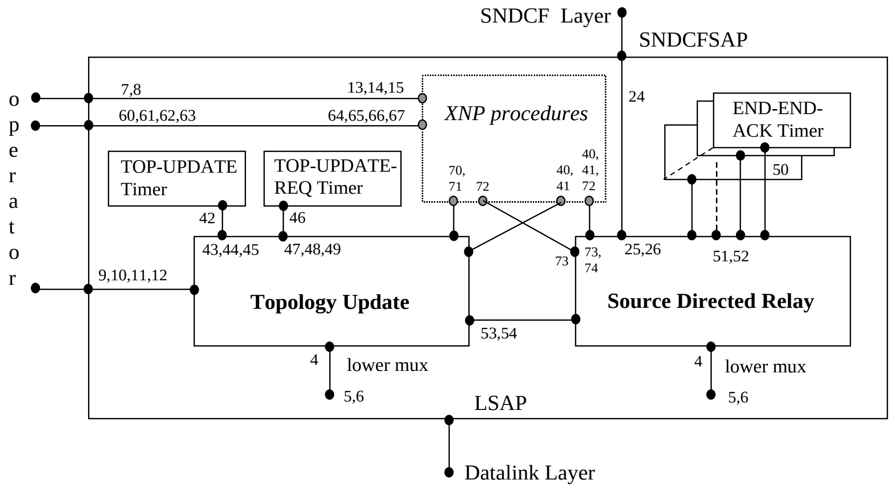


Figure 3: Intranet Layer Architecture

joining and leaving the net with either centralized or distributed control, and parameter update requests.

3 Test Case Generation

Formal methods in communications protocol specification and conformance testing have been widely used in the design and testing of real-life protocols [7, 18, 19, 31, 43, 44, 90]. In particular, the Estelle formal description technique (FDT) [12, 36, 66, 69] has been used on several occasions to resolve ambiguities within international protocols [9, 16, 42, 57, 67, 82].

A number of techniques have been proposed to generate test sequences from Estelle specifications [50, 51, 70, 71, 88]. However, full Estelle specifications of large systems may prove to be too complex for direct test case generation. As shown in Figure 4, there are several ways of generating test sequences from Estelle specifications. One approach would be to *expand* Estelle's EFSMs thereby converting them to pure FSMs. This expansion would be useful since methods exist for generating tests directly from pure FSMs (e.g., [2]). Unfortunately, completely converting even a simple EFSM can result in the state explosion problem, that is, the converted FSM may have so many states and/or transitions that either it takes too long to generate tests, or the number of tests generated is too large for practical use.

As an alternative, the UD and CCNY ATIRP research group used an intermediate approach, where an Estelle EFSM is partially expanded (hence resulting in some more states and transitions), but not expanded completely to a pure FSM. The EFSM is expanded partially just enough to generate a set of tests that is feasible and practical in size. Determining which features to expand in the general case is the difficult aspect of this research.

Test Case Generation Research:

Conformance test generation techniques reported in literature [2, 8, 47, 54, 64, 71], using a deterministic finite-state machine (FSM) model of a protocol specification, focus on the optimization of

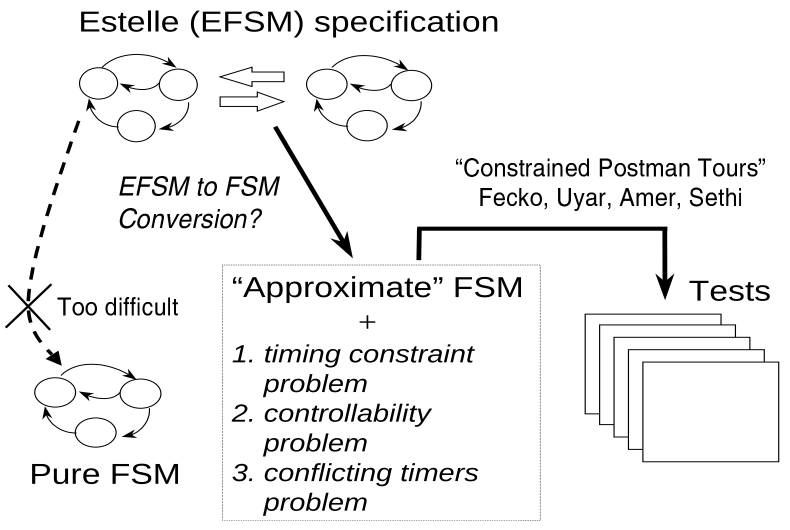

Estelle (EFSM) specification
EFSM to FSM Conversion?
"Constrained Postman Tours"
Fecko, Uyar, Amer, Sethi
Too difficult
"Approximate" FSM
+
1. timing constraint problem
2. controllability problem
3. conflicting timers problem
Tests
Pure FSM

Figure 4: Test Generation from Extended FSMs

the test sequence length. However, an IUT may have timing constraints imposed by active timers. If these constraints are not considered during test sequence generation, the sequence may not be realizable in a test laboratory. As a result, valid implementations may incorrectly fail the conformance tests, or nonconformant IUTs may incorrectly pass the tests.

Another problem in test sequence generation is due to the limited controllability of an IUT. Typically, the inputs defined for the interfaces with upper layers or with timers cannot be directly applied by the tester. In this case, the testability of an IUT may severely be reduced; in addition, non-determinism and/or race conditions may occur during testing.

When a test sequence is to be generated from an EFSM model, one must take into account that the variables used on the actions and conditions may require conflicting values for a given sequence. A test sequence becomes infeasible if there are one or more variables with conflicting values in it. Therefore, possible conflicts among the protocol's variables used in the actions and the conditions must be avoided during test sequence generation.

Another focus point on test sequence generation is the status of different protocol timers at each state (e.g., running, stopped, started, etc.) and the relationship between timers and the actions to trigger them (e.g., start, stop, re-start, or expiry of a timer, etc.). The so-called *conflicting timers problem* addresses that infeasible test sequences may be generated unless conflicting conditions based on timers are resolved.

The remainder of this section presents detailed definitions of these problems and outlines the research progress and the current results.

3.1 The Timing Constraint Problem

During testing, traversing each state transition of an IUT requires a certain amount of time. A test sequence that traverses too many *self-loops* (a *self-loop* is a state transition that starts and ends at the same state) in a given state will not be realizable in a test laboratory if the time to traverse the self-loops exceeds a timer limit as defined by another transition originating in this state. In this case, a timeout will inadvertently trigger forcing the IUT into a different state, and thereby disrupting the test sequence before all of the self-loops are traversed. If this unrealizable test sequence is not avoided during test generation, most IUTs will fail the test even when they meet the specification. Clearly, this is not the goal of testing. Therefore, a properly generated test sequence must take timer constraints into account.

Our research results optimize the test sequence length and cost, under the constraint that an IUT can remain only a limited amount of time in some states during testing, before a timer's expiration forces a state change [77, 78]. The solution first augments an original graph representation of the protocol FSM model. Then it formulates a Rural Chinese Postman Problem solution [52] to generate a minimum-length tour. In the final test sequence generated, the number of consecutive self-loops never exceeds any state's specified limit. In most cases, this test sequence will be longer than one without the constraint since limiting the number of self-loop traversals likely requires additional visits to a state which otherwise would have been unnecessary.

The methodology uses UIO sequences for state verification. However, the results presented also are applicable to test generation that uses distinguishing or characterizing sequences. Earlier results of this study, limited to verification sequences that are self-loops, are presented in [77]. The later paper [78] generalizes these earlier results to both self-loop and non-self-loop verification sequences.

3.1.1 Practical Motivation

Examples of protocols that contain many self-loop transitions in their FSM models include ISDN Q.931 for supplementary voice services, MIL-STD 188-220 [21] for Combat Net Radio communication, and LAPD [79], the data link protocol for the ISDN's D channel. For example, in ISDN Q.931 protocol (Basic voice services, for the user side), each state has an average of 9 inopportune transitions, which requires the traversal of 18 self-loop transitions during testing. A Q.931 implementation has several active timers that are running in certain states, e.g., timer *T304* running in state *Overlap sending*, and timer *T310* in state *Outgoing call proceeding.* An EFSM modeling the Topology Update (TU) functionality of 188-220's Intranet Layer has three active states in which one or two timers are running [77].

It is not always possible to delay the timeout at a tester's convenience. In real protocols, there may be timers whose timeouts are difficult to set by the tester, e.g., acknowledgment timers' timeout values often are computed by the implementation. Moreover, a tester may want to test an IUT's behavior for different settings of the IUT's internal timers, to be able to test the IUT's correctness for various configurations of the timers.

In addition to the original self-loops of a specification model, additional self-loops are typically created when generated test sequences use state verification techniques such as unique input/output (UIO) sequences [63], distinguishing sequences [6, 46], or characterizing sequences [6, 46].

3.1.2 Optimizing Tests under Timing Constraints

Let E_{self} and E_{vnsl} be the sets of self-loop and non-self-loop edges to be tested, respectively. Let $d_{self}(v_i)$, the number of self-loops of vertex v_i, be defined as the number of edges in E_{self}

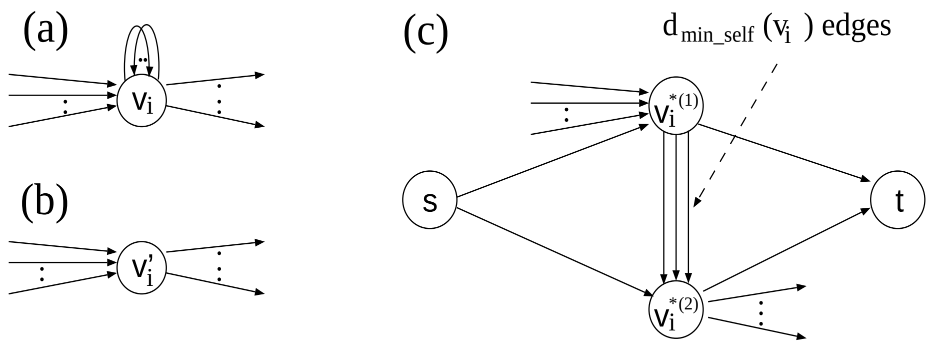


Figure 5: Conversion of v_i in G (part (a)), to $v_i^{'}$ in $G^{'}$ (part (b)) and to $v_i^{*(1)}, v_i^{*(2)}$ in G^* (part (c)).

incident on v_i. Let $d_{min\_self}(v_i)$ be the minimum number of times any tour covering all edges of $E_{vnsl} \cup E_{self}$ must include vertex $v_i \in V$.

Let $d_{state\_ver}(v_i)$ be the number of self-loop transitions used to verify whether an IUT is in state v_i. Suppose that during testing, a given vertex $v_i \in V$ can tolerate at most $max\_self(v_i)$ self-loops executed at one visit to vertex v_i. Attempting to remain in state v_i to execute $1 + max\_self(v_i)$ self-loops would result in disruption of a test sequence. Testing a self-loop transition involves traversing the self-loop transition followed by applying the state verification self-loop sequence, which contains $d_{state\_ver}(v_i)$ transitions.

Due to space limitations, we are unable to include the detailed derivation of $d_{min\_self}(v_i)$. In [77], we prove that the minimum number of times vertex v_i must be visited in a test sequence is as follows:

$$d_{min\_self}(v_i) \quad = \quad \begin{cases} d_{in}(v_i) & \text{if } d_{self}(v_i) \leq (d_{in}(v_i) * \Delta_1(v_i)) \\ \Gamma(v_i) & \text{if } d_{self}(v_i) > (d_{in}(v_i) * \Delta_1(v_i)) \end{cases} \tag{1}$$

where $d_{out}(v_i)$ and $d_{in}(v_i)$ are respectively the out-degree and the in-degree of vertex v_i in E_{vnsl}, and where

$$\Gamma(v_i) \quad = \quad d_{in}(v_i) + \lceil \frac{d_{self}(v_i) - (d_{in}(v_i) * \Delta_1(v_i))}{\Delta_2(v_i)} \rceil \tag{2}$$

$$\Delta_1(v_i) \quad = \quad \lfloor \frac{max\_self(v_i) - d_{state\_ver}(v_i)}{1 + d_{state\_ver}(v_i)} \rfloor \tag{3}$$

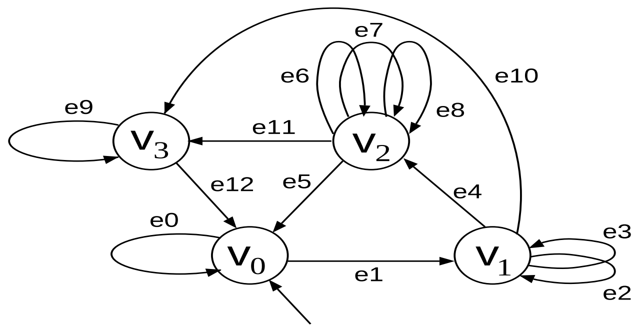
$$\Delta_2(v_i) \quad = \quad \lfloor \frac{max\_self(v_i)}{1 + d_{state\_ver}(v_i)} \rfloor \tag{4}$$

$G^{'}(V^{'}, E^{'})$ ($G^{'}$ is obtained from G by removing self-loop edges) is converted to $G^*(V^*, E^*)$ by splitting each vertex $v_i^{'} \in V^{'}$ satisfying

$$d_{min\_self}(v_i) \quad > \quad max(d_{in}(v_i), d_{out}(v_i)) \tag{5}$$

into the two vertices $v_i^{*(1)}$, $v_i^{*(2)} \in V^*$ (Figure 5).

