C. H. West
P. Zafiropulo

Automated Validation of a Communications Protocol: the CCITT X.21 Recommendation

Abstract: The call establishment procedure of the X.21 interface recommended by the International Telegraph and Telephone Consultative Committee (CCITT) has been validated as a test of a recently developed theory and of an implemented system for automated communications protocol validation. The test demonstrated the applicability of the validation technique and identified a number of points where the interface state diagram does not completely define the interface behavior.

Introduction

A recent theory due to Zafiropulo [1] has shown that it is possible to validate the syntax of a communications protocol between two processes that can be represented as a pair of directed graphs. A reformulated version of the theory has been programmed in a system that enables errors in protocols to be automatically identified [2].

The work described in this paper was undertaken in order to test the above theory and validation system in a real environment, by validating a reasonably complex protocol.

The X.21 interface has been chosen as a test case because of its current interest and also because it is formally defined in state diagram form and so can be readily validated.

In this paper we first briefly describe the X.21 interface and how its specification was interpreted for the purpose of validation. The results of the validation are then presented and their significance discussed.

X.21 interface specification

The X.21 interface is a recommendation of the International Telegraph and Telephone Consultative Committee (CCITT) for a standard means of connecting Data Terminal Equipment (DTE) to Data Circuit-termination Equipment (DCE) in a public data network [3]. In particular, it is the recommended interface for user classes of service employing a synchronous transmission mode. The interface is defined in detail in [3], which we refer to as the X.21 specification.

The X.21 specification defines all aspects of the interface, including the signaling protocol between the DCE and DTE, as well as signal formats and mechanical and electrical characteristics of the interface. Some of the above are indirectly defined by reference to other recommendations or standards.

The validation procedure we have developed has been applied only to the logical structure of the call establishment procedure for circuit switched service described in the X.21 specification. The call establishment procedure is described in [3] in four ways:

1. A text description of the states of the interface system during call establishment.
2. A state diagram specifying the signals sent between the DCE and DTE and the possible state sequences. Copies of the two sections of the state diagram are shown in Fig. 1.
3. Tables of time limits and timeouts, which give the maximum times available for the two processors to respond to each other's actions and the procedures to be adopted when no response is received.
4. Sequence diagrams that show typical timings of interactions between the DTE and DCE.

The X.21 specification clearly states that the sequence diagrams show only sample sequences derived from the state diagram and that the latter defines the logical relationships of the events at the interface.

Copyright 1978 by International Business Machines Corporation. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to *republish* other excerpts should be obtained from the Editor.

We have therefore interpreted the state diagram as being the definitive specification of the interface and the text description as providing supplementary information that helps a reader of the specification to understand the state diagram. The implication is that in the event of possible inconsistencies between the text and the state diagram, the latter will be interpreted as being definitive.

We have not addressed the problem of validating the time limits and timeouts which are defined to ensure continuing operation of the interface when either the DTE or DCE fails to respond to the other within a reasonable time. The lengths of timeouts are generally set in relation to the operating environment and the implementations for which an interface is envisaged. An evaluation of the actions taken can only be made if a detailed study of the possible causes of a response failure is made. We consider such a study to be beyond the scope of the current validation, which addresses only the operation of the interface in an error-free transmission environment. In general, the results presented below are insensitive to the specified timeout mechanisms, as the interaction sequences we have validated can all be executed without invoking timeouts.

X.21 state diagram and the options considered

The state diagram given in the X.21 specification is in two parts, as reproduced in Fig. 1.

The specification describes the state diagram as defining the logical relationship of events at the interface. Each state traversed by the interface system is represented as an ellipse, with rectangles representing related sequences of states and transitions which are not specified in detail. The top half of each ellipse and rectangle contains an identifying state name and number, and the bottom half the signals on four interchange circuits between the DTE and DCE that collectively define the state of the system. The top two signals refer to the Transmit (T) and Control (C) Interchange Circuits that carry signals from the DTE to the DCE; the bottom two, to the Receive (R) and Indication (I) Interchange Circuits carrying signals from the DCE to the DTE. Allowed transitions between states are indicated by directed links between them that are labeled either DTE or DCE to indicate the processor initiating the transition.

Either the DTE or DCE can initiate a call establishment sequence. In the case of the DTE, it first sends a CALL REQUEST, which is acknowledged by the DCE: it then indicates the call destination and facilities required by means of SELECTION SIGNALS. The DCE returns signals indicating the CALL PROGRESS and an optional CALLED LINE IDENTIFICATION, which are followed by a further exchange of signals leading to the data transfer state.

The DCE starts a call establishment procedure by signaling an INCOMING CALL, which is acknowledged. It may

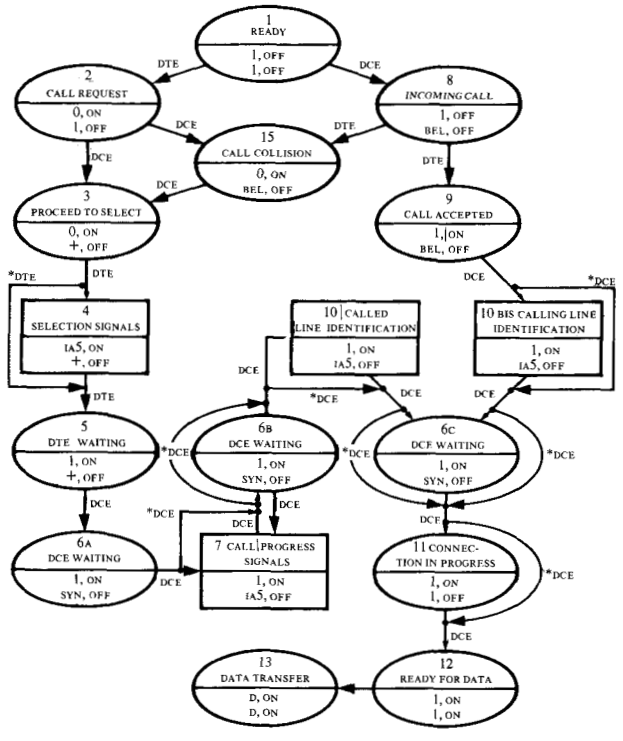


Figure 1 Two-part X.21 state diagrams from [3]. Transitions not included in the validation are indicated by asterisks. Signals are indicated as *X* when any signal may be present on the circuit.

then identify the caller. The remainder of the call establishment procedure is identical to the latter part of the one initiated by the DTE. If both the DTE and DCE simultaneously try to establish a call, a CALL COLLISION state is entered and the outgoing call is given priority.

