Synchronization and Specification Issues in Protocol Testing

BEHÇET SARIKAYA, STUDENT MEMBER, IEEE, AND GREGOR V. BOCHMANN, MEMBER, IEEE

Abstract—**Protocol testing for the purpose of certifying the implementation's adherence to the protocol specification can be done with a test architecture consisting of remote tester and local responder processes generating specific input stimuli, called test sequences, and observing the output produced by the implementation under test. It is possible to adapt test sequence generation techniques for finite state machines, such as transition tour, characterization, and checking sequence methods, to generate test sequences for protocols specified as incomplete finite state machines. For certain test sequences, the tester or responder processes are forced to consider the timing of an interaction in which they have not taken part; these test sequences are called nonsynchronizable. The three test sequence generation algorithms are modified to obtain synchronizable test sequences. The checking of a given protocol for intrinsic synchronization problems is also discussed. Complexities of synchronizable test sequence generation algorithms are given and complete testing of a protocol is shown to be infeasible.**

To extend the applicability of the characterization and checking sequences, different methods are proposed to enhance the protocol specifications: special test input interactions are defined and a methodology is developed to complete the protocol specifications.

I. Introduction

PROTOCOL implementation assessment methods are used to determine that a particular implementation (in the following simply called "implementation" or "*I*") adheres to the specification of the protocol. There seems to be agreement on a general architecture to be used for testing one or more layers of the OSI protocol hierarchy [4], [10], [17]. A remote tester (also called "active tester," "tester," or "*T*" for short) and a supplementary test module (also called "test responder," "responder," or "*R*") directly connected to the implementation, and playing the role of the implementation's service user, constitute the major parts of this architecture, as shown in Fig. 1(a).

This paper addresses the problem of selecting test sequences for protocol implementation assessment. Assuming finite state machine (FSM) models for protocol specification, various methods developed for FSM's implemented in hardware and software can be applied to the selection of test sequences for protocols, as reported earlier [2]. In the context of the underlying test architecture, however, certain problems of synchronization between the tester and the responder may arise.

The paper first gives a short review of the application [20] of three finite state test sequence selection methods, (i.e., transition tours [15], characterization [7], and checking sequences [13]) to protocol implementation assessment, and then explains in Section III the nature of the possible synchronization problems. Section IV discusses algorithms for selecting test sequences without synchronization problems, which are called in the following *synchronizable*. Some protocol examples are also given for which no complete synchronizable test sequence exists. The use of synchronizable test sequences simplifies the design of the tester and the responder.

Most protocol specifications define incompletely specified machines, i.e., for certain machine state and input signal pairs there is no transition specified. Therefore, the test sequence selection methods have been generalized for the case of incomplete machines [20]. Unfortunately, these methods are not always applicable. Section V thus contains a discussion of different strategies for partially completing protocol specifications in view of making them easier to test. The X.25 virtual circuit establishment and clearing protocol is taken as an example. Also, the class 0 transport protocol is taken as the main example to demonstrate the properties of the different test sequence selection methods. A short comparison of the three methods is also given.

In Section VI, the complexities of the test sequence generation algorithms and the effect of synchronization to the complexities are discussed. Complete testing of a real protocol is shown to be infeasible.

II. Test Sequence Generation for Protocols

The testing methods explained in [7], [13], [15] are briefly explained below, using as an example the state machine specification of the ISO/CCITT class 0 transport protocol [11] shown in Fig. 2.

A. Transition Tour Method

An input sequence starting with the initial state and covering all transitions defined in the protocol specification is called a transition tour [15]. A transition tour for the transport protocol is shown in Fig. 3(a).

Formulas for the upper bound on the length of transition tours depending on the size of the specification are given in Table I. For the specification of Fig. 2, the upper bound and the actual length of the transition tour of Fig. 3(a) are listed in Table II. Table I contains upper bound formulas and their complexities for all three methods, and Table II shows actual lengths and upper bounds for this and other types of protocols.

B. The W-Method: Characterization Sequences

A *characterization set W* of an FSM *A* is a *set* of input sequences such that the output observed from the application of *W* is different for each state [7]. Every reduced, completely defined FSM possesses a *W*-set.

A *testing tree* is defined to have the machine transitions as its branches and states as its nodes; it contains each transition exactly once. The root of this tree is the initial state.

A test sequence, called a *characterization sequence*, is obtained by the concatenation of the two sets *P* and *W*, where *P* is the set containing all partial paths in the testing tree, including the empty sequence. Each sequence in the concatenation

Paper approved by the Editor for Computer Communications of the IEEE Communications Society for publication after presentation in part at the SIGCOMM '83 Symposium on Communication Architecture and Protocols, Austin, TX, March 1983. Manuscript received December 10, 1982; revised September 16, 1983. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