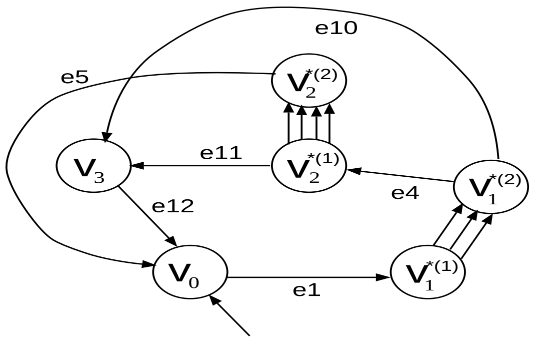
Then, $v_i^{*(1)}$ is connected to $v_i^{*(2)}$ with a set of edges with cardinality of $d_{min\_self}(v_i)$: $E_1^* \overset{def}{=} \bigcup_{v_i^{'} \in V^{'}} g((v_i^{*(1)}, v_i^{*(2)}), d_{min\_self}(v_i))$. Each edge in E_1^* is assigned infinite capacity β and a zero cost ψ. These fake edges will force additional visits to v_i in a minimum-cost tour of G.



Test sequence (34 edges)

e0 **e0** e1 **e2** e2 **e2** e10 **e9** e9 **e9** e12 **e0** e1 e3 **e2** e4
e6 e7 e6 **e6 e7** e11 **e9** e12 e1 e4 e7 **e6 e7** e8 **e6 e7** e5 **e0**

Figure 6: Minimum-cost test sequence without self-loop repetition constraint.



Test sequence (40 edges)

e0 **e0** e1 **e2** e10 **e9** e9 **e9** e12 **e0** e1 e2 **e2** e4 **e6 e7** e11 **e9** e12
e1 e3 **e2** e4 e6 **e6 e7** e5 **e0** e1 e4 e7 **e6 e7** e5 e1 e4 e8 **e6 e7** e5

Figure 7: Minimum-cost test sequence with self-loop repetition constraint.

We then use network flow techniques (similar to Aho et al. [2]) to maximize the flow on graph G^* with minimum cost. This flow defines a minimum-cost tour of G under timing constraints.

Example : *Consider the FSM (represented by the graph $G(V, E)$) with self-loop transitions shown in Figure 6. Suppose that vertices v_0, v_2, and v_3 of the FSM can tolerate at most three, and v_1 at most two self-loop transitions during each visit. Let transitions $e10$ and $e11$ correspond to timeouts. After either $e10$ or $e11$ is triggered, the FSM is brought into state v_3.*

UIO sequences and the values of max_self, d_{state_ver} and d_{min_self} for vertices v_0, v_1, v_2, and v_3 are as follows:

Vertex	UIO	max_self	d_{state_ver}	d_{min_self}
v_0	*e0*	*3*	*1*	*2*
v_1	*e2*	*2*	*1*	*3*
v_2	*e6,e7*	*3*	*2*	*4*
v_3	*e9*	*3*	*1*	*2*

The Chinese postman method [73] when applied to the graph without any self-loop repetition constraint results in the test sequence

$$\underline{e0, \mathbf{e0}, e1, \mathbf{e2}, e2, \mathbf{e2}, e10}, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e3, \mathbf{e2}, e4, \mathbf{e6}, \mathbf{e7},$$

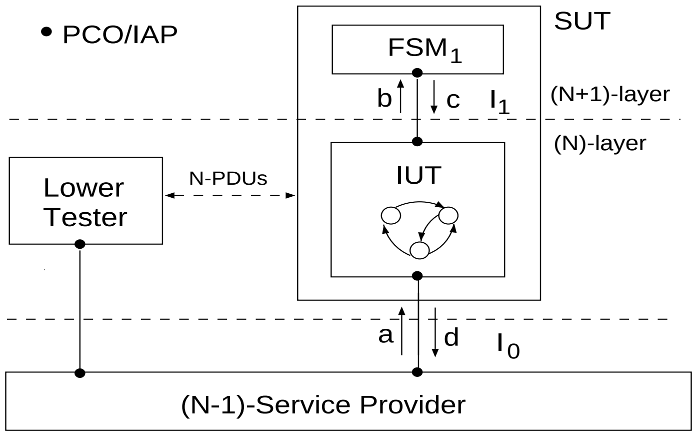


Figure 8: Testing (N)-layer IUT with an (N+1)-layer semicontrollable interface.

$$e6, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e8, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0} \tag{6}$$

containing 34 *edges. Edges used for the purpose of state verification appear in bold.*

As can be seen from the underlined part of the above test sequence, after $e1$ *is traversed, the IUT should stay in state* v_1 *for a time that allows at least three self-loop traversals. However, this part of the test sequence is not realizable in a test laboratory because the timeout edge* $e10$ *will be triggered after the second consecutive self-loop traversal (i.e.,* $max\_self(v_1) = 2$*). The IUT will prematurely move into* v_3 *and the test sequence will be disrupted.*

To address the problem of test sequence disruption due to timeouts, the graph of Figure 6 is converted to the graph shown in Figure 7. Since in this example all UIO sequences are self-loops, the simplified conversion presented in [77] is sufficient. The vertices for which a premature timeout may disrupt a test sequence, which are v_1 *and* v_2*, are split and then connected by* $d_{min\_self}(v_1) = 3$ *and* $d_{min\_self}(v_2) = 4$ *edges, respectively.*

Considering the constrained self-loop problem, the test sequence for the graph of Figure 7 is obtained as

$$\begin{gathered} e0, \mathbf{e0}, e1, \mathbf{e2}, e10, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e2, \mathbf{e2}, e4, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, e1, e3, \\ \mathbf{e2}, e4, e6, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0}, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e5, e1, e4, e8, \mathbf{e6}, \mathbf{e7}, e5 \end{gathered} \tag{7}$$

containing 40 *edges.*

Although longer than that of Figure 6, the test sequence in Figure 7 is minimum-length with the introduced self-loop constraint. During each visit to vertices v_0, v_1, v_2 *and* v_3*, the number of consecutive self-loop edges traversed is less than or equal to the maximum allowed number of self-loop traversals. Therefore, this test sequence is realizable in the test laboratory.*

3.2 The Controllability Problem

Consider a testing framework where the interface I_1 between the IUT and the (N)-layer in the System Under Test (SUT) [8] is not externally accessible (Figure 8). In other words, the inputs from (N+1)-layer cannot be directly applied to the IUT, nor can the outputs generated by the IUT be observed at (N+1)-layer. Such an interface I_1 is called *semicontrollable* if FSM_1 can be utilized to supply inputs to the IUT. On the other hand, the tester can apply inputs to the IUT directly by using a lower tester, which exchanges N-PDUs with the IUT by using the (N-1)-Service Provider. The interface I_0 between the lower tester and the IUT is therefore *directly controllable.*

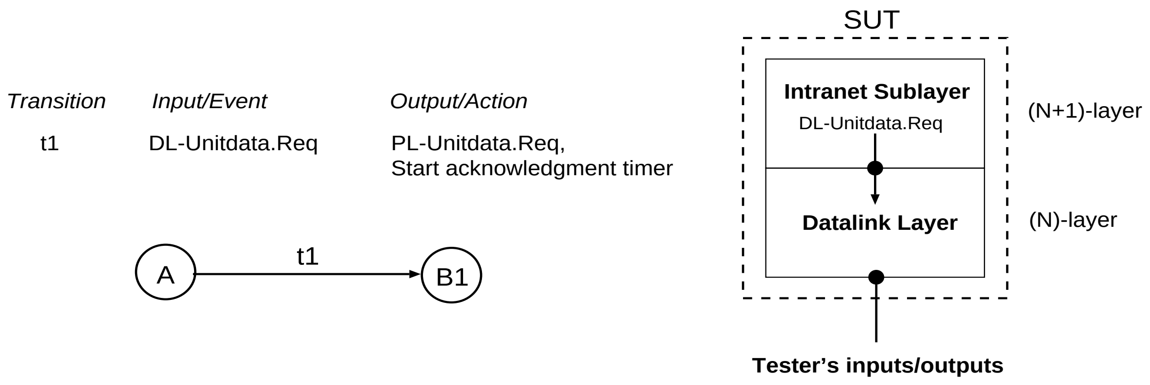


Figure 9: MIL-STD 188-220: Example of the controllability problem

Our approach addresses the problem of generating optimal realizable test sequences in an environment with multiple semicontrollable interfaces [26]. The methodology fully utilizes semicontrollable interfaces in an IUT while avoiding the race conditions. An algorithm is introduced in [26] to modify the directed graph representation of the IUT such that its semicontrollable portions become directly controllable, where possible. In the most general case, obtaining such a graph conversion may end up with exponentially large number of nodes. However, it is shown [26] that special considerations such as the small number of interfaces interacting with an IUT and diagnostics considerations make the problem size feasible for most practical cases.

3.2.1 Practical Motivation

As motivation for solving the controllability problem, a real protocol is considered where an SUT's (N+1)-layer must be utilized indirectly to test certain transitions within the (N)-layer IUT.

188-220 focuses on 3 layers: Physical, Datalink, and Network. The Network layer contains an Intranet sublayer. An SUT contains the (N)-layer IUT implemented in the Datalink layer, and the Intranet sublayer, which is part of the (N+1)-layer, as shown in Figure 9.

In the CECOM's environment used for testing 188-220 implementations, the upper layers cannot be directly controlled. Therefore, the IUT's transitions that are triggered by the inputs coming from the Network layer are not directly testable. An example SUT transition that causes a controllability problem is the transition $t1$ from the Class A–Type 1 Service Datalink module [21, 24], shown in Figure 9. The *input/event* field for this transition requires a *DL_Unitdata_Req* from the (N+1)-layer. Unfortunately, the interface between the IUT and the (N+1)-layer is not directly accessible for generating this input. Initially, it appears that transition $t1$ is untestable.

To trigger this transition, which requires the (N+1)-layer to pass a *DL-Unitdata.Req* down to the (N)-layer, feedback from the (N+1)-layer must be used. To force a *DL-Unitdata.Req* from the (N+1)-layer, the tester sends a *PL-Unitdata.Ind* to the IUT (similar to the message *a* in Figure 8) that contains an intranet layer message telling the (N+1)-layer to relay the frame to a different network node. The IUT outputs this message to the (N+1)-layer (see message *b* in Figure 8), and the (N+1)-layer FSM responds by outputting the desired *DL-Unitdata.Req* (message *c* in Figure 8). Finally, the datalink layer generates the desired output *PL-Unitdata.Req* (corresponding to message *d* in Figure 8), which can be observed by the lower tester.

In fact, 70% of the transitions the Class A–Type 1 Datalink Service module are based on not directly controllable inputs. Without indirect testing, test coverage would be seriously limited; only approximately 200 transitions out of 750 would be testable. However, by applying the technique outlined in this paper, over 700 of defined transitions (>95%) can be tested. The application of the presented technique to 188-220 is described in more detail in [25].

Similar controllability problems can also be pointed out in testing the IEEE 802.2 LLC Connection Component [26, 40].