Either processor can at any time issue a CLEAR REQUEST, which initiates a clear sequence that leads back

to the READY state. Thus, whereas there are only a few interaction sequences that lead to the DATA TRANSFER state, there are many ways in which a partially completed call establishment sequence can be terminated.

Figure 1 also shows a number of states that the DTE and DCE can enter when one of them is not ready to execute a call establishment procedure. Some of the states shown on the state diagram are described in the text as being optional, making possible a number of variations of the interface that may, for example, be defined by the network to which the equipment is attached. Rather than validate all combinations of options, we have chosen to validate a single configuration in which all of the optional states are included. We have thus studied an interface in which the states numbered 4, 7, 6B, 10, 6C, and 11 are traversed by sequences starting with the DTE initiating a CALL REQUEST, and the states 10bis, 6C, and 11 are traversed when the DCE signals an INCOMING CALL. To ensure these conditions, all links bypassing these states have been suppressed. Such links are indicated by asterisks preceding their labels in Fig. 1.

The optional bypassing of the DCE CLEAR CONFIRMATION state 17 has been similarly suppressed.

Two minor points about our interpretation should be mentioned for completeness. The condition of the R circuit in state 14 (DTE CONTROLLED NOT READY) is noted as being for further study. We have for present purposes defined it as being 1 because any other assignment would require the insertion of an intermediate state between the states READY and DTE CONTROLLED NOT READY.

The specification also states that the DCE CLEAR INDICATION and DTE CLEAR REQUEST states may be entered from all states. We have interpreted this as meaning all states during call establishment, but not including the READY state itself. This was the interpretation given in a recent paper on the X.21 interface [4]. Allowing these states to be entered from the READY state would render ambiguous a signaled transition into a NOT READY state, which is obviously not the intention of the interface designers.

Apart from the options considered and the two minor points of interpretation, we believe that we have validated the interface as it is specified.

Problems of interpretation of the state diagram

The X.21 interface specification gives the possible sequences of events in the X.21 interface in the form of a state diagram for the combined DTE-DCE system. The protocol validation procedure described in [1, 2] can only be applied to a pair of separately defined communicating processors.

To validate the X.21 interface, we have therefore had to go through a step that every designer implementing the interface must also go through, namely, one of deriving the logical structure of the DCE and DTE that are to communicate according to the state diagram in the X.21 specification.

Whereas the existence of the X.21 specification in the form of a combined state diagram implies that this can be done, it does not define an algorithm for doing it.

This raises a number of questions that should be considered. First, is there a unique, correct algorithm by means of which the logical structure of the DCE and DTE can be derived from the state diagram? If this is not the case, it is possible that different pairs of processors can be derived that can execute the X.21 interface specification.

In the latter case, suppose that two algorithms A and B exist that can each be used to derive a DTE/DCE pair. It is not immediately obvious that the DCE derived from algorithm A will operate correctly with the DTE derived from algorithm B. If it does not, it is conceivable that equipment designed by different manufacturers as implementations of the same standard may not be compatible. We know of no theory which proves that the combined state diagram used in the X.21 specification permits a unique derivation of the logical structure of the DTE and DCE. There are, however, a number of conditions that must be satisfied for such a derivation to be possible.

The first is that the combined state diagram must define the complete interaction domain of a DTE and DCE. Suppose that no provision had been made in the combined state diagram for a possible call collision. System behavior would then not be defined if a call collision occurred, and it would not be possible to uniquely define processors that could handle a call collision.

Second, any DTE or DCE derived from the combined state diagram should be capable of executing the complete execution domain of the state diagram. Otherwise one might initiate an interaction sequence that might not be executable by the other.

The first condition can be partially verified by a process of validation. The second can be verified by comparing the possible interaction sequences of each processor with those of the X.21 state diagram.

Method used to derive the DTE and DCE processors

In this section we discuss how we have derived the logical structure of the DTE and DCE processors from the state diagram. The derivation is reasonably simple, but we have no proof that it represents a unique interpretation of the specification.

Inspection shows that all transitions in the state diagram represent either a DTE- or DCE-initiated transition. This is indicated directly by the label of the link between each pair of states and may also be inferred from changes in the signals in the interchange circuits when the transition is executed.

All transitions result in changes of either the T and/or C circuits or of the R and/or I circuits, but not simultaneously of both pairs. There are thus two possible transition types between an arbitrary state pair A and B that can be characterized as in Fig. 2. The signals on the T, C, R, and I circuits in state A are shown as t(A), c(A), r(A), and i(A), respectively. If the transition to state B is DCE-initiated, the T and C circuits contain the same signals in state B, namely t(A) and c(A). Either or both of the R and I circuits are redefined to new values, which is indicated by relabeling the R and I signals in state B as r(B) and i(B).

Similarly, in a DTE-initiated transition, the R and I circuits are unchanged, but either or both of the signals on the T and C circuits are redefined in state B to be t(B) and c(B).

We have therefore modeled all transitions as being associated with an information exchange between the two processes. This is represented as in [1, 2] by the transmission of an indivisible unit of information or event from one process to the other. We define an event as being the new signals on the T and C circuits [t(B), c(B)] in the case of a DTE-initiated transition and those on the R and I circuits [r(B), i(B)] in the case of a DCE-initiated transition.

We then assume that a transition in the X.21 state diagram represents a related pair of transitions in the DTE and DCE. In one processor there is a transition associated with the transmission of an event; in the other processor, a transition resulting from the reception of the same event.