B. Sarikaya is with the School of Computer Science, McGill University, Montréal, P.Q., Canada.

G. v. Bochmann is with the Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, P.Q., Canada.

0090-6778/84/0400-0389$01.00

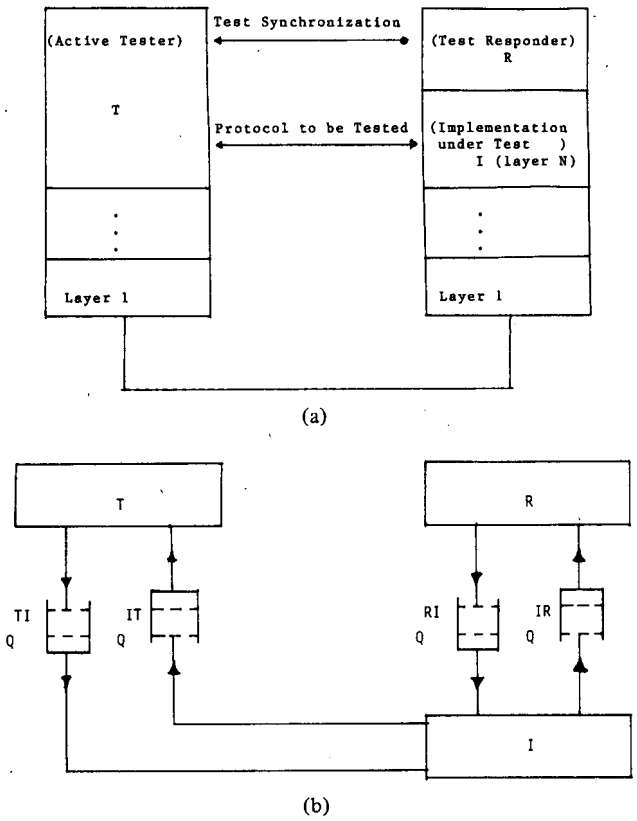
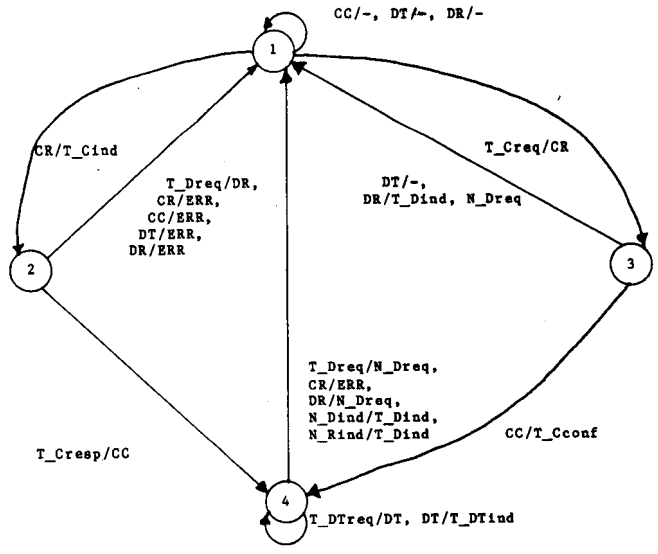


Fig. 1. (a) A test architecture for testing an (N)-layer protocol implementation in the context of the OSI reference model. (b) Basic interaction model of the test architecture in (a).



Notation for Input Events

From Tester

CR	Connect_request PDU
CC	Connect_Confirm PDU
DT	Data_Request PDU
DR	Disconnect_Request PDU
N_Dind	Network_Disconnect_Indication
N_Rind	Network_Reset_Indication

From Responder

T_Creq	T_Connect_Request
T_Dreq	T_Disconnect_Request
T_Cresp	T_Connect_Response
T_DTreq	T_Data_Request

Notation for Output Events

To Tester

ERR	Error PDU
N_Dreq	Network_Disconnect_Request
CC	Connect_Confirm PDU
DT	Data_Request PDU
DR	Disconnect_Request PDU
N_Dind	N_Disconnect_Indication
N_Rind	N_Reset_Indication

To Responder

T_Cind	T_Connect_Indication
T_Dind	T_Disc_Indication
T_Cconf	T_Connect_Confirm
T_DTind	T_Data_Indication

Fig. 2. Finite state machine for the class 0 transport protocol.

```
1 T.CR    2 R.T_Dreq  1 T.CC  1 T.DT  1 T.DR  1 T.CR     2 T.CR 1
  R.T_Cind  T.DR          -       -       -       R.T_Cind  T.ERR
 *
R.T_Creq 3 T.DT   1 T.CR    2 R.T_Cresp  4 R.T_DTreq  4 T.DT    1
T.CR       T.ERR    R.T_Cind   T.CC         T.DT          R.T_DTind

R.T_Dreq 1 T.CR    2 T.CC   1 T.CR    2 T.DT   1 T.CR    2
T.N_Dreq   R.T_Cind   T.ERR    R.T_Cind   T.ERR    R.T_Cind
           *

T.DR      1 R.T_Creq 3 T.DR            1 R.T_Creq 3 T.CC       4
T.ERR       T.CC       R.T_Dind, T.N_Dreq   T.CC       R.T_Cconf

T.CR   1 T.CR    2 R.T_Cresp  4 T.DR     1 T.CR     2 R.T_Cresp 4
T.ERR    R.T_Cind   T.CC          T.N_Dreq    R.T_Cind    T.CC

T.N_Rind 1 T.CR    2 R.T_Cresp  4 T.N_Dind  1
R.T_Dind   R.T_Cind   T.CC          R.T_Dind

Notation for Transitions:
Start State   Input Initiating Side.Input Primitive   Final State
              Output Receiving Side.Output Primitive
"*"s are used to indicate nonsynchronizable transitions.
```

(a)

```
1 R.T_Creq 3 T.CC     4 R.T_Dreq 1 T.CR    2 R.T_Dreq 1 T.CC 1 T.DT 1
  T.CR       R.T_Cconf  T.N_Dreq   R.T_Cind   T.DR         -       -

T.DR 1 T.CR    2 R.T_Cresp 4 R.T_DTreq 4 T.CR  1 T.CR    2 R.T_Cresp 4
-      R.T_Cind  T.CC         T.DT         T.ERR   R.T_Cind    T.CC

T.DT      4 T.N_Rind 1 R.T_Creq 3 T.DT 1 T.CR    2 R.T_Cresp 4
R.T_DTind   T.T_Dind   T.CR          -     R.T_Cind   T.CC

T.N_Dind 1 R.T_Creq 3 T.DR              1 T.CR    2 R.T_Cresp 4
R.T_Dind   T.CR       R.T_Dind, T.N_Dreq   R.T_Cind  T.CC

T.DR     1 T.CR    2 T.CR  1 T.CR     2 T.CC  1 T.CR     2 T.DT  1
T.N_Dreq   R.T_Cind  T.ERR   R.T_Cind   T.ERR   R.T_Cind   T.ERR

T.CR    2 T.DR  1
R.T_Cind  T.ERR
```

(b)

Fig. 3. (a) A transition tour for the transport protocol. (b) Synchronizable transition tour for the transport protocol.