3.2.2 Optimizing Tests with Multiple Semicontrollable Interfaces

To optimize tests with multiple semicontrollable interfaces, modeling SUT as a single FSM was proposed [26, 27]. A semicontrollable interface I_i is implemented as a separate FIFO buffer. During testing, a buffer may be empty or store an arbitrary sequence of inputs to the IUT generated indirectly through I_i. For each I_i, we define variable ω_i that has a distinct value for each permutation of inputs that the i-th buffer can hold. The proposed model consists of graph G (which represents the IUT's FSM) and the variables $\omega_1, \omega_2, \ldots, \omega_F$.

An FSM modeling the SUT can be obtained by expanding G and $\omega_1, \omega_2, \ldots, \omega_F$ into $G^{'}(V^{'}, E^{'})$. An algorithm for converting $G(V, E)$ to $G^{'}(V^{'}, E^{'})$ proceeds as follows (a detailed description of the algorithm along with its pseudocode is available in [26, 27]):

Step 0—Definitions:
Let B_i denote a sequence of inputs buffered at the i-th semicontrollable interface. Each state $v^{'} \in V^{'}$ has two components: the original state $v \in V$, and the current configuration of F buffers, i.e., $v^{'} = (v, \hat{B}_1, \ldots, \hat{B}_F)$. The algorithm constructs all possible buffer configurations with up to b_i inputs buffered at I_i.

Step 1—Initialize:
$r^{'}$, root of $G^{'}$, as $(r, \emptyset, \cdots, \emptyset)$ (root of G and configuration of empty buffers); $E^{'}$ as empty set; $V^{'}$ as $\{r^{'}\}$; Q, queue of vertices, as $V^{'}$

Step 2—Repeat until Q is empty:

1. extract $v^{'} = (v_{start}, \hat{B}_1, \ldots, \hat{B}_F)$ as first element from Q, where $(\hat{B}_1, \ldots, \hat{B}_F)$ is current configuration
2. given the current vertex $v^{'} = (v_{start}, \hat{B}_1, \ldots, \hat{B}_F)$, perform the following steps for each original outgoing edge $e = (v_{start}, v_{end}) \in E$:
    - create new configuration $(B_1, \ldots, B_F)$ based on the class of e (Figure 10):
        - **Class 1:** e is triggered by an input from and generates output(s) to an LT;
        - **Class 2:** e is triggered by an input from an LT and generates an output $o_{q,l}$ (buffered in B_q to create a new configuration) at I_q;
        - **Class 3:** e is triggered by $a_{p,k}$ (extracted from B_p to create a new configuration) from I_p and generates output(s) to an LT;
        - **Class 4:** e is triggered by an input $a_{p,k}$ from I_p and generates an output $o_{q,l}$ at I_q. Apply rules for *Class 3* and *Class 2* to create a new configuration.
    - create new vertex $v^{'}_{new} = (v_{end}, B_1, \ldots, B_F) \in V^{'}$, and new edge $e^{'}_{new} = (v^{'}, v^{'}_{new}) \in E^{'}$
    - include new edges in $E^{'}$ iff inputs in $(\hat{B}_1, \ldots, \hat{B}_F)$ cannot trigger other edges outgoing from v_{start}
    - append to Q end vertices $v^{'}_{new} \in V^{'}$ of new edges included in $E^{'}$

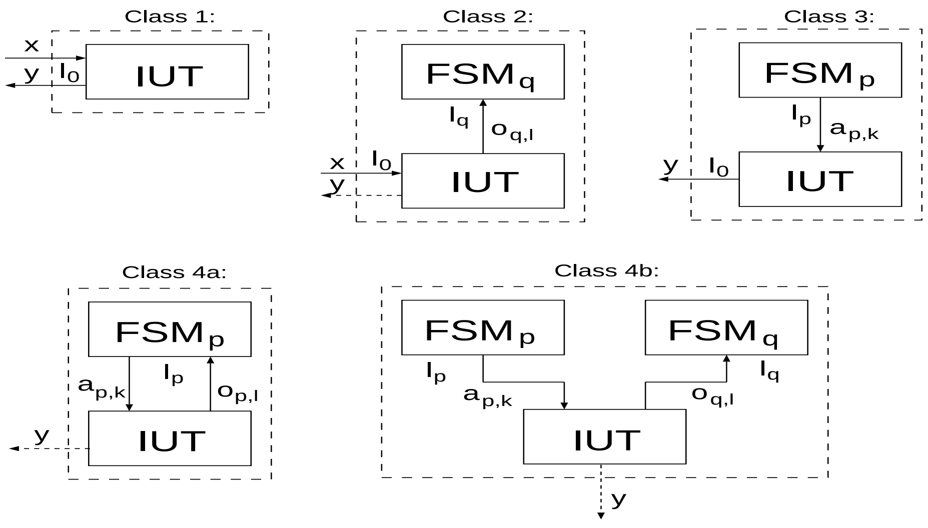


Figure 10: Classes of edges in $G^{'}$ (dashed-lined outputs are optional).

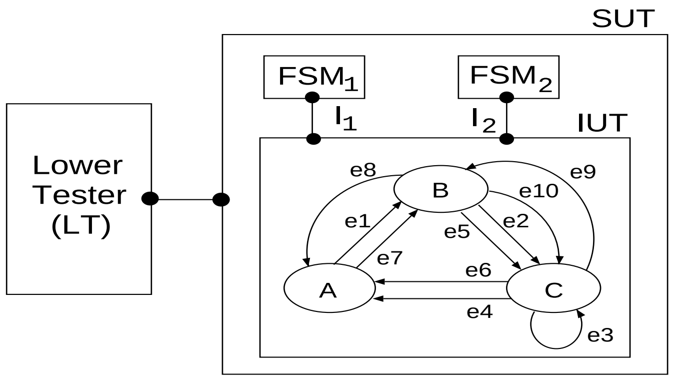


Figure 11: IUT interacting with two semicontrollable interfaces.

Step 3—Retain only strongly connected states:
remove from $V^{'}$ all vertices from which $r^{'}$ cannot be reached, and remove from $E^{'}$ all edges incident to such vertices

Based on the practical considerations discussed in [26], the algorithm can be refined to meet the following objective: *"generate a test sequence that, at any point in time, avoids storing more than one input in only one of the buffers (where possible)."* Satisfying this objective yields a linear running time in the number of semicontrollable interfaces and the number of edges in G. If this objective cannot be satisfied, the running time grows and nondeterminism may not be avoided during testing.

Example : *Consider the IUT of Figure 11 which is interacting with semicontrollable FSM_1 and FSM_2 through the semicontrollable interfaces I_1 and I_2, respectively. The IUT's FSM (represented by graph G) is described in Table 1. Transition $e1$, triggered by input x_1 from the lower tester, generates output $o_{1,1}$ to FSM_1. In response, FSM_1 sends input $a_{1,1}$ which triggers transition $e3$. (In general, $a_{i,j}$ is the expected response to $o_{i,j}$.) Transition $e2$, which is triggered by a lower tester's input x_2, outputs $o_{2,1}$ to FSM_2, which responds with input $a_{2,1}$ triggering $e4$. Then $e4$ outputs $o_{1,2}$ to FSM_1, which responds with $a_{1,2}$ triggering $e8$. On the other hand, transitions $e5$, $e6$, $e7$, $e9$, and $e10$, can be triggered directly by the lower tester. $e6$, $e7$,*

Table 1: Inputs and outputs for the edges of Figure 11. $A?x$ denotes receiving input x from A. $B!y$ denotes sending output y to B.

Edge	*Input*	*Output*	*Edge*	*Input*	*Output*
e1	$LT?x_1$	$FSM_1!o_{1,1}$	e6	$LT?x_6$	$LT!y_6$
e2	$LT?x_2$	$FSM_2!o_{2,1}$	e7	$LT?x_7$	$LT!y_7$
e3	$FSM_1?a_{1,1}$	$LT!y_3$	e8	$FSM_1?a_{1,2}$	$LT!y_8$
e4	$FSM_2?a_{2,1}$	$FSM_1!o_{1,2}$	e9	$LT?x_9$	$LT!y_9$
e5	$LT?x_5$	$FSM_2!o_{2,2}$	e10	$LT?x_{10}$	$LT!y_{10}$

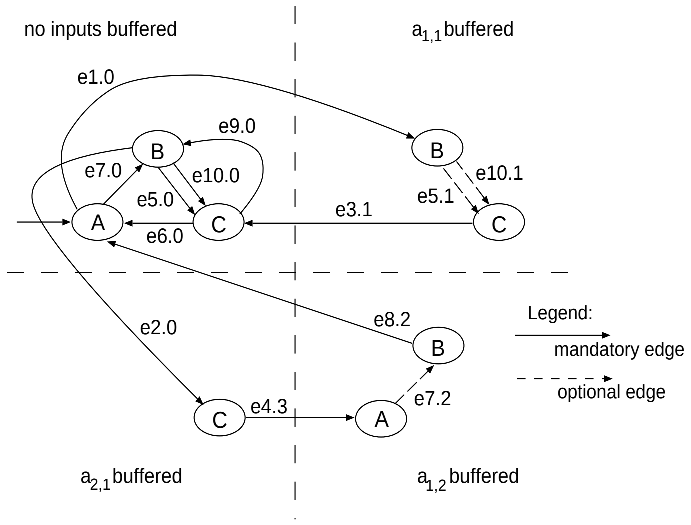


Figure 12: Graph transformation applied to the graph of Fig. 11. Mandatory and optional edges appear in solid and dashed lines, respectively.

Table 2: Minimum-length test sequence for the IUT of Figure 11.

Step	*Edge*	*Input*	*Output*	*Step*	*Edge*	*Input*	*Output*
$\rightarrow 1$	e1.0	$LT?x_1$	$FSM_1!o_{1,1}$	8	e7.2	$LT?x_7$	$LT!y_7$
2	e5.1	$LT?x_5$	$FSM_2!o_{2,2}$	$\rightarrow 9$	e8.2	$FSM_1?a_{1,2}$	$LT!y_8$
$\rightarrow 3$	e3.1	$FSM_1?a_{1,1}$	$LT!y_3$	10	e7.0	$LT?x_7$	$LT!y_7$
$\rightarrow 4$	e6.0	$LT?x_6$	$LT!y_6$	$\rightarrow 11$	e5.0	$LT?x_5$	$FSM_2!o_{2,2}$
$\rightarrow 5$	e7.0	$LT?x_7$	$LT!y_7$	$\rightarrow 12$	e9.0	$LT?x_9$	$LT!y_9$
$\rightarrow 6$	e2.0	$LT?x_2$	$FSM_2!o_{2,1}$	13	e10.0	$LT?x_{10}$	$LT!y_{10}$
$\rightarrow 7$	e4.3	$FSM_2?a_{2,1}$	$FSM_1!o_{1,2}$	14	e6.0	$LT?x_6$	$LT!y_6$

$e9$, and $e10$, do not generate outputs to the semicontrollable interfaces. $e5$ generates output $o_{2,2}$ to FSM_2, which does not send any input to the IUT.

After conversion (Figure 12), each state of G is replaced with at most four related states in $G^{'}$ corresponding to the buffer configurations at a semicontrollable interface. Each edge e is annotated as $e.x$, where $x = 0, 1, 2, 3$, depending on the input buffered in the $e.x$'s start state, as shown in Figure 12. The solid edges in Figure 12 are the mandatory edges that are incident to nodes that correspond to the case where both buffers are empty; the dashed-line edges are the ones that can be traversed only when either buffer contains an input. Due to the practical diagnostic considerations [26], we prefer testing edges when no inputs are buffered in semicontrollable interfaces. The Aho et al. [2] optimization technique gives the minimum-length test sequence for $G^{'}$ shown in Table 2. Steps with $(\rightarrow)$ indicate that an edge is tested in this step. Note that, for simplicity, the UIO sequences [63] are not included in this sequence.

3.3 Inconsistency Detection and Elimination Problem

Feasible test sequence generation is essential for assuring the proper operation and interoperability of different components in computer and communication systems. The use of formal description languages such as VHDL and Estelle enable the precise description of such systems and help minimize the implementation errors due to misinterpretations. However, the specifications written in VHDL and Estelle are often extended FSMs (EFSMs), making the automated test generation a more complex task due to possible inconsistencies among the action and condition variables [22].