Figure 3 shows how the two types of transitions, DTE-initiated and DCE-initiated, are modeled in the two processors. In both cases the initiating processor transmits the event, the other receives it. If this transformation is applied to each transition in the state diagram, state diagrams for both the DTE and DCE can be derived, as shown in Fig. 4. Both have the same topology and number of states as the original diagram. Transitions between corresponding pairs of states in the two diagrams indicate changes in either the T and/or C circuits or the R and/or I circuits and are symmetric in the sense that they correspond to transmission and reception of the same event.

It should be noted that the final transition in the X.21 state diagram, i.e., to the DATA TRANSFER state, has been excluded. This is a transition that is not initiated by one of the processors in the same way as all others in the X.21 state diagram. It can occur as soon as the READY FOR DATA state is reached and is not considered to be part of the call establishment procedure.

The combined state diagram representation used in the X.21 specification appears to have the property that the structure of a pair of derived DTE and DCE state diagrams must have the same topology and the same send/receive symmetry that we have noted above. It is therefore only

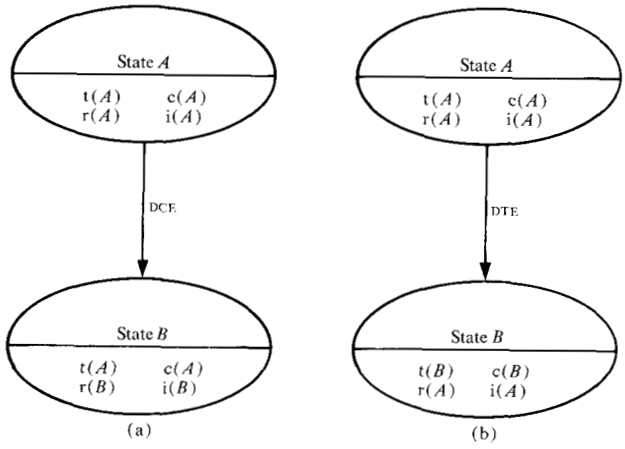
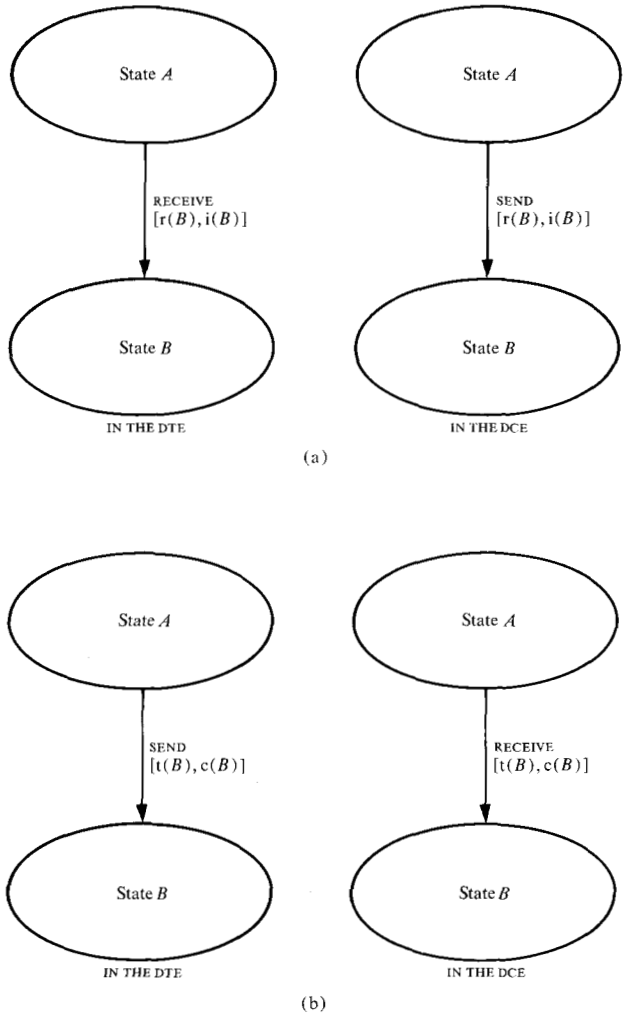


Figure 2 Characteristic DCE-initiated (a) and DTE-initiated (b) transitions in the X.21 state diagram.

Figure 3 Transformation of a DCE-initiated (a) and a DTE-initiated (b) transition.