TABLE I
UPPER BOUND FORMULAS AND THEIR COMPLEXITIES

Upper Bound for Transition Tour Length:	
$l(T) \le q+(q-1)(n-1)$	$O(n^2k)$
Upper Bound for W-Sequence Length(with resets):	
$l(W) \le (1/2)knw(n+1)+m(q+1)$	$O(n^4k)$
Upper Bound for Checking Sequence Length:	
$l(C) \le 2nL+(n-1)^{**}2+q(n+L)$	$O(n^{n+2}k)$

where k=maximum number of specified entries for a given state (number of input symbols at maximum)
L=length of DS
m=sum of the lengths of the members in the W-set
n=number of states
q=total number of specified entries in the Table (maximum n·k)
w=number of members in the W-set

TABLE II
ACTUAL LENGTHS (L) AND UPPER BOUNDS (U) OF TEST SEQUENCES

	Tour		W-Method		D-Method	
	L	U	L	U	L	U
Transport Protocol	34	81	65	92	64	122
X25 DTE with "Read State" transition (and "Set State 1" for W-Method)	54	273	140	216	106	380
UK Transport Protocol with "Read State" (and "Set State 1" for W-Method)	57	311	153	363	127	453
X25 DTE Semi-Completed (W-method with resets)	97	369	408	600	-	-
UK Transport Pr. Semi-Completed (W-method without resets)	337	1429	2305	4505	-	-

of P and W is applied starting with the initial state and followed by a *transfer sequence* back to the initial state (also called *reset*) to be ready for the next sequence.

As long as a W-set exists, this method is applicable to incompletely specified machines. For the transport protocol of Fig. 2, DR (the disconnect request protocol data unit) is a W-set, a single sequence of length one. A complete test sequence for the protocol is given in Fig. 4.

C. The D-Method: Checking Sequences

Checking sequences can be used to test machines that have a distinguishing sequence (DS) [13]. A checking sequence consists of two parts: first a state recognition part, and then a transition checking part. The state recognition part starting with the initial state is designed to display the response of each state to the sequence DS·DS. Transfer sequences might be used in this part when necessary. The transition checking part checks individual transitions that are not checked in the state recognition part and can be defined as

$$\mathrm{TC} = Ux_i \cdot \mathrm{DS} \qquad \text{(A)}$$

where U stands for set union and the x_i are the machine transitions to be checked.

As long as a DS exists, the D-method is applicable to incompletely specified machines. A DS for the transport protocol of Fig. 2 is again DR, which is of length one. (See also [20].)

D. Fault Detection Capability of the Methods

The transition tour method is the simplest approach, but it does not detect all errors in an implementation, i.e., errors in the next state function of the FSM may remain undetected. A characterization sequence or a checking sequence detects any misbehavior, also in the case of incompletely specified machines, as long as the method is applicable. However, this is only true as long as it can be assumed that the implementation behaves like a FSM with a number of states smaller than or equal to the specification. Unfortunately, this assumption is often difficult to check, and implementations may introduce additional states due to resource management and other practical considerations. It is therefore interesting to note that additional implementation states can be accommodated by the W-method, with, however, the penalty of increased test sequence lengths.

The fault detection is demonstrated by the following simple example. Assume that the following erroneous behavior is realized by an implementation of the transport protocol. In state 4 under input $N_D$ind, the next state is 4 (instead of 1 as indicated in Fig. 2), i.e., a transfer error. The characterization sequence of Fig. 4 applied to this implementation would lead to

1	CR	2	$T_C$resp	4	N_D ind	4	DR 1
	T_C ind		CC		T_D ind		N_D req

The unexpected output of $N_D$req to the input DR reveals the error. The transition tour of Fig. 3(a) results in

1	CR	2	T_C resp	4	N_D ind	4
	T_C inc		CC		T_D ind	

No error is detected!

III. Synchronization Problems in Test Sequences

Test sequences reported in [20] were generated with the assumption that the tester and responder are directly synchronized with one another. In the architecture of Fig. 1(a), however, the tester and responder are distributed over two sites, and they are only synchronized through the interactions with the implementation. This may lead to synchronization problems between the tester and the responder, which are explained in this section. Section IV then contains considerations for avoiding them.

```
1 T.DR  1 R.T_Creq 3 T.DR     1 T.CR    2 T.DR  1 T.CC 1 T.DR 1
    -     T.CR       R.T_Dind   R.T_Cind  T.ERR     -      -

T.DT 1 T.DR 1 T.DR 1 T.CR    2 R.T_Dreq 1 T.DR  1 R.T_Creq 3 T.CC   4
  -      -      -     R.T_Cind  T.DR         -     T.CR       R.T_Cconf

T.DR    1 R.T_Creq 3 T.DT  1 T.DR 1 R.T_Creq 3 T.DR   1 T.DR 1 T.CR 2
T.N_Dreq  T.CR       T.ERR     -    T.CR       R.T_Dind   -  R.T_Cind

R.T_Cresp 4 R.T_Dreq 1 T.DR 1 T.CR    2 R.T_Cresp 4 R.T_DTreq 4
T.CC        T.N_Dreq     -    R.T_Cind  T.CC          T.DT

T.DR     1 T.CR   2 R.T_Cresp 4 T.CR  1 T.DR 1 T.CR     2 R.T_Cresp 4
T.N_Dreq   R.T_Cind   T.CC       T.ERR     -    R.T_Cind   T.CC

T.DT      4 T.DR     1 T.CR    2 R.T_Cresp 4 T.DR     1 T.DR 1
R.T_DTind   T.N_Dreq   R.T_Cind  T.CC          T.N_Dreq    -

T.CR    2 R.T_Cresp 4 T:N_Dind 1 T.DR 1 T.CR     2 R.T_Cresp 4
R.T_Cind  T.CC        R.T_Dind     -    R.T_Cind   T.CC

T.N_Rind 1 T.DR 1 T.CR    2 R.T_Cresp 4 T.DR     1 T.CR     2 T.CR  1
R.T_Dind     -    R.T_Cind  T.CC          T.N_Dreq   R.T_Cind   T.ERR

T.DR 1 T.CR    2 T.CC  1 T.DR 1 T.CR     2 T.DT  1 T.DR 1 T.CR    2
  -    R.T_Cind  T.ERR     -    R.T_Cind   T.ERR     -    R.T_Cind

T.DR  1 T.DR 1
T.ERR     -

(Same Notation as in figure 3a)
```