Test generation from EFSM models has been an active research area. Sarikaya *et al.* used the functional program testing approach to generate test sequences from the EFSMs [64]. Software data flow testing approaches have been used to generate tests for the communication protocols [70, 71, 72]. Ural applied the all-uses [72, 61] criterion, used for testing software written in block structured programming languages, to Estelle [12, 13] specification of protocols. Miller and Paul [54] introduced a method to generate tests for both control and data for EFSM models. Chanson and Zhu [15] proposed a test generation method which considers both the control flow and data aspects of the EFSM models. Lee and Yannakakis [49] provided a method to convert a class of EFSMs, where input variables are assumed to have finite domains, into equivalent FSMs. For a restricted class of LOTOS expressions, called P-LOTOS, Higashino and Bochmann propose a method that provides solutions to a set of interrelated problems such as the test case derivation and the detections of nonexecutable branches, deadlocks, and nondeterminism [32].

We studied the problem of generating feasible test sequences for the EFSM by analyzing the interdependencies among the action and condition variables of the EFSM models. In the earlier phases of this research, *action* and *condition* inconsistencies in the EFSM models were defined [80, 81]. It has been shown that once the inconsistencies are eliminated, the existing finite-state machine (FSM)-based test generation methods can be used to generate feasible test sequences from the resulting consistent EFSM graphs.

The algorithms for the detection and elimination of *conflicts* in EFSM models utilize *symbolic execution*, *linear programming*, and *graph splitting* methods. After all conflicts are eliminated, all paths of the final resulting EFSM graph are feasible and can be used as an input to the FSM-based test generation methods. The basic concepts for the inconsistency elimination algorithms were outlined in [81], which were later generalized to include graphs with loops [76]. The formal descriptions of the inconsistency detection and elimination algorithms have been given in [22].

3.3.1 Action Conflicts

If there is no solution for the set of equations formed by the actions of an edge e_i and the condition of another edge e_j, where $head(e_j)$ can be reached from $tail(e_i)$ or $head(e_j)$ = $tail(e_i)$, then the two edges of e_i and e_j are said to have an **action conflict** (for an edge directed from node v_a to node v_b, the head and tail nodes are defined as v_a and v_b, respectively).

For example, in Figure 13, there is an **action conflict** between the action of e_1 and the condition of e_8 due to variable b.

In general, the effects of the edge actions on variables (i.e., variable modifications) can be represented as matrices. For an EFSM graph with m variables, $var_1, var_2, \ldots, var_m$, a pair of matrices $A(m\text{x}m)$ and $\tilde{B}(m\text{x}1)$ called *the modification matrix* and *the modification vector*, respectively, are defined.

The *accumulated effects* of the actions in the paths leading to a node v_i can be represented by a set of *Action Update Matrix* pairs defined as:

$$\text{AUM}(v_i, J) = \{A_{v_i,0}, \tilde{B}_{v_i,0}, A_{v_i,1}, \tilde{B}_{v_i,1}, \cdots, A_{v_i,J-1}, \tilde{B}_{v_i,J-1}\} \quad (8)$$

where $A_{v_i,k}$, $\tilde{B}_{v_i,k}$, and J are the k^{th} modification matrix, k^{th} modification vector ($0 \leq k < J$), and the number of AUM pairs associated with v_i, respectively. The symbolic values of a variable var_r are represented in the r^{th} rows in AUM(v_i, J). Only one AUM pair, where A and $\tilde{B}$ are initialized to the identity matrix and to a zero vector, respectively, is created for the initial node.

The number of AUM pairs associated with v_i solely depends on the number of different ways in which the actions of the edges leading to v_i modify variables. If the overall variable modifications of the actions of any two paths leading to v_i are the same, only one AUM pair is sufficient to account for the effects of the actions in the two paths. Therefore, only unique AUM pairs are associated with v_i. See Section 3.3.3

Symbolic execution is utilized in the construction of AUM pairs. When an action conflict is detected, the EFSM graph is split from the node where the conflict occurs. The analysis continues until all action conflicts are eliminated. By applying the algorithms for action conflict detection and elimination as presented in [22, 76], the resulting EFSM graph after the action conflict is eliminated is shown in Figure 14.

3.3.2 Condition Conflicts

After all action conflicts are eliminated from the EFSM graph, the next step involves the detection and elimination of condition conflicts. The edges e_i and e_j are said to have a condition conflict if there is no solution for the set of equations formed by the accumulated conditions of the edges of a sub-path $e_1 \cdot e_2 \cdot \ldots \cdot e_i$ and an edge e_j, where $head(e_j)$ can be reached from $tail(e_i)$ or $head(e_j)$ = $tail(e_i)$.

In Figure 14, for example, there is a condition conflict between the edges of $e_{5(0)}$ and $e_{4(1)}$ since each edge requires a conflicting value of variable c. Figure 15 shows the final conflict-free EFSM graph.

Since the conditions of the edges of a test sequence constitute a system of constraints, simplified version of linear programming algorithms can be used in deciding whether a certain path predicate is

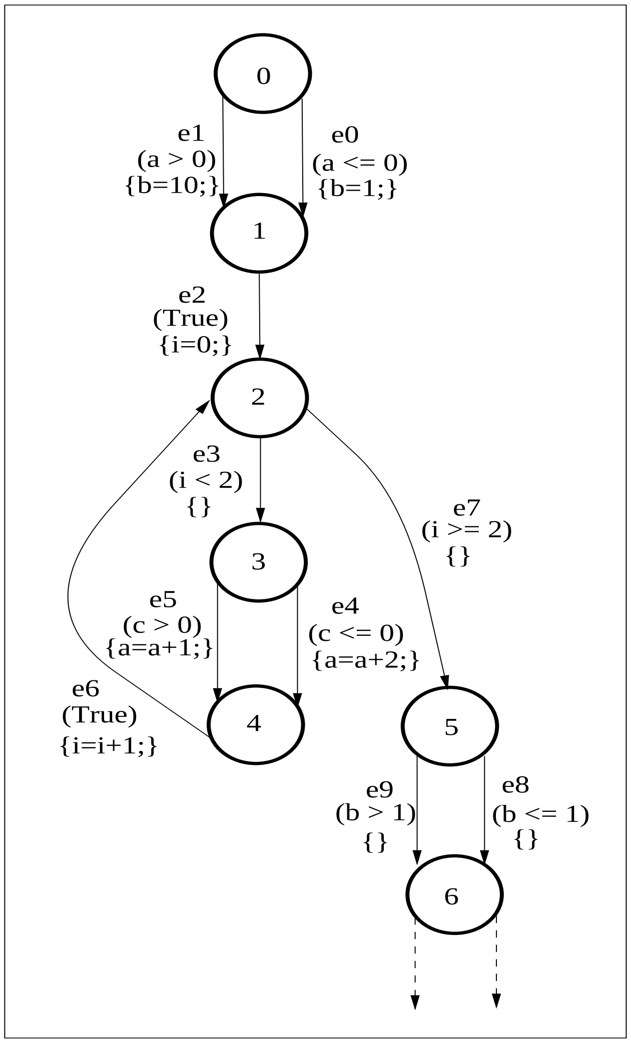


Figure 13: An EFSM graph with conflicts

feasible [1]. The edge conditions in a path from the starting node v_0 to a node v_i can be represented in matrices. A triplet of matrices are defined as C (mxp), $\tilde{OP}$ (px1), and $\tilde{D}$ (px1), where m is the number of variables, p is the number of conditions in the path from v_0 to v_i, C is the coefficient matrix, $\tilde{OP}$ is the operator vector containing the relations of $=, <, >, ! =, \cdots$, etc., and $\tilde{D}$ is the scalar vector containing the scalar values of the conditions in the path.

The AUM pairs discussed in Section 3.3.1 are applied to the edge conditions of the EFSM graph as follows. A single condition of an edge $e_r = (v_i, v_j)$ is in the form of $\tilde{C} * \tilde{V}(\tilde{OP})\tilde{D}$. The condition of e_r will be modified based on the symbolic values of the variables var_0 through v_{m-1}, which are represented by the AUM(v_i, J). The current values of the variables including all the modifications represented by an AUM pair of v_i are in the form of: $\tilde{V} = A_{v_i,k} * \tilde{V} + \tilde{B}_{v_i,k}$. Substituting $\tilde{V}$ values in an edge condition will result in $\tilde{C}(A_{v_i,k} * \tilde{V} + \tilde{B}_{v_i,k})(\tilde{OP})\tilde{D}$, which simplifies as $\tilde{E} * \tilde{V}(\tilde{OP})f$, where $\tilde{E} = \tilde{C} * A_{v_i,k}$ is an m-element vector and f is a scalar. An edge $e_r = (v_i, v_j)$ whose condition is infeasible based on the AUM pairs of v_i is deleted from the graph. The values assumed by the variables used in the condition of e_r can be determined from:

$$C * \tilde{V} = C * (A_{v_i,k} * \tilde{V} + \tilde{B}_{v_i,k}) \tag{9}$$

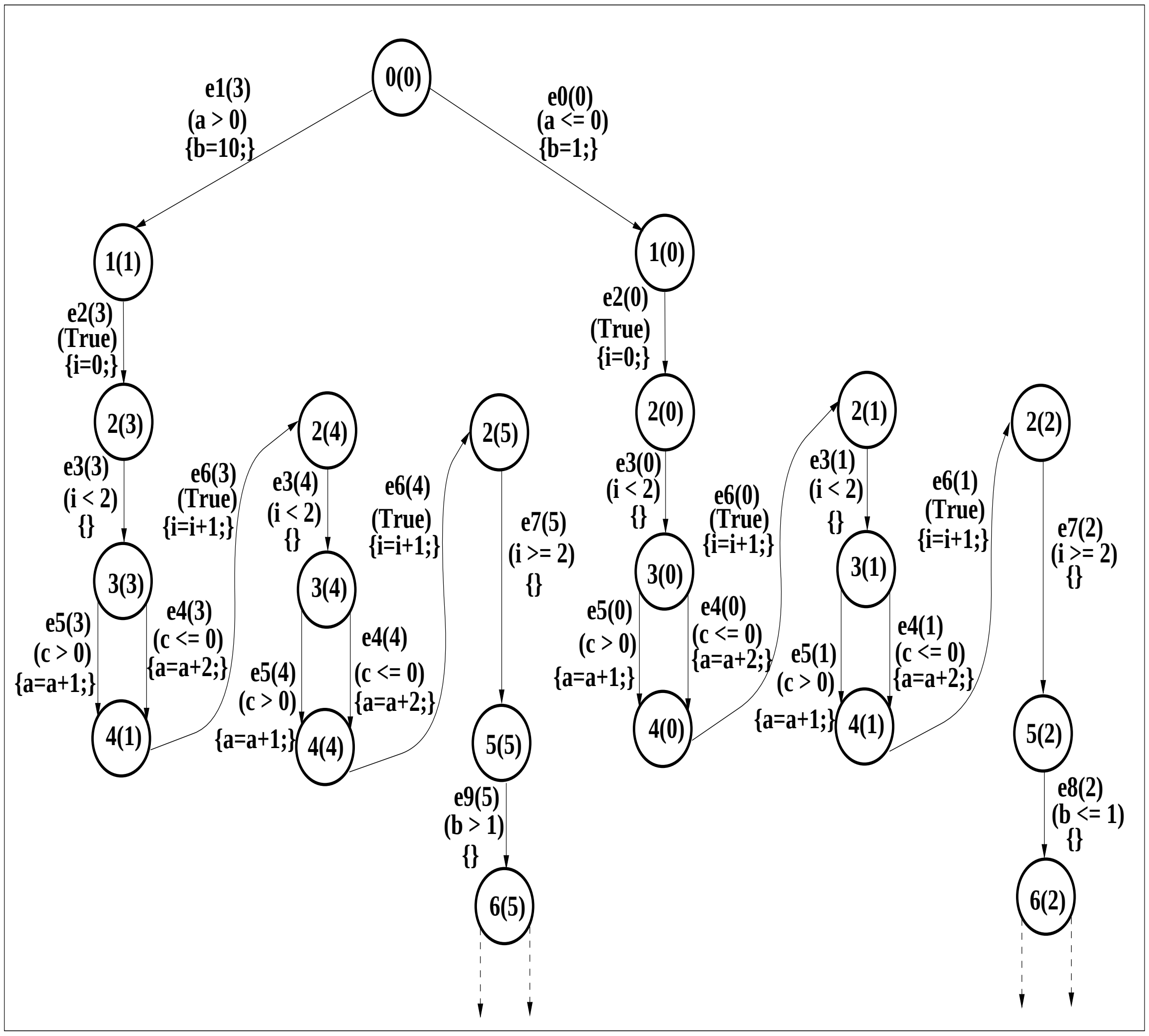


Figure 14: The resulting EFSM graph after splitting the graph of Figure 13 due to e_1 action and e_8 condition.

where C is the coefficient matrix for the condition of e_r and $0 \leq k < J$, where J is the number of AUM pairs associated with v_i.