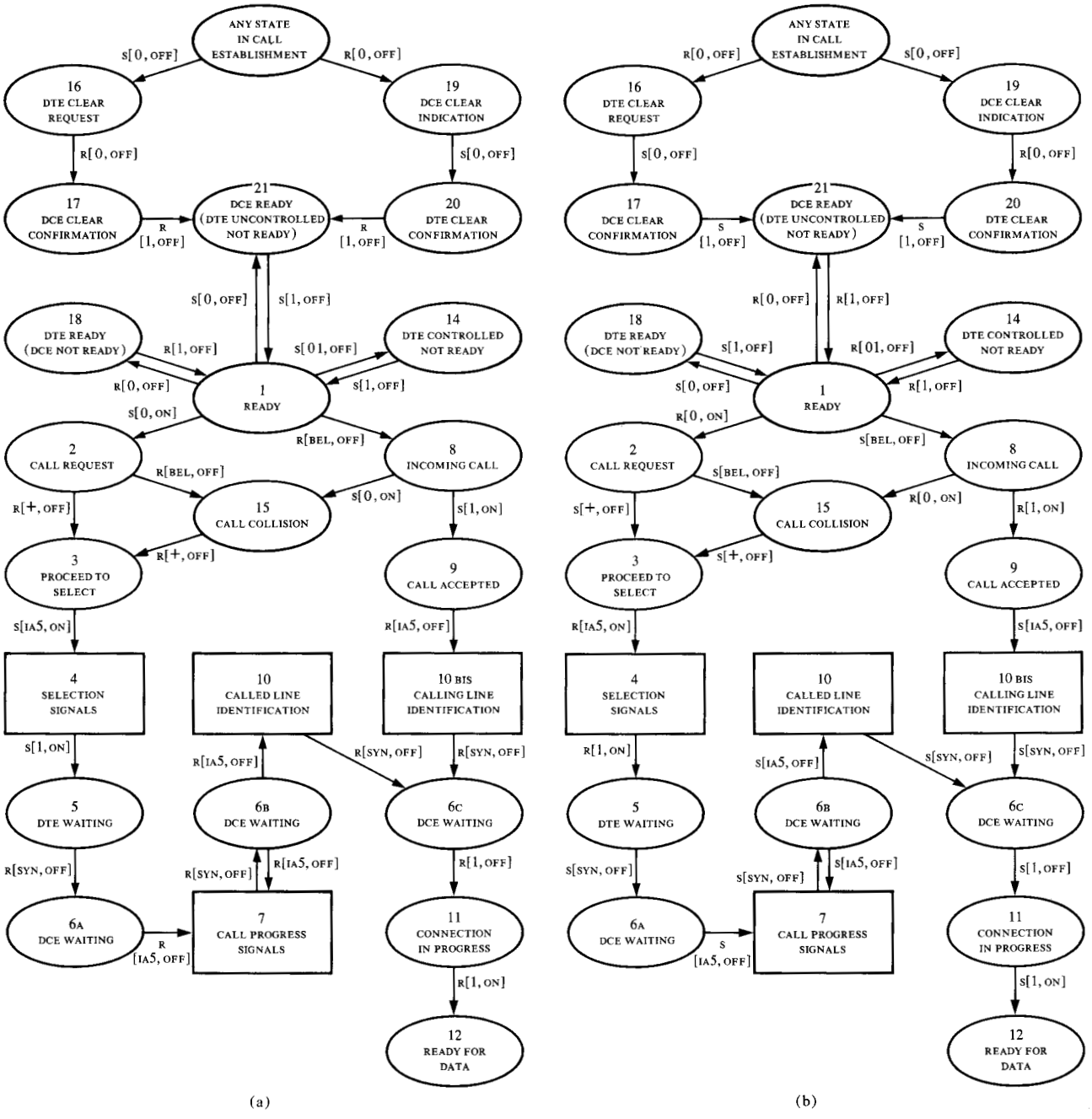


Figure 4 Derived state diagrams for the DTE (a) and DCE (b), where S indicates send and R receive.

suitable for the representation of a restricted class of interacting processors. One implication of this is discussed in a subsequent section.

The DTE and DCE state diagrams, derived as described above and shown in Fig. 4, are in a form suitable for application of the validation procedure described in [1, 2]. It is merely necessary to identify each distinct event exchanged across the interface with an integer and to label transitions in the state diagrams with positive or negative integers, indicating that a given event is received or transmitted, as described in [1].

Outline of the validation procedure

The validation procedure applied to the DTE and DCE state diagrams derived above is discussed in detail in [1, 2] and so is only briefly reviewed here.

Each state diagram is expressed in terms of a transition matrix whose elements contain the labels of the transitions between states. Paths consisting of a sequence of transitions in the state diagram are derived by iteratively multiplying the transition matrix by itself in the set theoretic sense. When the transition matrix is raised to the power N, its elements contain all paths of length N in the state diagram. After each iteration, paths that start at and return to the initial state (in this case the READY state) are saved. In this way all possible paths or interaction sequences that both processors can execute can be determined.

We refer to an individual interaction sequence starting from the initial state as a unilogue. The Cartesian product of the set of unilogues derived for the DTE with that derived for the DCE is a set of duologues, or pairs of potential interaction sequences that collectively represent the total interaction domain of the two processors. Each duologue is individually tested to determine whether it can be correctly executed, is nonoccurrable, or is erroneous.

A duologue is correctly executable if an attempt to execute it results in both processors returning to the initial state having correctly received all events transmitted by the other.

A duologue is nonoccurrable if its attempted execution invariably results in the execution of some other duologue being completed.

A duologue is erroneous if its attempted execution brings the system into a deadlock condition or results in an event being transmitted by one processor that cannot be subsequently received by the other.

The validation procedure validates the "syntax" of the interaction, whether or not the exchange of messages always takes place in a predefined and predictable way. It does not address the problem of the validity of the "semantics" of the interaction, namely, whether or not the executable interactions accomplish a meaningful exchange of control information or data.

Determination of the set of duologues

The validation procedure described in [1, 2] can be applied to protocols between processors that return to an initial state after a finite number of event exchanges. The procedure is thus limited in its application to protocols that do not contain loops, i.e., cycles of state transition sequences that can be repeatedly executed without the processes traversing their initial states.

It has been assumed in the validation procedure for the DTE and DCE that each starts in the READY state. Several other states, such as DTE CONTROLLED NOT READY or DTE READY (DCE NOT READY), could also have been chosen as the initial state. Other choices would have had the disadvantage of increasing the number of loops not traversing the initial state.

With the choice of the READY state as the initial state, there is one loop in each processor that makes the potential interaction domain infinite. This is the loop between CALL PROGRESS SIGNALS (state 7) and DCE WAITING (state 6B) that enables multiple CALL PROGRESS blocks to be transmitted. In any practical implementation, the number of CALL PROGRESS blocks transmitted will be finite, so that we can consider a finite interaction domain containing all interaction sequences starting with a CALL REQUEST that contains any number of CALL PROGRESS blocks up to a predetermined maximum.

In the validation we have performed, we have arbitrarily set this number as three, by not considering unilogues that contain more than three traversals of any individual state.

With this limitation, a total of 153 unilogues was derived by multiplication of the transition matrix for each processor, so that there was a total of 23 409 duologues to be validated.

This number was considerably higher than we had expected. It was found to be a result of a property of the X.21 state diagram that we had not foreseen and is a result of the way a call collision is represented.

The text description of the CALL COLLISION state in the X.21 specification is as follows:

"A CALL COLLISION is detected by a DTE when it receives INCOMING CALL in response to CALL REQUEST. It is detected by a DCE when it receives CALL REQUEST in response to INCOMING CALL."

Our interpretation of the above was that the statements represented the only circumstances in which the CALL COLLISION state was entered, although this is not explicitly stated.

The paths in the derived state diagrams that correspond to the above interpretation are shown in Fig. 5.