Fig. 4. A characterization sequence for the transport protocol.

A. Basic Interaction Model

The implementation (I), tester (T), and responder (R), shown in Fig. 1(a), are modeled as processes, each represented as an FSM, which communicate by exchanging messages through FIFO queues [Fig. 1(b)]. T and R may send or receive a message to/from I when they execute a state transition. I can receive a message from T or R, or it can send a message to one or both of T or R after receiving a message from one of them. The system has a predefined initial state: all queues empty and all three processes in their initial states.

1) Basic Interaction Sequences (BIS): A sequence of transitions of the implementation, tester, and responder defines a transition sequence for each process. In the following, abstraction is made from the particular transitions, only the information whether an input is received (R) or an output is sent (S) is recorded, and over which FIFO queue. Following some ideas from [19], we call such an abstracted view of a transition a *basic transition*, and the individual interactions of a basic transition *basic interactions*. For example, the basic interactions corresponding to the transition from state 1 under input CR of the protocol of Fig. 2 are to receive a message through Q^{TI} from T (written R^{IT}), and to send a message through Q^{IR} to R (written S^{IR}). Hence, it corresponds to the basic transition $R^{IT}S^{IR}$. All possible basic transitions of process I under the model of Fig. 1(b) are enumerated in Table III(a), assuming that each transition of I starts with an input interaction. It is noted that some of the entries in Table III(a) (namely R^{IR}, $R^{IR}S^{IR}$, and $R^{IR}S^{IR}S^{IT}$) do not occur for the transport protocol of Fig. 2, but they may occur in other cases.

Test sequences discussed in Section II are composed of transitions of process I, each starting in the initial state. Hence, a test sequence, such as the one in Fig. 3(a), can be easily converted into a corresponding BIS by replacing each transition with its corresponding basic transition.

From a given test sequence, it is possible to obtain BISes for T and R as well. The execution steps of I that do not involve any interaction with either T or R will be shown as $\hat{}$ in the corresponding BIS. The BIS for T corresponding to a given

TABLE III
LIST OF BASIC TRANSITIONS (a) AND NONSYNCHRONIZABLE PAIRS OF TRANSITIONS (b) FOR AN IMPLEMENTATION (I)

(a)	(b)
R^{IT}	$R^{IT}R^{IR}$, $R^{IT}R^{IR}S^{IT}$, $R^{IT}R^{IR}R^{IR}$, $R^{IT}R^{IR}S^{IR}S^{IT}$
R^{IR}	$R^{IR}R^{IT}$, $R^{IR}R^{IT}S^{IT}$, $R^{IR}R^{IT}S^{IR}$, $R^{IR}R^{IT}S^{IT}S^{IR}$
$R^{IT}S^{IT}$	$R^{IT}S^{IT}R^{IR}$, $R^{IT}S^{IT}R^{IR}S^{IT}$, $R^{IT}S^{IT}R^{IR}S^{IR}$, $R^{IT}S^{IT}R^{IR}S^{IR}S^{IT}$
$R^{IT}S^{IR}$	-
$R^{IR}S^{IT}$	-
$R^{IR}S^{IR}$	$R^{IR}S^{IR}R^{IT}$, $R^{IR}S^{IR}R^{IT}S^{IT}$, $R^{IR}S^{IR}R^{IT}S^{IR}$, $R^{IR}S^{IR}R^{IT}S^{IT}S^{IR}$
$R^{IT}S^{IT}S^{IR}$	-
$R^{IR}S^{IR}S^{IT}$	-

BIS for I can be obtained from the latter by replacing an R^{IT} with S^{TI}, and an S^{IT} with R^{TI} and an execution step not involving "T" by ^. A BIS for R can be obtained in a similar way.

B. Synchronization

Definition: Considering two consecutive basic transitions of I, one of the test modules, say T (or R), faces a *synchronization problem* if T (or R) did not take part in the first transition and if the second transition requires that it sends a message to I.

Lemma: A pair of basic transitions has a synchronization problem if (and only if) the corresponding BIS for T and/or R has any sends (S) preceded by a ^.

Proof: Follows from the definition.

Two consecutive basic transitions of I will be called a *synchronizable pair of transitions* if the second transition can follow the first one without generating a synchronization problem. For example, R^{IR} followed by R^{IT} would violate synchronization because of the Lemma above. Similarly, R^{IR} and $R^{IR}S^{IR}$ cannot be followed by R^{IT} or $R^{IT}S^{IT}$ or $R^{IT}S^{IT}S^{IR}$. Also, R^{IT} and $R^{IT}S^{IT}$ cannot be followed by R^{IR} or $R^{IR}S^{IR}$ or $R^{IR}S^{IR}S^{IT}$. All nonsynchronizable pairs of basic transitions are listed in Table III(b).

Theorem: A given test sequence is synchronizable if (and only if) any two subsequent transitions of the sequence correspond to a synchronizable pair of basic transitions.

Proof:

(If part) Follows from repeated application of the Lemma to the test sequence.