The accumulated different conditions of the paths leading to v_i can be represented by a set of *Accumulated Condition Matrix* (ACM) triplets: ACM(v_i, J) = ($C_{v_i,0}$, $\tilde{OP}_{v_i,0}$, $\tilde{D}_{v_i,0}$, $C_{v_i,1}$, $\tilde{OP}_{v_i,1}$, $\tilde{D}_{v_i,1}$, $\cdots$, $C_{v_i,J-1}$, $\tilde{OP}_{v_i,J-1}$, $\tilde{D}_{v_i,J-1}$), where $C_{v_i,k}$, $\tilde{OP}_{v_i,k}$, $\tilde{D}_{v_i,k}$, and J are the k^{th} coefficient matrix, k^{th} operator matrix, k^{th} scalar value matrix ($0 \leq k < J$), and the number of the ACM triplets associated with v_i, respectively.

3.3.3 Complexity of the Algorithms

The action inconsistency detection and elimination algorithms use a two-phase modified breadth-first graph traversal, called P1-MBF and P2-MBF. P1-MBF is the main graph traversal from which P2-MBF may be invoked multiple times. During the condition inconsistency detection phase, the graph is traversed in a regular depth-first manner.

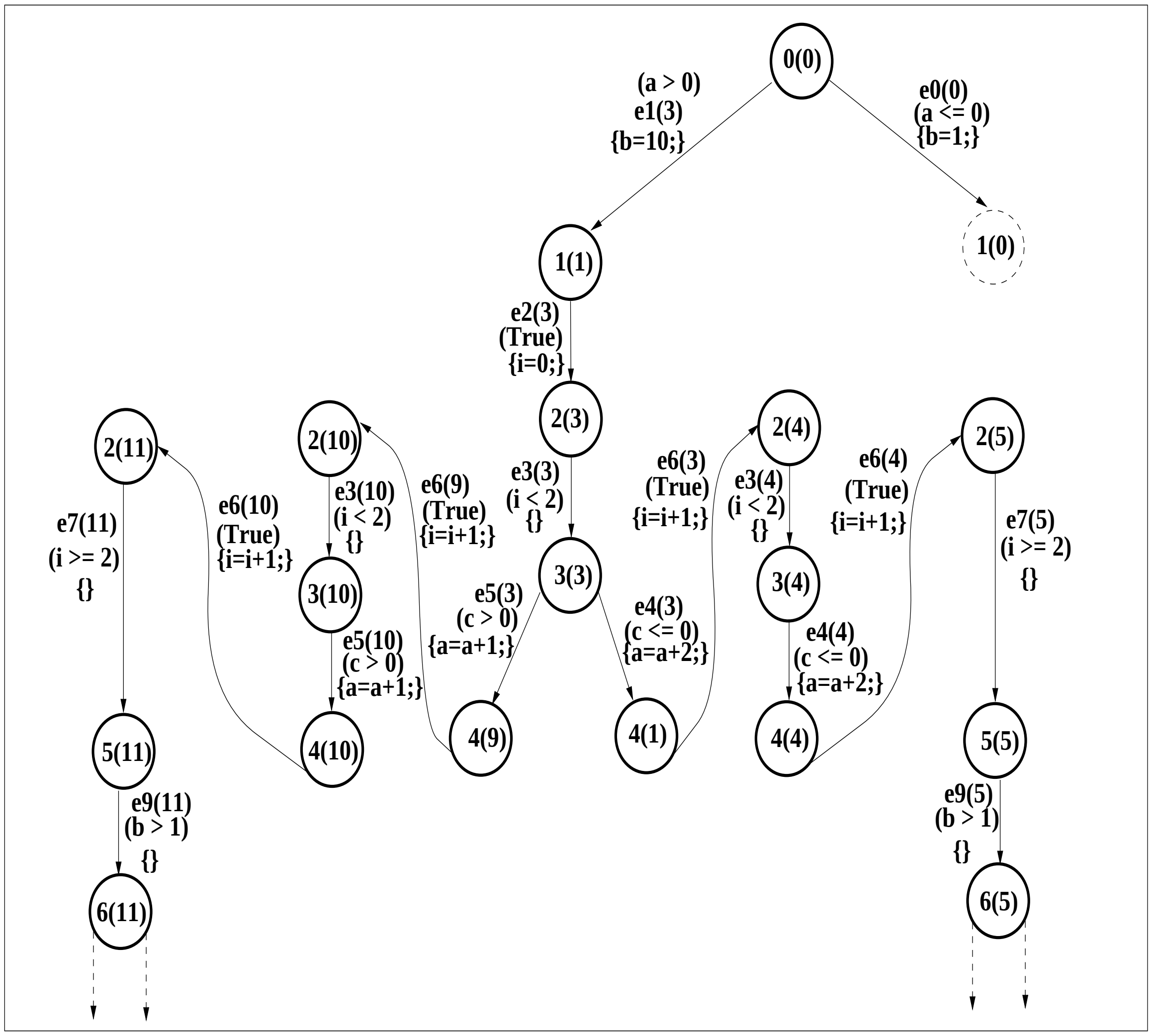


Figure 15: The final conflict-free EFSM graph after all conflicts are eliminated (the subgraph starting from the node $v_{1(0)}$ is not shown for simplicity)

The complexity of the action conflict detection and elimination is contributed by a two-phase MBF graph traversal and constructing the number of AUM pairs for each node, for each edge for each AUM pair.

The complexity for the two-phase MBF graph traversal is $O(E^2)$ [20]. For each node v_i, the number of AUM pairs is $\sum_1^{|V|-1} |E^{v_j \to v_i}| \times |\text{AUM}(v_j, J)|$ (where $|E^{v_j \to v_i}|$ is the number of edges from v_j to v_i).

The complexity for the condition conflict detection and elimination is bounded by the number of AUM pairs of each node and executing the linear programming for each edge. A simplified version of linear programming, which is used to eliminate infeasible conditions, takes min(m^2, S^2) steps where m is the number of variables and S is the number of constraints [1].

Therefore, for the general case, the complexity of algorithms for handling the action conflicts is exponential with respect to the number of simple paths (i.e., the number of AUM pairs). Similarly, the condition conflict elimination can be exponential with respect to the number simple paths. However, based on our experience with several protocols (including nested and/or concatenated loops), the complexity of both algorithms and, hence, the size of the final conflict-free graph are bounded by the number of different values each condition variable assumes as it is used in the conditions.

3.4 The Conflicting Timers Problem

To ensure feasibility of tests in a laboratory, automated test generation for network protocols with timer requirements must consider conflicting conditions based on a protocol's timers. Our ATIRP research developed a new model for testing real-time protocols with multiple timers, which captures complex timing dependencies by using simple linear expressions involving timer-related variables. Similar dependencies, but based on arbitrary linear variables, are present in EFSM models of VHDL specifications [74]. Uyar and Duale present algorithms for detecting [74] and removing [22, 81] such inconsistencies in VHDL specifications. The new modeling technique combined with the inconsistency removal algorithms are expected to significantly shorten test sequences without compromising their fault coverage.

The model, specifically designed for testing purposes, avoids performing a full reachability analysis and significantly limits the explosive growth of the number of test scenarios. These goals are achieved by incorporating certain rules for the graph traversal without reducing the set of testable transitions. The technique also models a realistic testing framework in which each I/O exchange takes a certain time to realize, and a tester has an ability to turn timers *on* and *off* in arbitrary transitions and to algorithmically find proper timeout settings.

The methodology presented in this paper is expected to detect transfer and output faults [49], where an IUT moves into a wrong state (a state other than the one specified) or generates a wrong output (an output other than the one specified) to a given input, respectively. The detection of transfer faults can significantly be improved by using the well-known state verification methods such as UIO sequences, characterization sets, or distinguishing sequences. These techniques should be applied while generating a minimum-cost test sequence from the final conflict-free graph.

The proposed solution is likely to have a broader application due to a proliferation of protocols with real-time requirements. The functional errors in such protocols are usually caused by the unsatisfiability of time constraints and (possibly conflicting) conditions involving timers; therefore, significant research is required to develop efficient algorithms for test generation for such protocols. Our methodology is expected to contribute towards achieving this goal. The preliminary results are reported in [29].

In the test cases delivered to CECOM (see Section 5), conflicting conditions based on 188-220's timers are resolved by manually expanding EFSMs based on the set of conflicting timers. This procedure results in test sequences that are far from minimum-length. The technique presented here allows us to automatically generate conflict-free test sequences for 188-220.

Suppose that a protocol specification defines a set of timers $K = \{tm_1, \ldots, tm_{|K|}\}$, such that a timer tm_j may be started and stopped by arbitrary transitions defined in the specification. Each timer tm_j can be associated with a boolean variable T_j whose value is true if tm_j is running, and false if tm_j is not running. Let ϕ be a time formula obtained from variables $T_1, \ldots, T_k$ by using logical operands $\wedge$, $\vee$, and $\neg$. Suppose that a specification contains transitions with time conditions of a form "if ϕ" for some time formula ϕ. It is clear that there may exist infeasible paths in an FSM modeling a protocol, if two or more edges in a path have inconsistent conditions. For example, for transitions e_1: if (T_j) then $\{\varphi_1\}$ and e_2: if $(\neg T_j)$ then $\{\varphi_2\}$, a path (e_1, e_2) is inconsistent unless the action of φ_1 in e_1 sets T_j to false (which happens when timer tm_j expires in transition e_1). The solution to the above problem is expected to allow generating low-cost tests free of such conflicts.

188-220's Datalink Layer Estelle specification defines several timers that can run concurrently and affect the protocol's behavior. For example, *BUSY* and *ACK* timers may be running independently in *FRAME_BUFFERED* state. If either timer is running, a buffered frame cannot be transmitted. If *ACK* timer expires while *BUSY* timer is not running, a buffered frame is retransmitted. If, however, *ACK* timer expires while *BUSY* timer is running, no output is generated. Besides Estelle specifications, feasibility constraints related to multiple concurrent timers are also of special concern for specifications in SDL.

The conflicting timers problem is a special case of the feasibility problem of test sequences, which is an open research problem for the general case [30, 72]. However, there are two simplifying features of the conflicting timers problem: (1) timer-related variables are linear, and (2) the values of time-keeping variables implicitly increase with time. Considering these features makes it possible to find an efficient solution to this special case.

3.4.1 General approach

The goal of the presented technique is to achieve the following fault coverage: *cover every feasible state transition defined in the specification at least once.* During the testing of a system with multiple timers, when a node v_p is visited, an efficient test sequence should either (1) traverse as many self-loops (i.e., transitions that start and end in the same state) as possible before a timeout or (2) leave v_p immediately through a non-timeout transition. Once the maximum allowable number of self-loops are traversed, a test sequence may leave v_p through any outgoing transition. Such an approach does not let perform full reachability analysis; however, it can be shown that considering only the above two cases is sufficient to include at least one feasible path for each transition provided such a feasible path is not prohibited by the original specification.

Suppose that there are 15 untested self-loops (each requiring 1 sec to test) in state v_{57}, and that, when the test sequence visits v_{57}, the earliest timer to expire is tm_4, with 10.5 sec remaining until its timeout. In this example, the test sequence will either leave v_{57} immediately or traverse 10 of the untested self-loops. Suppose that the latter option is chosen and, later during the test sequence traversal, v_{57} is visited again with tm_2 leaving 3.1 sec until the earliest timeout. In this case, 3 more untested self-loops of v_{57} can be covered by the test sequence. Traversal will continue until all of the v_{57}'s self-loops are tested.

In more complicated cases, in addition to the aforementioned timing constraints, traversal of a self-loop requires that its associated time condition be satisfied, i.e., certain timers be active (or, similarly, other timers be inactive). These time conditions will also be taken into account while selecting which self-loops to traverse. In the above example, if 6 or more self-loops of v_{57} have 'tm_4 not running' as their time condition, the test sequence, which tries to execute 10 of the untested self-loops, will cause a timer conflict due to the unsatisfiability of the time condition.