An examination of the generated sets of unilogues showed that the CALL COLLISION state could be entered in other circumstances. Figure 6 shows two alternative paths that can also lead to the CALL COLLISION state. These correspond to the DCE signaling INCOMING CALL after it has received CALL REQUEST and the DTE initiating a CALL REQUEST after it has received an INCOMING CALL signal. It is not clear that these two alternative ways of entering the CALL COLLISION state are intended to be available in the X.21 interface. They can be excluded by removing the transition from CALL REQUEST to CALL COLLISION in the DCE state diagram and that from INCOMING CALL to CALL COLLISION in the DTE state diagram. However, the resulting state diagrams are then topologically different and the interface can no longer be represented by a combined state diagram of the form given in the X.21 specification.

As we have interpreted the state diagram in the X.21 specification as being definitive, we have left these transi-

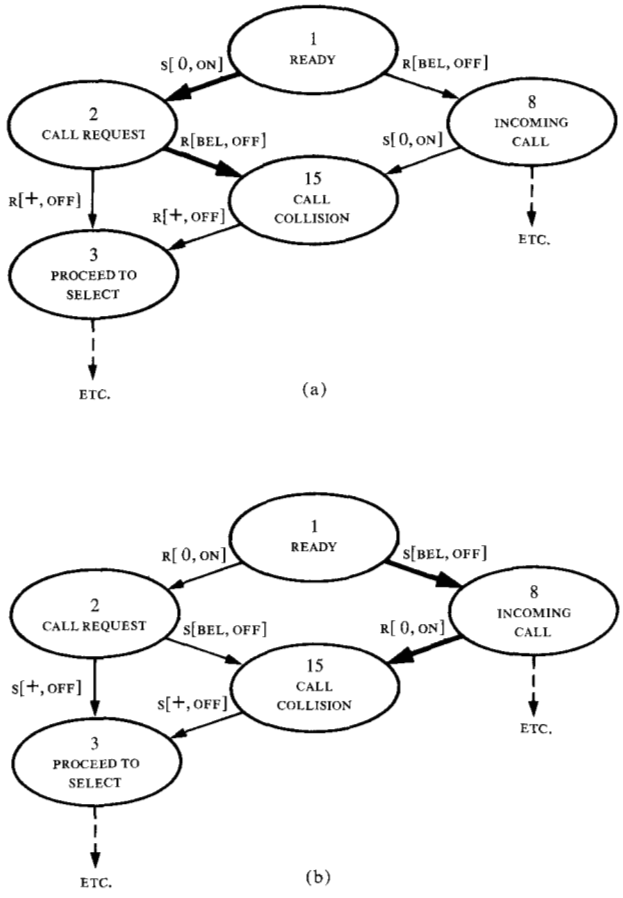
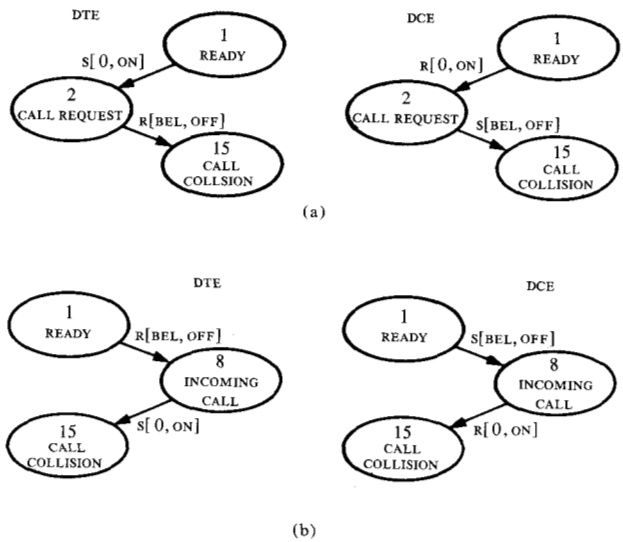


Figure 5 Call collision implied by text. DTE state diagram is at (a) and DCE state diagram at (b).

Figure 6 Alternative ways of entering the CALL COLLISION state. At (a) DCE signals INCOMING CALL after receiving CALL REQUEST; at (b) DTE signals CALL REQUEST after receiving INCOMING CALL.



tions in the state diagrams we have validated. They do not influence the results of our validation, but their inclusion modifies the way in which the interface operates.

Validation results

The 23 409 duologues have been validated according to the procedure described in [1, 2]. A total of 29 error conditions was indicated by the programs. All were examples of one processor being in a state such that no means of accepting an event transmitted by the other was defined. We use the term "error" to describe such a condition, but in applying it to the X.21 specification as it exists at present, it should be remembered that some features of the X.21 interface are under study, and the "errors" we have found can be interpreted as indicating areas of the interface specification that are incompletely defined.

The 29 errors detected can be divided into three classes.

The first class consists of eight errors that are the result of collisions resulting when either the DCE or DTE indicates a transition to a NOT READY state at the same time that the other is initiating a call establishment procedure or is itself making a transition to a NOT READY state.

The second class consists of ten errors which are probably less significant, being the result of collisions that can occur when a call establishment procedure is cleared by the DTE or DCE when a response from the other is outstanding.

The remaining 11 may be classified as miscellaneous. Some are secondary errors that may occur in interaction sequences after one of the collisions described above has occurred. Others are conditions that have been flagged as errors as a result of the way the validation procedure has been defined, and inspection shows that they do not represent actual errors in the interface.

These three classes of errors are discussed in more detail in the following sections.

With some justification, it may be argued that inclusion of the timeouts in the validation would permit most of the collisions identified by the validation to be resolved. In order to do this, it is necessary to make assumptions concerning the behavior of a processor after it has received an event not explicitly provided for in the state diagram and in circumstances which are possibly unforeseen by an implementer. Rather than make specific assumptions, we consider the behavior of a processor in such circumstances to be undefined. Readers who are particularly interested in the validation results as they pertain to the X.21 interface should bear this in mind.

- *Collisions due to transitions to NOT READY states*

The X.21 interface specification recognizes that collisions are possible when both the DTE and DCE independently initiate a call establishment procedure. In this case the

Table 1 Error conditions due to NOT READY transitions.