(Only if part) By contradiction.

The test sequences shown in Figs. 3(a) and 4 were derived in [20] without concern for possible synchronization problems. It is easily seen that they contain synchronization problems, as indicated by "*"s. The sequence of Fig. 3(a) contains two violations, both of the type $R^{IT}S^{IT}R^{IR}S^{IT}$, which is one of the pairs listed in Table III(b). The sequence of Fig. 4 contains four violations of the same kind.

C. Protocol Specifications with Intrinsic Synchronization Problems

For certain protocol specifications, it is impossible to avoid synchronization problems. Such a situation occurs in the case that a transition p_i from state j to state k is of one of the types R^{IR}, $R^{IR}S^{IT}$, $R^{IR}S^{IR}$, or $R^{IR}S^{IR}S^{IT}$, and each transition p_i entering state j is of one of the types R^{IT} or $R^{IT}S^{IT}$. Then each pair p_ip_j is a nonsynchronizable pair of transitions. Therefore, the execution of the transition p_j implies a synchronization problem. We call such a transition *nonsynchronizable*. A dual situation exists for the case that all p_i are of types R^{IR} or $R^{IR}S^{IR}$, and p_j is of one of the types R^{IT}, $R^{IT}S^{IT}$, $R^{IT}S^{IR}$, or $R^{IT}S^{IT}S^{IR}$. We call a state *nonsynchronizable* if it can only be reached through nonsynchronizable transitions.

Protocol specifications having nonsynchronizable transitions and/or states are called *intrinsically nonsynchronizable*. It is clear that any complete test sequence generated for such specifications will carry synchronization problems. It can be seen that the transport protocol of Fig. 2 does not have any nonsynchronizable transitions; therefore, it does not have any intrinsic synchronization problem.

D. An Example: The X.25 DTE

Fig. 5 shows a state table for an X.25 DTE. Each entry in the table represents a transition to be executed by the DTE for a given present state and input interaction. An entry defines the result of the transition in the form

next state

output interaction

Fig. 5 was obtained as follows. The X.25 specification [5] defines the DCE behavior as far as DCE–DTE interactions are concerned. A symmetric behavior was assumed for the DTE, which accounts for the first four columns of rows 1–7 in Fig. 5. In addition, user interactions (see last four columns, and certain outputs) have been added to the table in a relatively straightforward way [18]. We note that state 6 handles both user clears and protocol errors, as specified in X.25. For the user errors a "user error" state (state 9) is introduced, as explained in Section V. Clear collision in the user–DTE interface is handled by adding a "clear-collision" state (state 8). Since according to X.25, the DCE does not send a clear-confirmation packet in the case of a clear collision at the DTE–DCE interface, there is no transition to state 8 in response to an interaction from the DCE.

Fig. 5 contains eight nonsynchronizable transitions: the entries in states 8 and 9 for all interactions from the DCE. The reason for this problem is that no transition from the DCE leads to either of these states. Modifications to the state table to avoid these intrinsic synchronization problems will be discussed in Section V.

IV. Generation of Synchronizable Test Sequences

Each test sequence generation method discussed in Section II may give rise, for a given protocol specification, to different test sequences depending on the way the method is implemented. It is clear from the discussion of Section III that some of these sequences are not applicable in the test architecture of Fig. 1(a) because they violate the synchronization rules. The different methods can be adopted to generate only synchronizable test sequences, unless the protocol is intrinsically nonsynchronizable. These adaptations are specific to each method, as explained below. The basic approach in all cases is to check each new transition added to the sequence in order to see whether it is synchronizable with its predecessor. This check is based on Table III(b) which lists all nonsynchronizable pairs of transitions.

A. Transition Tours

Any graph traversal algorithm such as the one given in [22] can be modified to obtain a transition tour. Each transition to be added to the sequence by the algorithm is first checked whether it forms a synchronizable pair together with the last transition of the sequence [using Table III(b)]. If it is not synchronizable, a different transition from the present state is considered. If no suitable transition exists from the present state, the selection algorithm backtracks to the previous state continuing the tour from there in a different way. This process continues until all the transitions of the machine are covered.

	Cl_ind	Cl_conf	Conn	Inc	N_Creq	N_Cresp	N_Dreq	N_Clresp
1	7 N_Dind	6 Err [Note2]	6 Err	3 N_Cind	2 Req	9 N_Err	6 Cl_req (N_Err)	9 N_Err
2	7 N_Dind	6 Err	4 N_Cconf	5 -	9(2) N_Err (-)	9 N_Err	6 Cl_req (N_Err)	9 N_Err
3	7 N_Dind	6 Err	6 Err	6 Cl_req	5 Req	4 Acc	6 Cl_req	9 N_Err
4	7 N_Dind	6 Cl_req	6 Cl_req	6 Cl_req	9 N_Err	9 N_Err	6 Cl_req	9 N_Err
5	7 N_Dind	6 Err	4 N_Cconf	6 Err	9 N_Err	9 N_Err	6 Cl_req (N_Err)	9 N_Err
6	1 N_Dconf	1 N_Dconf	6 Discard	6 Discard	9 N_Err	9 N_Err	6 Discard [Note1]	6 -
7	7 Discard	6 Err	6 Err	6 Err	9 N_Err	9 N_Err	8 -	1 Cl_conf
8	8 Discard	6 Err	6 Err	6 Err	9 N_Err	9 N_Err	8 Discard	1 Cl_conf
9	1 N_Dind	1 N_Dind	9 Discard	9 Err	9 Discard	9 -	1 Cl_req [Note1]	9 -

[Note1]:This transition may take place after time-out.
[Note2]:Specification defines state 7 as the next state for the DCE

Notation for Input Events

	From DCE		From User
Cl_ind	Clear Indication	N_Creq	N_Connect_Request
Cl_conf	Clear Confirmation	N_Cresp	N_Connect_Response
Conn	Call Connect	N_Dreq	N_Disconnect_Request
Inc	Incoming Call	N_Clresp	N_Clear_Response

Notation for Output Events

Req	Call Request	N_Cind	N_Connect_Indication
Acc	Call Accepted	N_Cconf	N_Connect_Confirmation
Cl_req	Clear Request	N_Dconf	N_Disconnect_Confirmation
Cl_conf	Clear Confirmation	N_Dind	N_Disconnect_Indication
Err	Error	N_Err	N_Error_Indication

"-" means no output is generated.