In general, the goal of an optimization is to generate a low-cost test sequence that follows the above guidelines, satisfies time conditions of all composite edges and is not disrupted by timeout events during traversal (i.e., contains only feasible transitions).

Similar inconsistencies, but based on arbitrary linear variables, are present in EFSMs modeling

VHDL specifications. ATIRP researchers Uyar and Duale presented algorithms for detecting [74] and removing [75] inconsistencies in VHDL specifications. Recent research in UD and CCNY focused on adapting these algorithms to detecting and removing inconsistencies caused by a protocol's conflicting timers. The software implementation of these algorithms developed within ATIRP is described in the next section.

4 Software for Automated Test Generation

The process of generating tests involved the development of two systems of software: (1) *efsm2fsm-rcpt*, and (2) INDEEL. These two systems are now described in turn.

4.1 *efsm2fsm-rcpt*

Figure 16 depicts the major software components that were developed to generate test sequences from an EFSM [28]. The software contains two packages: (1) *efsm2fsm*, and (2) *rcpt*. The former was designed and implemented at UD. The latter was based on the software written at CCNY, which originally was able to handle graphs of at most 100 transitions in a plain input/output format, without any of the additional parameters specifically required for 188-220B tests. This component was enhanced to generate tests for 188-220B for a proprietary CECOM's format. Also, the software was significantly redesigned to process large graphs (1000s of transitions), which enabled its application to more complex real-life protocols.

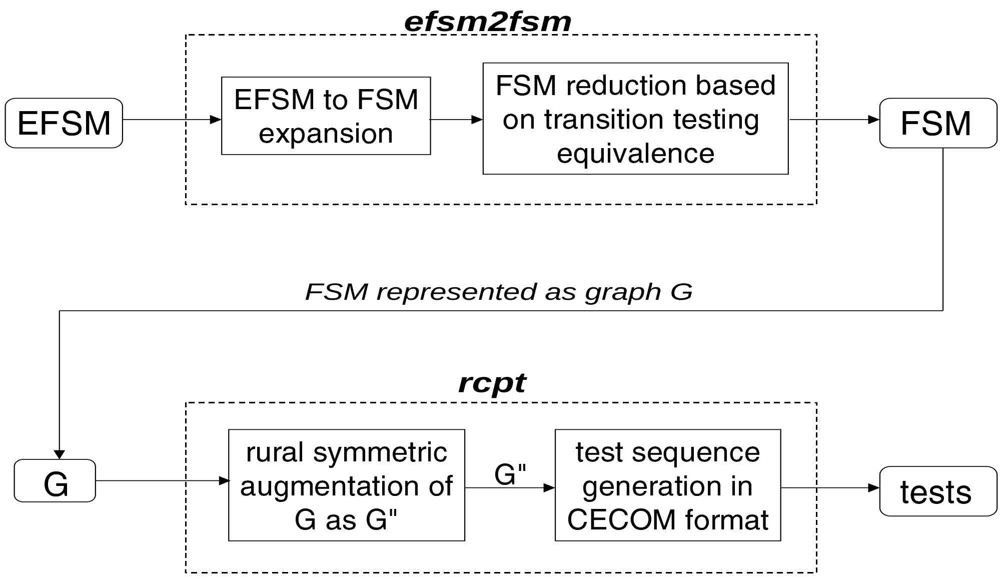

efsm2fsm
EFSM
EFSM to FSM expansion
FSM reduction based on transition testing equivalence
FSM
FSM represented as graph G
rcpt
G
rural symmetric augmentation of G as G"
G"
test sequence generation in CECOM format
tests

Figure 16: Software for automated test generation.

4.1.1 *efsm2fsm*

efsm2fsm takes a protocol's EFSM representation as input and performs its expansion to an FSM. Each EFSM's transition is associated with the following parameters: transition name in the Estelle specification, transition description, start and end states, input and output names, numerical values specifying the corresponding fields in 188-220B's PDUs, and changes in the variables' values (i.e., start and end configurations. To express the start and end configurations, a simple notation was defined. In the potential future work on this package, it is essential that this notation be replaced with a different one, which should be more expressive and flexible.

To facilitate creating the input to *efsm2fsm*, spontaneous transitions are allowed to be specified in the input EFSM. These transitions are then concatenated with regular transitions (i.e., triggered by an external input) to eliminate spontaneous transitions from the resulting FSM. This procedure can be briefly described as follows. Suppose that in a path

$$v_0 \xrightarrow{t_1} v_1 \xrightarrow{t_2} v_2 \ldots v_{i-1} \xrightarrow{t_i} v_i \ldots v_{n-1} \xrightarrow{t_n} v_n \tag{10}$$

where v_i and t_i denote a state and a transition, respectively, t_1 is regular and $t_2, \ldots, t_n$ are spontaneous. Then transitions $t_1, \ldots, t_n$ are concatenated into a single transition $t_{1,n}$ from state v_0 to state v_n. Their inputs, outputs, and other parameters are combined and associated with transition $t_{1,n}$. States $v_2, \ldots, v_{n-1}$ are marked as temporary, and subsequently removed from the FSM along with their outgoing transitions.

After the expansion to an FSM, transitions that are equivalent from a testing point of view could be identified, leading to a minimum-cost test sequence covering at least one transition from each equivalence class. However, building such a test sequence is NP-hard [28]. Therefore, simple heuristics bringing about 20%-30% reduction in the number of transitions were implemented.

It is possible to manually prepare the input file for the package such that an EFSM's states are divided into two groups: (1) states with no inputs buffered, and (2) states with one input buffered at a semicontrollable interface. Then semicontrollable interfaces can be utilized for certain simplified cases such as using the 188-220B Intranet layer for indirect testing of 188-220B Datalink layer (in these tests, only one semicontrollable interface is used with a small number of semicontrollable inputs). A self-loop repetition constraint can be taken into account for the case of self-loop state verification sequences.

To run the package for a protocol's EFSM specified in file *protocol.efsm*, the following command must be used:

```
efsm2fsm protocol.efsm [-options]
```

producing two files *protocol.fsm* and *protocol.stat*. The former contains the output FSM. All information associated with transitions in the input EFSM is preserved. This enables the *rcpt* package to populate the fields defined in the CECOM's proprietary format for test sequences. The latter file contains statistics such as the number of states and transitions in the EFSM/FSM, and the percentage effectiveness of the reduction heuristics.

Note that the original EFSM to FSM conversion technique implemented should be replaced by the application of the inconsistency elimination algorithms implemented in INDEEL (see Section 4.2). Using INDEEL to eliminate inconsistencies results in a conflict-free EFSM that is significantly smaller than the FSM.

4.1.2 *rcpt*

The FSM produced by *efsm2fsm* is then fed to *rcpt*, which builds a corresponding directed graph representation G. Then, network flow techniques are applied to find a rural symmetric augmentation of G as $G^{''}$. Finally, *rcpt* finds an Euler tour of $G^{''}$, and outputs to a file a resulting test sequence conforming to the CECOM's proprietary format.

Suppose that *protocol.fsm* is an input file containing a protocol's FSM. Then the following command runs the package:

rcpt [-cecom/-plain] protocol.fsm output_file

where *plain* option refers to a plain input/output file format. *protocol.fsm* file in plain format can be prepared manually. The *cecom* option selects test generation in the CECOM format. In this case, the input file *protocol.fsm* should be generated by the *efsm2fsm* package. The tests are stored in the number of files named *protocol.i*, where i is the index of a test group.

4.2 INDEEL: Software for Inconsistency Detection and Elimination

A software package, called INDEEL (INconsistencies DEtection and ELimination), has been implemented at CCNY based on the inconsistency elimination algorithms given in [76, 22]. As part of the ongoing collaboration between the CCNY and the UD, the application of these algorithms has been extended to generate test sequences for the protocols with conflicting timers such as 188-220.

INDEEL contains 13,000+ lines of C code. As its input, the software reads a user specified file containing the description of an EFSM graph with the following properties:

- The specification consists of a single process and thus there are no communicating EFSMs.
- If the specification contains function calls, they can be described within the process with a simple transformation.
- Pointers, recursive functions, and syntactically endless loops are assumed not to be present in the specification.
- All conditions and actions are linear.

Overall complexity of the algorithms used in INDEEL were discussed in Section 3.3.3. INDEEL uses an iterative approach: every time an action or condition inconsistency is detected and eliminated, an intermediate output graph is generated in a file, using the same format as in the input file. This intermediate output file then becomes the new input file to INDEEL for continued analysis. This iterative procedure is repeated until the graph becomes free of inconsistencies. The intermediate and the final output graphs are provided as files.

INDEEL starts its analysis by considering the action inconsistencies; it then proceeds to the detection and elimination of the condition inconsistencies (if any). During the analysis of the action inconsistencies, INDEEL constructs a set of *Action Update Matrix (AUM) pairs* for each node. The AUM pairs represent the effects of the actions of the traversed edges leading to a given node v_i. Similarly, the accumulated different conditions of the paths leading to v_i can be represented as

Table 3: 188-220 Datalink tests. A single step corresponds to one input/output exchange.

Test set	*# of states*	*# of transitions*	*# of test steps*
	Class A Type 1 service		
general behavior	298	799	1732
precedence	303	401	1316
multidestination	112	119	145
	Class C Type 1 service		
general behavior	298	799	1732
precedence	193	357	1314
multidestination	112	119	145
	Class C Type 4 service		
general behavior	235	925	2803
outstanding frames	48	172	264
multidestination	112	119	145

a set of *Accumulated Condition Matrix* (ACM) triplets containing the coefficients, operators, and constants of the edge conditions.

To reduce the space complexity, during the AUM and ACM constructions, the software uses a single matrix called $path\_matrices$ in which the numbers of the edges in the paths from the initial node to v_i are stored.

5 Technology Transfer Results

Using research results from Section 3, and software as described in Section 4.2, UD and CCNY collaborated with CECOM to generate tests for the SAP components of 188-220's Data Link Layer Classes A and C. Table 3 shows the sizes of the expanded EFSMs and the tests that were generated from them. For example, the precedence tests set for Class A–Type 1 Service was based on an expanded EFSM of 303 states and 401 transitions. The minimum-length test sequence generated for this machine consists of 1,316 input/output pairs covering every transition in the expanded EFSM at least once.

Figure 17 shows a sample of the delivered test scripts. The figure depicts the test group #92 from Datalink Class A–Type 1 service tests. Each test group is a subsequence of a full test sequence that starts and ends in the initial state. In the first step, the technique of utilizing semicontrollable interfaces presented in Section 3.2 is used. The lower tester sends a packet with three destination addresses: *IUT_addr*, *des_addr_1*, and *des_addr_2*. The setting *Relay=Yes* in the *INTRANET* clause tells the first addressee, i.e., the IUT, to relay the packet to the two remaining addressees. As a result, the IUT sends a packet with its address as a source, and *des_addr_1* and *des_addr_2* as destinations, as if it were originated by the IUT's Intranet Layer. In the second and third steps, the IUT's packet sent in the first step is acknowledged by *des_addr_2* and *des_addr_1*, respectively. Each test step is further annotated with the test description, the number of the corresponding Estelle transition(s), and the appropriate section(s) from the 188-220 official document.