State of DTE when error occurs	*Event received* R circuit	I circuit	*Indicated DCE state*
DCE READY (DTE UNCONTROLLED NOT READY)	BEL	OFF	INCOMING CALL
DTE CONTROLLED NOT READY	BEL	OFF	INCOMING CALL
DCE READY (DTE UNCONTROLLED NOT READY)	0	OFF	DTE READY (DCE NOT READY)
DTE CONTROLLED NOT READY	0	OFF	DTE READY (DCE NOT READY)
State of DCE when error occurs	*Event received* T circuit	C circuit	*Indicated DTE state*
INCOMING CALL	01	OFF	DTE CONTROLLED NOT READY
DTE READY (DCE NOT READY)	0	ON	CALL REQUEST DCE READY (DTE
DTE READY (DCE NOT READY)	0	OFF	UNCONTROLLED NOT READY)
DTE READY (DCE NOT READY)	01	OFF	DTE CONTROLLED NOT READY

DTE is given priority when a transition to the CALL COLLISION state is made, and the DCE cancels the INCOMING CALL. There is no indication in the specification of how other possible collisions are to be handled. These can occur when the DTE or DCE attempts to establish a call while the other is making a transition into a NOT READY state or when both simultaneously go into a NOT READY state.

A summary of the error conditions of this type that the validation programs indicated is given in Table 1. This shows for each processor the state it is in when the error occurs, the signals on the incoming circuits for which no transition from the current state is defined, and the state of the other processor that these signals indicate.

In discussing the collisions that result in the errors listed in Table 1, we concentrate on the one generated when the DCE indicates INCOMING CALL while the DTE is making a transition to DTE CONTROLLED NOT READY. This is potentially an important case, as the CONTROLLED NOT READY state is available to indicate to the network that the terminal is operating off-line. Depending on the usage patterns of a particular terminal, there may be a greater probability of this collision occurring than there is of a CALL COLLISION. The validation procedure indicated that there is no provision for [BEL, OFF] being received by the DTE when it is in the state DCE READY (DTE CONTROLLED NOT READY) and for [01, OFF] being received by the DCE in state INCOMING CALL. This situation arises when the two processors simultaneously make transitions from the READY state into the above states.

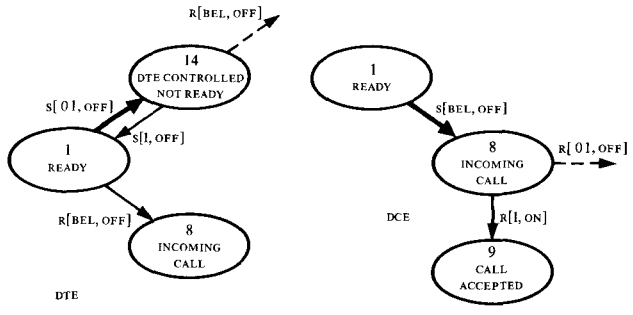


Figure 7 A collision generated by a NOT READY transition.

Figure 7 shows this in detail. Part of the state diagrams for both processors showing the transitions in question are shown. The transitions that lead to the collision are shown by broad arrows; needed departing links from the states in which the errors occur that would resolve the errors are indicated by dashed arrows.

We have carefully studied this example to convince ourselves that the errors were due to incomplete specification of the interface and were not a result of a misinterpretation of the specification or an erroneous validation procedure.

A number of points discussed in the specification should be mentioned. The CONTROLLED NOT READY state is still a subject of further study by the CCITT; in later versions it may be possible to enter it from other states, and a family of CONTROLLED NOT READY signals may be defined. Neither of these alternatives would directly re-

solve the collision. A second point is that there is a timeout after the DCE has indicated INCOMING CALL that allows it to return to the READY state if the DTE does not signal CALL ACCEPTED within a specified time. This timeout is obviously disabled if a CALL COLLISION is detected, but this is not stated explicitly in the specification. The collision in question produces a situation similar to that in which the CALL COLLISION state is entered. The DTE appears to respond to INCOMING CALL with a signal other than the expected CALL ACCEPTED, and the specification does not define whether or not the timeout remains activated in these circumstances.

We have investigated the possibility that the collision manifests itself as a result of the way we have performed the validation. To do this we considered two ways of implementing the processors.

The first was a modular design in which each processor contained an input decoder, whose function was to monitor the signals on the input circuits, assemble characters and messages, and indicate to a decision making unit when a complete input signal or message had been received. We concluded that the collision would manifest itself in the decision making unit in the same way as it occurs in the state diagrams, and we could see no way of designing modular processors that would avoid it.

The second implementation was one in which the input circuits were monitored directly by decision making logic on a bit by bit basis. In this implementation, each serial bit of a character would produce an intermediate state change in a processor. We concluded that the collision would still occur, but now in each of the intermediate states.

We therefore concluded that the collision described above can occur and that the X.21 specification as it currently exists contains no mechanism for resolving it.

Similar collisions occur when either the DTE enters DTE UNCONTROLLED NOT READY or when the DCE enters DCE NOT READY, while the other is initiating a call establishment sequence. These collisions are sufficiently similar to the one discussed that we will not describe the results in detail. It should be noted that the possibility of a collision between DTE READY (DCE NOT READY) and CALL REQUEST is mentioned in [3] during the discussion of test loop activation. The collisions that occur when both processors indicate NOT READY are also similar, but here it should be noted that the possibility of both processors indicating NOT READY is not discussed at all in the X.21 specification, although such states are obviously possible.

We have made no serious attempt to extend the combined state diagram in the X.21 specification to resolve the above collisions. Our brief investigations indicate that it is difficult to represent such extensions in terms of a combined state diagram without introducing undesirable transitions. This appears to be another manifestation of the problems of a single state diagram representation as discussed with respect to the CALL COLLISION state in an earlier section.

- *Collisions during clearing*

Either the DTE or DCE can initiate a CLEAR REQUEST at any time during a call establishment operation.

They do this by indicating [0, OFF], which initiates a return to READY via a number of intermediate states.