Fig. 5. State table of X.25 DTE.

In general, it may be necessary to deviate from the goal of obtaining minimum length sequences.

Applying such an algorithm to the transport protocol, the transition tour of Fig. 3(b) is obtained. The length of this sequence is 34, as in Fig. 2; in this case the length is not increased.

B. Characterization Sequences

Algorithms to find a W-set and to construct a testing tree (and, hence, to calculate $P \cdot W$, without resets) are given in [6]. Any shortest path finding algorithm, such as the one in [8], can be used for determining the resets. Synchronizable characterization sequences can be obtained in three steps as follows.

In Step 1, all subsequences of $P \cdot W$ (without resets) are checked for synchronization problems using a "subsequence checking algorithm" which checks all pairs of consecutive transitions in a sequence for synchronization problems, using Table III(b). If a subsequence of $P \cdot W$ has synchronization problems, the use of a different W set or testing tree P may be considered, possibly leading to longer sequences.

In Step 2, each subsequence of $P \cdot W$ is completed by appending a synchronizable reset sequence using a backtracking algorithm similar to the one for transition tours explained above.

In Step 3 the subsequences obtained in step 2 are merged together to obtain a single synchronizable test sequence. Any "concatenation algorithm" could be used which puts the subsequences in such an order that no synchronization problem is generated.

A synchronizable W-sequence for the transport protocol can be obtained using the same testing tree as for Fig. 4. Due to longer reset sequences, it contains four more transitions than Fig. 4.

C. Checking Sequences

Ignoring the problem of synchronization, an algorithm for finding a DS can be found in [13], and algorithms for state recognition and transition checking parts are reported in [9]. Shortest path algorithms can be used for finding transfer sequences.

The following measures are proposed to obtain synchronizable test sequences.

1) A synchronizable DS must be found, not necessarily of minimal length.

2) The state recognition part obtained according to [9] is checked using the "subsequence checking algorithm" mentioned above. In case of synchronization problems, changing the transfer sequences should first be considered. The use of a different DS may also be considered.

3) The transition checking step is checked with a two-part procedure. First each subsequence $x_i \cdot DS$ in the set TC as defined in Section II is checked by the "subsequence checking algorithm." If one of the tests fails, a different DS should be generated, if it exists. In the second step, the transition checking part as a whole is checked for synchronization. In the case of synchronization problems, a different order of the subsequences and/or different transfer sequences should be considered.

4) Finally, the state recognition and transition checking parts are combined using an appropriate transfer sequence.

A synchronizable checking sequence for the transport protocol can be obtained containing three more transitions than the sequence reported in [20].

V. Specification Enhancements for Testing

The transition tour method is generally applicable for the generation of test sequences; unfortunately, it does not have full fault detection capability. The other two methods could be applied as long as a DS or a W-set exists, which was the case for the transport protocol.

Two approaches can be taken in order to make the W- and D-methods applicable, if the original protocol specification does not have a DS and/or W-set:

1) the protocol specification may be enhanced by defining special test interactions and transitions (i.e., "read state" and "set state" transitions as described in [16]), and

2) the specification (usually incomplete) may be completed until a W-set or DS is obtained.

These two enhancement techniques will be discussed in the following sections.

A. Special Test Transitions

A "read state" transition is by definition a DS and a W-set, and a "set state" transition can be used as a transfer sequence (in particular for resets) of length one.

The advantages of using "read state" and "set state" transitions for testing can be summarized as follows.

1) It becomes possible to apply any of the test sequence generation methods discussed in Section II.

2) Minimum length test sequences are obtained in all three cases. W-set and DS are minimal (of length one) and transfer sequences for all methods have length one.

3) Incompletely specified machines can be tested verifying only the specified part. It should be noted that the special test transitions are also subject to testing. Any implementation error of these transitions will be detected by the D- or W-methods.

Test sequences for the X.25 DTE and the U.K. transport protocol [14] were obtained using the special test transitions. The lengths of these sequences are given in Table II. In obtain-

ing the upper bounds the following parameters were used:

$n = 8, q = 35, L = 1, k = 5, m = 1, w = 1$

for the X.25 DTE, and

$n = 10, q = 32, L = 1, k = 6, m = 1, w = 1$

for the U.K. transport protocol.

B. Completing Specifications

In order to make the protocol more easily testable, an incomplete specification of a protocol may be completed in the following four stages.

1) One state is added to the model, namely the "protocol and user error" state (if not already present).

2) Unspecified transitions for any input and state are specified to lead to the error state.

3) There should exist at least one transition which takes the protocol out of the error state, as, for instance, the disconnect request input.

4) The error state should ignore all other inputs and stay in the same state.

A criterion for the choice of outputs for the added transitions should be to avoid intrinsic synchronization problems. (See also examples in Section V-C.)

With the above approach, a W-set is obtained unless the resulting specification has an intrinsic synchronization problem. The existence of a DS depends on the protocol, and it might be necessary to introduce new output symbols in order to obtain a DS (see also [13]). After obtaining a W-set and/or DS as explained above, the error transitions specified in stages 2) and 4) above can be removed, if the input symbol does not occur in the W-set and/or DS.