The implementations of 188-220 from several manufacturers are being tested at CECOM. The tests generated by the UD and CCNY team have uncovered several implementation errors, including lack of mandatory capabilities in Datalink layer, and problems with multi-hop Intranet Relaying.

```
//              Test Group #92
// ---------------------------------------------------
    TESTGROUP=92;
    LAYER=Datalink;

// Test 1
    STIMULUS=send; // PL-Unitdata.Ind
    TIME=long;
    // DL1

    INTRANET={
      Type=IP;
      LowDelay=Yes;
      HighThroughput=No;
      HighReliability=No;
      Precedence=1; // PRIORITY
      OrgAddr=des_addr_17;
      DestRelay={
        Addr=IUT_addr;
        Distance=1;
        Des=No;
        Relay=Yes;
        Ack=No;
      };
      DestRelay={
        Addr=des_addr_1;
        Distance=2;
        Des=Yes;
        Relay=No;
        Ack=No;
      };
      DestRelay={
        Addr=des_addr_2;
        Distance=2;
        Des=Yes;
        Relay=No;
        Ack=No;
      };
    };
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=1; // PRIORITY
        IDNum=1;
        PDU=ui_0;
      };
      Command=Yes;
      SrcAddr=des_addr_17;
      DestAddr=IUT_addr;
    };

    RESULTS=receive; // PL-Unitdata.Req
    TIME=normal;
    // DL1
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=1; // PRIORITY
        IDNum=1;
        PDU=ui_1;
      };
      Command=Yes;
      SrcAddr=IUT_addr;
      DestAddr=des_addr_1;des_addr_2;
    };

  TESTDESCRIPTION={
    Intranet layer passes down a multidestination packet
    which is queued by datalink layer. Packet requires
    a coupled ack. There are no outstanding frames.
    No outstanding frame. Queued frame transmitted to multiple
    destinations. Frame requires a coupled ack. Ack timer
    started.
    };
  // ESTELLE     TYPE1SAP_3,4,TYPE1SAP_18
  // SECTION(S)  5.3.16_5.3.6.1.1_C4.3,5.3.4.2.2.2.1_5.3.6.1.1

// Test 2
    STIMULUS=send; // PL-Unitdata.Ind
    TIME=normal;
    // DL1
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=2; // ROUTINE
        IDNum=1;
        PDU=urr_0;
      };
      Command=No;
      SrcAddr=des_addr_2;
      DestAddr=IUT_addr;
    };
    RESULTS=noop; // none

    TESTDESCRIPTION={
      Second destination acks a multidestination packet.
      First has not acked yet.
     };
    // ESTELLE     TYPE1SAP_12
    // SECTION(S)  5.3.7.1.5.5_5.3.6.1.6_C4.3

// Test 3
    STIMULUS=send; // PL-Unitdata.Ind
    TIME=normal;
    // DL1
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=2; // ROUTINE
        IDNum=1;
        PDU=urr_0;
      };
      Command=No;
      SrcAddr=des_addr_1;
      DestAddr=IUT_addr;
    };
    RESULTS=noop; // none

    TESTDESCRIPTION={
      First destination acks a packet. Ack timer is stopped.
      No frame queued for transmission.
      };
    // ESTELLE     TYPE1SAP_12
    // SECTION(S)  5.3.7.1.5.5_5.3.6.1.6_C4.3
```

Figure 17: A sample of test scripts delivered to CECOM.

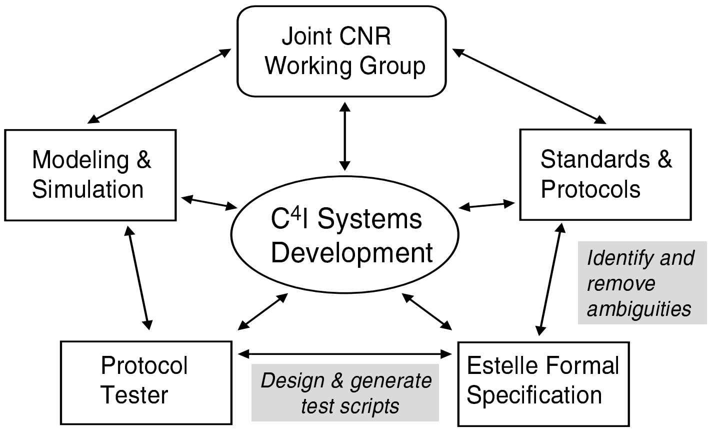


Figure 18: Estelle as part of synergistic efforts to develop C^4I systems.

6 Conclusions: Improvements to Protocol Development Process

6.1 Integration of Estelle into System Development

Traditional sequential process of system development is known to be inefficient since it allows unnecessary duplication and does not facilitate tracking of rapidly changing technology. With 188-220 as a critical component, a synergistic framework for C^4I (Command, Control, Communications, Computers, and Intelligence) systems development has been established [23] (Figure 18). It combines several parallel activities: developing protocol standards and specifications, formally specifying protocols in Estelle, building conformance tester hardware and software, "field testing", modeling and simulation, as well as resolving and documenting the solutions to standards-related technical issues by the Joint CNR Working Group. (WG participants include representatives from DoD services/agencies, industry, and academia.)

Using formal methods as part of this process helped create a high quality protocol standard, which is robust and efficient. Due to the structured nature of Estelle, the specification process progressed at an accelerated pace compared to the other standards. 188-220 was completed on time, setting a rare example in the protocol standards arena.

Since it is relatively easier to extract modeling information from a formal specification, the researchers at UD and CCNY were able to solve a number of theoretical problems, which resulted in the development of new testing methodologies. By applying these new results, the conformance tests for 188-220 were generated while the protocol was still evolving. Performing initial conformance tests on prototypes uncovered several interoperability errors early in the development process. Following this success of the 188-220 development, the synergistic efforts to develop C^4I systems with the help of formal methods serves as a model for DoD standards process and development for the future [23].

6.2 Advantages of Formal Methods in Eliminating Protocol Errors

In addition to the vagueness introduced by a natural language description, ambiguities and contradictions are difficult to detect when related protocol functionalities are defined in different document sections separated by several pages of unrelated text. Such problems are eliminated in a formal Estelle specification. All actions in a particular context are defined in one place within the Estelle specification. The specifications make the conditions for state transitions explicit through Estelle constructs. Indeed, the very process of creating these constructs enables formal specifiers to detect some of these types of ambiguities which are difficult to see in normal reading of a document written in English.

6.3 Observations on Applicability of Formal Methods

As concluding remarks for this paper, we report the following observations based on our experience during the formal specification and test generation for 188-220.

To develop an Estelle formal specification of a protocol, we must not only define its architecture and interface components (e.g., as in Figures 2 and 3 for 188-220), but we must also carefully specify the behavior of each module of these components. This definition, achieved through the creation of EFSMs, is the most difficult and time-consuming step of creating a formal specification. A syntax-directed editor improves the readability for testers who are not FDT-trained; it also is useful in writing non-trivial specifications. Moreover, the modeling and specification languages, such as SDL [33, 34] and UML [58], enjoy widespread industrial popularity, partially due to their standard graphical representation. Therefore, it will be a natural extension for Estelle to include a graphical editor [65]. Once all states and transitions of a protocol (including inputs and outputs) are finalized, the writing of the Estelle code itself is fast and straightforward.

Since 188-220 is a multilayer, multifunction protocol of a considerable size and complexity, manual generation of conformance test sequences would be both inefficient and ineffective. As seen from Table 3, the tests already delivered to CECOM contain approximately 10,000 test steps. It is clear that manually generating test sets of this size from the protocol textual description is not a trivial task.

A number of conformance test generation techniques have been proposed [2, 8, 10, 54, 62, 64, 68, 71], each of which is expected to give better results for a certain class of protocol specifications depending on the nature and size of the protocol. The experience obtained in generating tests for 188-220 suggests that to successfully test today's complex protocols by using formal methods, an ideal test generation tool should support multiple test generation techniques [49]. They can range from Postman tours [2] or fault-oriented tests [83, 85] for mid-size protocols when the number of states ranges on the order of thousands, to guided random walk approaches [47, 86] for larger protocols when the number of states ranges in the tens of thousands.

The state explosion problem has been a major issue for generating FSM models out of EFSM representations of protocols [17, 60, 84, 85]. One common procedure for converting EFSMs into FSMs simultaneously performs reachability analysis and online minimization [17, 48]; this conversion is based on combining *equivalent states* [63] using *bisimulation equivalence* [55]. Another approach proposes the elimination of inconsistencies in EFSM models [74, 75]. Efficient algorithms such as these should be implemented in any test generation tool using FSM models. If the final FSM model is not confined to a manageable size, the test sequences generated from it will be infeasibly long regardless of the test generation method.

Finally, a test house may require its own proprietary format for the executable tests. Although TTCN is accepted as input by many test tools, a proprietary test format may be preferable for a given protocol if this format is more readable by testers, or is simpler to parse by software tools. The output of a test generation tool should be easily custom-tailored for a particular format, possibly by using simple application generators.

7 Acknowledgments

The authors thank Samuel Chamberlain of ARL; Ted Dzik and Ray Menell of CECOM; and Mike McMahon and Brian Kind of ARINC, Inc. for their collaboration in this research.

References

[1] I. Adler and N. Megiddo, A Simplex Algorithm Whose Average Number of Steps is Bound Between Two Quadratic Functions of the Smaller Dimension," *Journal of the ACM*, vol. 32, No. 4, 871–895, Oct. 1985.

[2] 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 rural Chinese postman tours. *IEEE Trans. Commun.*, 39(11):1604–1615, Nov. 1991.

[3] P. D. Amer, G. Burch, A. S. Sethi, D. Zhu, T. Dzik, R. Menell, and M. McMahon. Estelle specification of MIL-STD 188-220A DLL. *Proc. IEEE MILCOM*, Oct. 1996.

[4] P. D. Amer, M. A. Fecko, A. S. Sethi, M. U. Uyar, T. J. Dzik, R. Menell, and M. McMahon. Using Estelle to evolve MIL-STD 188-220. In Budkowski et al. [13], 55–58.

[5] B. Baumgarten, H.-J. Burkhardt, and A. Giessler, editors. *Proc. IFIP Int'l Workshop Test. Communicat. Syst. (IWTCS)*, Darmstadt, Germany, Sept. 1996. Boston, MA: Kluwer Academic Publishers.

[6] A. Bhattacharyya. *Checking Experiments in Sequential Machines*. Wiley & Sons, New York, NY, 1989.

[7] J. Bi and J. Wu. Application of a TTCN-based conformance test environment to the Internet email protocol. In Kim et al. [45], 324–330.

[8] B. S. Bosik and M. U. Uyar. FSM-based formal methods in protocol conformance testing: from theory to implementation. *Comput. Networks & ISDN Syst.*, 22(1):7–34, Sept. 1991.

[9] J. Bredereke and R. Gotzhein. Specification, detection, and resolution of IN feature interactions with Estelle. *Proc. IFIP Formal Desc. Tech. (FORTE)*, 376–378. Chapman & Hall, 1995.

[10] E. Brinksma. A theory for the derivation of tests. *Proc. IFIP Protocol Specif., Test., & Verif. (PSTV)*. Amsterdam: North-Holland, 1988.

[11] S. Budkowski, A. Cavalli, and E. Najm, editors. *Proc. IFIP Joint Int'l Conf. FORTE/PSTV*, Paris, France, Nov. 1998. Boston, MA: Kluwer Academic Publishers.

[12] S. Budkowski and P. Dembinski. An introduction to Estelle: A specification language for distributed systems. *Comput. Networks & ISDN Syst.*, 14(1):3–24, 1991.

[13] S. Budkowski, S. Fischer, and R. Gotzhein, editors. *Proc. Int'l Workshop FDT Estelle*, Evry, France, Nov. 1998. Evry, France: Institut National des Télécommunications (INT).

[14] R. Burch, P. Amer, and S. Chamberlain. Performance evaluation of MIL-STD 188-220A: Interoperability standard for digital message transfer device subsystems. *Proc. IEEE MILCOM*, San Diego, CA, Nov. 1995.

[15] S. Chanson and J. Zhu. A Unified Approach to Protocol Test Sequence Generation. Proc. of *IEEE INFOCOM* 1993, pp. 1d.1.1-1d.1.9.

[16] O. Catrina, E. Lallet, and S. Budkowski. Automated implementation of the Xpress Transport Protocol (XTP) from an Estelle specification. *Electronic J. Networks & Distrib. Process.*, (7):3–19, Dec. 1998.

[17] K. T. Cheng and A. S. Krishnakumar. Automatic generation of functional vectors using the extended finite state machine model. *ACM Trans. Design Automation of Electronic Syst.*, 1(1):57–79, Jan. 1996.

[18] S.-K. Cheong, K.-H. Lee, and T.-W. Jeong. The analysis of integrating test results for ATM switching systems. In Baumgarten et al. [5], 83–89.

[19] J. Y. Choi and B. K. Hong. Generation of conformance test suites for B-ISDN signalling relevant to multi-party testing architecture. In Baumgarten et al. [5], 316–330.

[20] T. Cormen, C. Leiserson and R. Rivest. Introduction To Algorithms, McGraw Hill, 1996.

[21] DoD. *Military Standard—Interoperability Standard for Digital Message Device Subsystems (MIL-STD 188-220B)*, Jan. 1998.