When [0, OFF] is indicated, each processor waits for the same acknowledging [0, OFF] response from the other, which must arrive within a specified time. The validation procedure indicated that if the initial CLEAR REQUEST were indicated when a response was expected as part of the normal call establishment sequence, this could be received before the acknowledging [0, OFF] in either the DCE CLEAR INDICATION state of the DCE or the DTE CLEAR REQUEST state of the DTE. The state diagrams do not obviously indicate that this can occur. Figure 8 shows one example of this type of collision. The DTE has initiated a call establishment procedure and followed the CALL REQUEST sequence as far as state DTE WAITING (state 5). While it is doing so, the normal signal exchange has resulted in the DCE reaching the same state. The collision occurs when the DTE at this point issues a CLEAR REQUEST and goes to state 16 while at the same time the DCE makes a transition to DCE WAITING (state 6A) by sending [SYN, OFF]. When the DCE receives the CLEAR REQUEST, it makes the transition to the DTE CLEAR REQUEST state also, but the DTE will receive the [SYN, OFF] signal in state 16 before the acknowledging [0, OFF]. The X.21 state diagram shown in Fig. 1 shows that the R and I circuits may contain any signals while the system is in the DTE CLEAR REQUEST state, and this may be interpreted as implying that the collision indicated is covered by the X.21 specification. However, it is perhaps not obvious to a designer that the R and I circuits in this state may not only indicate several different signals but may also change. A design that fails to take account of this type of collision may result in a serious problem when it occurs.

Table 2 shows all of the signals that the processors can receive when a clear sequence has been initiated; one is by no means obvious. This is the receipt of [01, OFF] by the DCE in DCE CLEAR INDICATION, which can occur when the DCE signals [BEL, OFF] to indicate INCOMING CALL then immediately makes a CLEAR REQUEST. At the same time the DTE goes into a CONTROLLED NOT READY state by signaling [01, OFF], which is first detected by the DCE when it reaches DCE CLEAR INDICATION. Such an interaction sequence may not be possible in any given implementation because of timing constraints, but it illustrates the exhaustive nature of the validation procedure we are using.

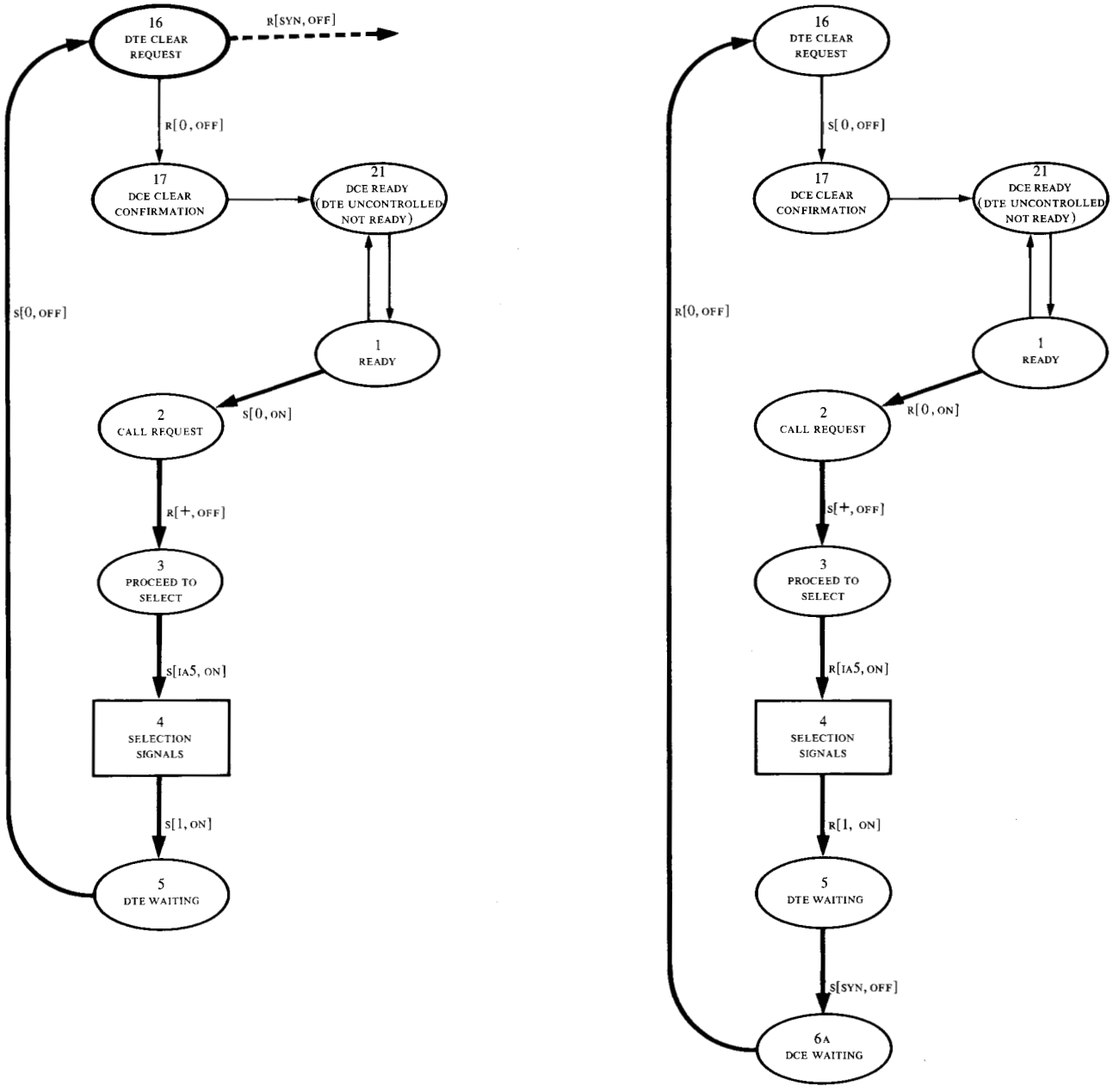


Figure 8 A collision generated during a clear sequence.