C. Completing the X.25 Specification for Testing

The completion procedure above was applied to the X.25 DTE protocol, leading to the specification of Fig. 5. As discussed in Section III-E, the resulting table has intrinsic synchronization problems. There are two possible solutions to the problems related to state 8. The first is to completely delete this state by modifying the entry in state 7 under input $N_D$req to go to state 1 with no output. The second solution requires two modifications to the state table:

In state 6 under $Cl_$ind: next state 8, no output

In state 8 under $Cl_$conf: next state 1, output $N_D$conf

Note that this solution requires the DCE to transfer a clear confirmation packet even in the case of a clear collision, which is not foreseen in X.25. In the following discussion we assume the first solution.

No solution will be offered for the problems related to the user-error state, since these entries can be removed from the table (they are not required by the test methods considered below).

1) Output Specifications and W-Sets: As far as output in response to erroneous inputs is concerned, we adopt the proposals of [1]. As far as interactions with the user are concerned, $N_$Err indications are returned in response to all erroneous inputs from the user. With certain modifications to Fig. 5 (indicated inside parentheses), we identify the following W-sets.

$w1 = [\text{conn, inc}]$

$w2 = [N\_C\text{req}, N\_D\text{req}]$

The existence of these two W-sets, each containing only interactions from the DCE or the user, respectively, facilitates the selection of a synchronizable characterization sequence. It is possible to recognize the state of the implementation under test by applying one of these sets, depending on which side received the last output from the implementation. The synchronization problems are thus avoided.

2) Transition Tour and Characterization Sequences: A transition tour for the X.25 DTE defined in Fig. 5 was generated to cover the transitions in the state table, except the user error transitions which were considered to remain unspecified. The sequence has a length of 97, and no synchronization problems.

A characterization sequence was generated based on the two W-sets defined above, and the P-set (empty sequence ignored):

$[Cl\_\text{ind} \cdot [Cl\_\text{ind}, Cl\_\text{conf}, \text{conn}, \text{inc}, N\_C\text{req}$

$\cdot [N\_C\text{req}, N\_D\text{req}], N\_D\text{req}, N\_Cl\text{resp}],$

$Cl\_\text{conf} \cdot [Cl\_\text{ind}, \text{conn}, \text{inc}], \text{conn}, \text{inc} \cdot [Cl\_\text{ind},$

$Cl\_\text{conf}, \text{conn}, \text{inc}, N\_C\text{req} \cdot [N\_C\text{req}, N\_D\text{req}],$

$N\_C\text{resp}, N\_D\text{req}], N\_C\text{req} \cdot [Cl\_\text{ind}, Cl\_\text{conf},$

$\text{inc} \cdot [Cl\_\text{ind}, Cl\_\text{conf}, \text{conn}, \text{inc}, N\_C\text{req}, N\_D\text{req}],$

$\text{inc} \cdot [Cl\_\text{ind}, Cl\_\text{conf}, \text{conn}, \text{inc}], N\_C\text{req},$

$N\_D\text{req} \cdot [N\_C\text{req}, N\_D\text{req}, N\_Cl\text{resp}]], N\_D\text{req}]$

This sequence has a length of 408 and no synchronization problems.

The X.25 DTE state table does not possess any DS; hence, the checking sequence method is not applicable in this case.

VI. Complexity of Test Sequence Generation

The following algorithm checks for intrinsic synchronization problems. It finds all nonsynchronizable transitions and states. The algorithm determines for each state the set of test sides associated with the incoming transitions to that state, and checks if this set includes all the initiating sides of the outgoing transitions from the state. The complexity of this algorithm is $O(n^2k)$.

Ignoring the synchronization problem, the complexities of the lengths of the test sequences are shown in Table I. The complexities of the algorithms for finding these test sequences are as follows.

Transition Tour: The depth-first search algorithm [21] has the complexity of $O(n + k)$ and the breadth-first search algorithm [8] used for transfer sequence generation of $O(nk)$. Since this algorithm can be called $nk - 1$ times in the worst case, the complexity of the transition tour generation algorithm is $O(n^2k^2)$.

W-Method: Reference [7] gives the complexity of the algorithms for this method as $O(n^3k)$.

D-Method: For the complexity of an algorithm to find a DS, we assume that the most expensive operation is searching a list containing all possible state groupings (this list has a size of $2^n - 1$), whose complexity is $O(2^n)$. From [13] in the worst case, this search will be done n^n times. The algorithms to generate a D-sequence [9] are graph traversal algorithms, with a complexity of $O(n^2k^2)$; thus, the complexity of the algorithms for this method becomes $O(n^n + n^2k^2)$.

In finding synchronizable test sequences, the algorithm for intrinsic synchronization problems is a necessary first step. Hence, n^2k should be added to the complexities of the algorithms. Our practical experience with the protocols listed in Table II shows that in these cases it is possible to find synchronizable test sequences with lengths close to those which are found when the synchronization problem is ignored.

In the case of real protocols, the number of states (see Section II-D) and inputs can be very large due to the implementation considerations and to the variations in interaction parameters of input primitives. For example, a 125 byte DT

primitive of the transport protocol of Fig. 2 would introduce 2^{1000} different inputs. The above considerations therefore show that "complete testing"of a real protocol is practically impossible. Similar conclusions were reached in [16] for the complete testing of an HDLC protocol. For the application of the test methods described in this paper, it must therefore be assumed that a suitable FSM is a realistic approximation of the protocol to be tested.

VII. Conclusions

Test sequence generation methods (transition tours, *W*- and *D*-methods) are applicable to protocols specified as incomplete FSM's. The transition tour has a limited and the other methods have full fault detection capabilities. The transition tour method is generally applicable; the application of the other two methods requires the protocol possessing a *W*-set or DS, respectively.