[22] A. Duale and U. Uyar, Generation of feasible test sequences for EFSM models. In H. Ural, R. Probert, and G. v. Bochmann, eds, *Proc. IFIP Int'l Conf. Testing o Communicating Systems, TestCom*, Ottawa, Sept. 2000, 91-109.

[23] T. Dzik and M. McMahon. MIL-STD 188-220A evolution: A model for technical architecture standards development. *Proc. IEEE MILCOM*, Monterey, CA, Nov. 1997.

[24] M. A. Fecko, P. D. Amer, A. S. Sethi, M. U. Uyar, T. Dzik, R. Menell, and M. McMahon. Formal design and testing of MIL-STD 188-220A based on Estelle. *Proc. IEEE MILCOM*, Monterey, CA, Nov. 1997.

[25] M. A. Fecko, M. U. Uyar, P. D. Amer, and A. S. Sethi. Using semicontrollable interfaces in testing Army communications protocols: Application to MIL-STD 188-220B. *Proc. IEEE MILCOM*, Atlantic City, NJ, Oct. 1999.

[26] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer. Issues in conformance testing: Multiple semicontrollable interfaces. In Budkowski et al. [11], 111–126.

[27] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer. Conformance testing in systems with semicontrollable interfaces. *Annals of Telecommun.*, 55(1):70–83, Jan. 2000.

[28] M. A. Fecko. Timing and control issues in conformance testing of protocols PhD Dissertation, CISC Dept., Univ. of Delaware, 1999.

[29] M. A. Fecko, M. U. Uyar, A. Y. Duale, and P. D. Amer. Test generation in the presence of conflicting timers. In H. Ural, R. Probert, and G. v. Bochmann, eds, *Proc. IFIP Int'l Conf. Testing o Communicating Systems, TestCom*, Ottawa, Sept. 2000.

[30] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Trans. Software Eng.*, 17(6):591–603, Jun. 1991

[31] R. Gecse. Conformance testing methodology of Internet protocols: Internet application-layer protocol testing—HTTP. In Petrenko and Yevtushenko [59], 35–48.

[32] T. Higashino and G. Bochmann. Automatic Analysis and Test Case Derivation for a Restricted Class of LOTOS Expressions with Data Parameters. *IEEE Trans. on Software Eng.*, vol. 20, No. 1, Jan. 1994, pp. 29-42.

[33] D. Hogrefe. Validation of SDL systems. *Comput. Networks & ISDN Syst.*, 28(12), 1996.

[34] Int'l Telecomm. Union, Geneva, Switzerland. *ITU Recommendation Z100: Specification and Description Language (SDL)*, 1989.

[35] ISO, Information Processing Systems—OSI, Geneva, Switzerland. *ISO/IEC International Standard 8571-1: File Transfer, Access and Management—Part 1: General introduction*, 1988.

[36] ISO, Information Processing Systems—OSI. *ISO International Standard 9074: Estelle—A Formal Description Technique Based on an Extended State Transition Model*, 1989.

[37] ISO, Information Technology—OSI, Geneva, Switzerland. *ISO/IEC International Standard 13712-3: Remote Operations: OSI realizations—Remote Operations Service Element (ROSE) protocol specification*, 1995.

[38] ISO, Information Technology—OSI, Geneva, Switzerland. *ISO/IEC International Standard 8327-1: Connection-oriented Session Protocol—protocol specification*, 1996.

[39] ISO, Information Technology—OSI, Geneva, Switzerland. *ISO/IEC International Standard 10026-3: Distributed Transaction Processing—Part 3: Protocol specification*, 1998.

[40] ISO/IEC. *International Standard ISO/IEC 8802-2, ANSI/IEEE Std. 802.2*, 2nd edition, Dec. 1994.

[41] ITU. *Recommendation Q.2110: Service Specific Connection-Oriented Protocol (SSCOP)*.

[42] A. Jirachiefpattana and R. Lai. Uncovering ISO ROSE protocol errors using Estelle. *Comput. Stand. & Interf.*, 17(5-6):559–583, 1995.

[43] S. Kang, Y. Seo, D. Kang, M. Hong, J. Yang, I. Koh, J. Shin, S. Yoo, and M. Kim. Development and application of ATM protocol conformance test system. *Proc. IFIP Int'l Workshop Test. Communicat. Syst. (IWTCS)*, Budapest, Hungary, Sept. 1999.

[44] T. Kato, T. Ogishi, A. Idoue, and K. Suzuki. Intelligent protocol analyzer with TCP behavior emulation for interoperability testing of TCP/IP protocols. *Proc. IFIP Joint Int'l Conf. FORTE/PSTV*, 449–464, Osaka, Japan, Nov. 1997.

[45] M. Kim, S. Kang, and K. Hong, editors. *Proc. IFIP Int'l Workshop Test. Communicat. Syst. (IWTCS)*, Cheju Island, Korea, Sept. 1997. Boston, MA.

[46] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, New York, NY, 1978.

[47] D. Lee, K. K. Sabnani, D. M. Kristol, and S. Paul. Conformance testing of protocols specified as communicating FSMs—a guided random walk approach. *IEEE Trans. Commun.*, 44(5), May 1996.

[48] D. Lee and M. Yannakakis. Online minimization of transition systems. *Proc. 24th Annual ACM*, Victoria, Canada, 1992.

[49] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines—a survey. *Proc. IEEE*, 84(8):1090–1123, Aug. 1996.

[50] D. Y. Lee and J. Y. Lee. Test generation for the specification written in Estelle. *Proc. IFIP Protocol Specif., Test., & Verif. (PSTV)*, Stockholm, Sweden, June 1991.

[51] D. Y. Lee and J. Y. Lee. A well-defined Estelle specification for the automatic test generation. *IEEE Trans. Comput.*, 40(4), Apr. 1991.

[52] J. K. Lenstra and A. H. G. Rinnooy Kan. On general routing problems. *Networks*, 6:273–280, 1976.

[53] H. Li, P. Amer, and S. Chamberlain. Estelle specification of MIL-STD 188-220A: Interoperability standard for digital message transfer device subsystems. *Proc. IEEE MILCOM*, San Diego, CA, Nov. 1995.

[54] R. E. Miller and S. Paul. On the generation of minimal-length conformance tests for communication protocols. *IEEE/ACM Trans. Networking*, 2(1):116–129, Feb. 1993.

[55] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[56] P. Mondain-Monval. ISO Session Service specification in Estelle. Technical Report SEDOS Rep. 70, ESPRIT Project, Nov. 1986.

[57] C. Negulescu and E. Borcoci. SSCOP protocol throughput evaluation—simulation based on Estelle specification. In Budkowski et al. [13], 75–98.

[58] Object Management Group, Framingham, MA. *OMG Standard: Unified Modeling Language (UML) 1.1*, 1997.

[59] A. Petrenko and N. Yevtushenko, editors. *Proc. IFIP Int'l Workshop Test. Communicat. Syst. (IWTCS)*, Tomsk, Russia, Sept. 1998. Boston, MA: Kluwer Academic Publishers.

[60] D. H. Pitt and D. Freestone. The derivation of conformance tests from LOTOS specifications. *IEEE Trans. Softw. Eng.*, 16(12):1337–1343, 1990.

[61] S. Rapps and E. Weyuker. Selecting Software Test Data Using Data Flow Information. *IEEE Trans. on Software Eng.*, vol. SE-11, No. 4, Apr. 1985.

[62] J. Romijn and J. Springintveld. Exploiting symmetry in protocol testing. In Budkowski et al. [11], 337–351.

[63] K. K. Sabnani and A. T. Dahbura. A protocol test generation procedure. *Comput. Networks & ISDN Syst.*, 15:285–297, 1988.

[64] B. Sarikaya, G. von Bochmann, and E. Cerny. A test design methodology for protocol testing. *IEEE Trans. Softw. Eng.*, 13(5):518–531, May 1987.

[65] J. Templemore-Finlayson, J l. Raffy, P. Kritzinger, and S. Budkowski. A graphical representation and prototype editor for the formal description technique Estelle. In Budkowski et al. [11], 37–55.

[66] R. Tenney. A tutorial introduction to Estelle. Technical Report 88-1, Univ. of Mass, Boston, June 1988.

[67] J. Thees. Protocol implementation with Estelle—from prototypes to efficient implementations. In Budkowski et al. [13], 187–193.

[68] J. Tretmans. Conformance testing with labelled transitions systems: Implementation relations and test generation. *Comput. Networks & ISDN Syst.*, 29(1):49–79, 1996.

[69] K. Turner. *Formal Description Techniques*. North-Holland, Amsterdam, 1989.

[70] H. Ural and B. Yang. A test sequence selection method for protocols specified in Estelle. Technical Report TR-88-18, Univ. of Ottawa, June 1988.

[71] H. Ural and B. Yang. A test sequence selection method for protocol testing. *IEEE Trans. Commun.*, 39(4), 1991.

[72] H. Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311-325, Jun. 1992.

[73] M. U. Uyar and A. T. Dahbura. Optimal test sequence generation for protocols: the Chinese postman algorithm applied to Q.931. *Proc. IEEE GLOBECOM*, 68–72, Dec. 1986.

[74] M. U. Uyar and A. Y. Duale. Modeling VHDL specifications as consistent EFSMs. *Proc. IEEE MILCOM*, Monterey, CA, Nov. 1997.

[75] M. U. Uyar and A. Y. Duale. Removal of inconsistencies in VHDL specifications. *Proc. US Army Research Lab ATIRP Conf.*, College Park, MD, Feb. 1998.

[76] M. U. Uyar and A. Y. Duale. Conformance tests for Army communication protocols. *Proc. US Army Research Lab ATIRP Conf.*, College Park, MD, Mar. 2000.

[77] M. U. Uyar, M. A. Fecko, A. S. Sethi, and P. D. Amer. Minimum-cost solutions for testing protocols with timers. *Proc. IEEE Int'l Performance, Comput., & Commun. Conf. (IPCCC)*, 346–354, Phoenix, AZ, Feb. 1998.

[78] M. U. Uyar, M. A. Fecko, A. S. Sethi, and P. D. Amer. Testing protocols modeled as FSMs with timing parameters. *Comput. Networks*, 31(18):1967–1988, Sept. 1999.

[79] M. U. Uyar and M. H. Sherif. Protocol modeling for conformance testing: Case study for the ISDN LAPD protocol. *AT&T Technical J.*, 69(1), Jan. 1990.

[80] M. U. Uyar and A. Y. Duale, "Modeling VHDL Specifications as Consistent EFSMs," *Proc. IEEE MILCOM*, Monterey, CA, Oct. 1997, 740-744.

[81] M. U. Uyar and A. Y. Duale. Resolving inconsistencies in VHDL Specifications. *Proc. IEEE MILCOM*, Atlantic City, NJ, Oct. 1999, No. 5.1.3.

[82] E. Vázquez, P. Sandoval, M. Sedano, and J. Vinyes. Automatic implementation of TP4/IP with an Estelle workstation—development methodology and performance evaluation. *Proc. IFIP Protocol Specif., Test., & Verif. (PSTV)*, 125–139. Amsterdam: North-Holland, 1992.

[83] G. von Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo. Fault models in testing. *Proc. IFIP Int'l Workshop Protocol Test Syst. (IWPTS)*, 17–30. Amsterdam: North-Holland, 1992.

[84] C. J. Wang and M. T. Liu. Axiomatic test sequence generation for extended finite state machines. *Proc. 12th Conf. Distrib. Comput. Syst.*, 252–259, 1992.

[85] C. J. Wang and M. T. Liu. Generating test cases for EFSM with given fault models. *Proc. IEEE INFOCOM*, 774–781, 1993.

[86] C. West. Protocol validation by random state exploration. *Proc. IFIP Protocol Specif., Test., & Verif. (PSTV)*. Amsterdam: North-Holland, 1986.

[87] For details on Estelle specification of MIL-STD 188-220, consult www.cis.udel.edu/˜ amer/CECOM/.

[88] J. Wytrębowicz and P. Roliński. Analysis tools for Estelle specifications. In Budkowski et al. [13], 141–155.

[89] XTP Forum, Santa Barbara, CA. *Xpress Transport Protocol Specification, Rev. 4.0*, 1995.

[90] S. Yoo, L. Collica, and M. Kim. Conformance testing of ATM Adaptation Layer protocol. In Baumgarten et al. [5], 237–252.