With a remote testing architecture, as shown in Fig. 1, the synchronization between the tester and responder modules becomes an issue. It is shown in Section III that test sequences can be checked for synchronization problems by associating each interaction of a test sequence with the tester or responder module, respectively. Synchronizable test sequences can be generated using modified versions of the algorithms developed for each of the testing methods. Longer test sequences might be the price to pay. However, this is only possible if the protocol design does not include any intrinsic synchronization problems. It is not clear whether intrinsic synchronization problems can always be avoided by making appropriate changes to the protocol specification. The case of X.25 is discussed in Section V.

Two methods to enhance protocol specifications are described in Section V. These methods lead to protocol specifications that are more easily testable, especially in view of using the *W*- and *D*-methods. However, it is not clear whether such methods can be applied for the case of any particular protocol, since they lead to additions and/or changes to the protocol specification.

While this paper concentrates on exhaustive test methods and limits the discussion to protocol specifications that are given as finite state machines, most real protocols are more complex in nature. Therefore, the methods discussed here will only be applicable to a limited extent in the case of a real protocol. For instance, many protocol specifications include additional state variables and parameters for the input-output interactions [12], and typical test sequences must include means for verifying the correct behavior in relation to these interaction parameters (see, for example, [3]). Complexity considerations of Section VI (see also [16]) preclude the feasibility of exhaustive testing in these cases. Some results on test sequence selection considering interaction parameters are reported in [21]. More research is needed for a better understanding of these issues.

Acknowledgment

We are grateful to the anonymous referees and C. Sunshine for suggesting many improvements on an earlier version of this paper. We also thank E. Cerny for his helpful criticisms.

References

[1] D. Belsnes and E. Lynning, "Some problems with the X.25 packet level protocol," *Comput. Commun. Rev.*, vol. 7, no. 4, pp. 41–52, 1977.

[2] G. v. Bochmann *et al.*, "Experience with formal specifications using an extended state transition model," *IEEE Trans. Commun.*, vol. COM-30, pp. 2506–2513, Dec. 1982.

[3] ——, "Testing transport protocol implementations," in *Proc. Can. Inform. Processing Soc. Conf.*, May 1983, pp. 123–129.

[4] G. v. Bochmann and E. Cerny, "Protocol assessment," Rep. Dep. Commun. Canada, Feb. 1982.

[5] "Recommendation X.25," CCITT Study Group VII, Working Paper 2, pp. 100–190, fasc. VIII.2, Sept. 1981.

[6] H. Chaigne *et al.*, "Un generateur de tests pour systemes modelises par automates d'etats finis," BIGRE (IRISA), Rennes, France, no. 27, Dec. 1981.

[7] T. S. Chow, "Testing software design modeled by finite state machines," *IEEE Trans. Software Eng.*, vol. SE-4, no. 3, 1978.

[8] S. Even, *Graph Algorithms.* Rockville, MD: Comput. Sci. Press, 1979.

[9] G. Gonenc, "A method for the design of fault detection experiments," *IEEE Trans. Comput.*, vol. C-19, no. 6, 1970.

[10] Several papers in *Proc. 2nd Int. Workshop Protocol Specification, Testing, Verification*, 1982.

[11] ISO, "Connection oriented transport protocol specification," Apr. 1983.

[12] ISO, "A FDT based on an extended state transition model," working doc. of Subgroup B, ISO TC97/SC161 WG1, Mar. 1984.

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

[14] P. F. Linington, Ed., "A network independent transport service," Feb. 1980.

[15] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transition tours," in *Proc. IEEE Fault Tolerant Comput. Conf.*, 1981.

[16] T. F. Piatkowski, "On the feasibility of validating and testing ADCCP implementations," *NBS Trends Appl.*, May 1980.

[17] D. Rayner, "A system for testing protocol implementations," in *Proc. 2nd Int. Workshop Protocol Specification, Testing, Verification*, 1982; also *Comput. Networks*, vol. 6, Dec. 1982.

[18] R. Razouk, "Modelling X.25 using the graph model of behavior," in *Proc. 2nd Int. Workshop Protocol Specification, Testing, Verification*, 1982.

[19] R. Rubin and C. H. West, "An improved protocol validation technique," *Comput. Newtorks*, May 1982.

[20] B. Sarikaya and G. v. Bochmann, "Some experience with test sequence generation for protocols," in *Proc. 2nd Int. Workshop Protocol Specification, Testing, Verification*, 1982.

[21] B. Sarikaya, "Test design for computer network protocols," McGill Univ., Montreal, P.Q., Canada, to be published, 1984.

[22] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, 1972.

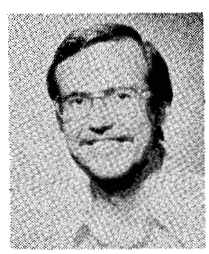
★

Behçet Sarikaya (S'80) received the B.Sc. degree in electrical engineering and the M.Sc. degree in computer science from the Middle East Technical University, Ankara, Turkey, in 1973 and 1976, respectively.

He is currently a Ph.D. candidate at the School of Computer Science, McGill University, Montréal, P.Q., Canada, and completing his work at the Département D'Informatique, Université de Montréal. His research interests include protocol specification and validation, distributed system implementation, and analytic performance modeling.

Mr. Sarikaya is a student member of the Association for Computing Machinery.

★



Gregor v. Bochmann (M'82) received the Diploma in physics from the University of Munich, Munich, Germany, in 1968, and the Ph.D. degree from McGill University, Montréal, P.Q., Canada, in 1971.

He has worked in the areas of programming languages and compiler designs, communication protocols, and software engineering. He is currently a Full Professor in the Département d'Informatique et de Recherche Operationnelle, Université de Montréal, Montréal. His present work is aimed at design models for communication protocols and distributed systems. From 1977 to 1978 he was a Visiting Professor at the Ecole Polytechnique Federale, Lausanne, Switzerland. From 1979 to 1980 he was a Visiting Professor in the Computer Systems Laboratory, Stanford University, Stanford, CA.