Note that the reception of [1, OFF] by the DTE in the DTE CLEAR REQUEST state appears as an error as we omitted the optional transition directly from DTE CLEAR REQUEST to DCE READY (DTE UNCONTROLLED NOT READY) from our validation. However, the inclusion of this transition makes it possible for the [1, OFF] sent to indicate CONNECTION IN PROGRESS to be interpreted as DCE READY, which can lead to further errors if the DCE replies to DTE CLEAR REQUEST with DCE CLEAR CONFIRMATION.

- *Other indicated errors*

The other 11 errors indicated by the validation procedure are not very significant but are briefly mentioned for completeness. Most of them were indicated on interaction sequences that started with a collision and for which an error listed in Table 1 was also indicated. An extension of the interface that resolved the collision problems discussed in the previous section would also resolve these secondary errors.

Table 2 Signals received as a result of clear collisions.

State of DTE when error occurs	*Event received* R circuit	I circuit	*Indicated DCE state*
DTE CLEAR REQUEST	BEL	OFF	INCOMING CALL
DTE CLEAR REQUEST	+	OFF	PROCEED TO SELECT
DTE CLEAR REQUEST	SYN	OFF	DCE WAITING
DTE CLEAR REQUEST	IA5	OFF	CALL PROGRESS SIGNALS or LINE IDENTIFICATION
DTE CLEAR REQUEST	1	OFF	CONNECTION IN PROGRESS
DTE CLEAR REQUEST	1	ON	READY FOR DATA
State of DCE when error occurs	*Event received* C circuit	T circuit	*Indicated DTE state*
DCE CLEAR INDICATION	01	OFF	DTE CONTROLLED NOT READY
DCE CLEAR INDICATION	0	ON	CALL REQUEST
DCE CLEAR INDICATION	1	ON	READY FOR DATA
DCE CLEAR INDICATION	IA5	ON	SELECTION SIGNALS

Two of these should, however, be discussed in more detail. Referring to the state diagrams in Fig. 4, it can be seen that a transition of the DCE to INCOMING CALL can occur in collision with a transition of the DTE to UNCONTROLLED NOT READY, as has been discussed in a previous section. No error as a result of the collision is immediately found in the DCE as the incoming signal for DTE UNCONTROLLED NOT READY is the same as that for DTE CLEAR REQUEST, which can be accepted in INCOMING CALL.

Thus, whereas no error is immediately found in the DCE as a result of the collision, a situation leading to an error develops as the DCE assumes that the DTE has received INCOMING CALL and has immediately initiated a clear sequence. The DCE responds to [0, OFF] accordingly with the result that an error is flagged in state DCE CLEAR CONFIRMATION, as the DCE can here detect a return of the DTE to the READY state.

A symmetric error was flagged in the DTE as a result of a collision between CALL REQUEST and DCE NOT READY.

Whereas both of these errors would be eliminated by a resolution of the initial collisions, the ambiguity between the initiation of a clearing sequence and a transition to a NOT READY state places constraints on how the collisions may be resolved.

A few errors were flagged because the validation procedure indicates errors when a process returns to its initial state without receiving all events transmitted by the other. In the X.21 interface, it is conceptually possible to do this, when both the DTE and DCE simultaneously cycle from the READY state to a NOT READY state and back again. The validation procedure records this as an error when either returns to the READY state before detecting the NOT READY signals from the other. This could not occur in any real implementation.

Two errors were flagged as a result of an ambiguity we had inadvertently included in the validation procedure. The whole validation procedure, including the derivation of the state diagrams for the DCE and DTE, was automatic once the single state diagram for the X.21 interface was specified. The signals on the R and I circuits in both the CALL PROGRESS and CALLED LINE IDENTIFICATION states in the X.21 state diagram are both given as [IA5, OFF]. Only in an appendix in the specification is the format of the signals specified that permits the DTE to distinguish which of the two signals is being sent by the DCE. This means that an automated validation based on the X.21 state diagram alone reflects an ambiguity between these states which produced an error indication that would occur in practice only if the CALL PROGRESS and CALLED LINE IDENTIFICATION signals were incorrectly interpreted.

Conclusions

The principal aim of the validation we have described was to demonstrate that the validation procedure we have developed can be applied to a reasonably complex protocol. The results demonstrate that a state diagram representation of a protocol permits automated analysis of system behavior at the conceptual level. The use of a well-defined mature interface as a test of the validation system means that the results are not as significant as might be obtained from validating a protocol at an earlier stage of its development. Nevertheless, the results we have

obtained are sufficiently interesting to confirm our belief that the validation procedure described in [1, 2] is a useful tool that will reduce the development time and increase the reliability of communications protocols.

The validation procedure did not identify any errors in the X.21 specification that could result in incorrect operation of the interface during the most probable call establishment procedures.

It did, however, identify a number of collision conditions that have a small but finite probability of occurring and which are not resolved by the specification as described in [3].

Implementations that have been designed with the knowledge that the conditions discussed above can occur will certainly include mechanisms for resolving them. If, however, some designers are unaware that the collisions can occur, they may design interfaces that behave unpredictably in such circumstances, possibly resulting in unrecoverable execution errors.

As a result of validating the X.21 interface, we have obtained a great deal of information about it that would have been difficult to obtain otherwise. It is extremely difficult to understand exactly how two processors interact via a protocol. Generating all of the possible interaction sequences as part of a validation procedure is very useful in this context. We have discussed a number of difficulties resulting from the representation of the X.21 interface in terms of a combined state diagram. It requires a designer implementing the interface to go through a process of interpretation which, even if well defined, is a step which can result in errors that can be costly to correct at later stages of processor development.

The single state diagram representation appears to impose a symmetry on the DCE and DTE design that may prove undesirable in resolving interface problems that occur as the result of intrinsic collisions. We believe that a specification of the interface in terms of separate state diagrams for the two processors would help alleviate these problems.

Acknowledgment

The authors thank E. Port for suggesting the investigation.

References

1. P. Zafiropulo, "Protocol Validation by Duologue Matrix Analysis," *Research Report RZ 816*, IBM Zurich Research Laboratory, 8803 Rüschlikon, Switzerland, 1977. See also P. Zafiropulo, "A New Approach to Protocol Validation," *Proceedings of the International Communications Conference*, Vol. II (ICC 77), Chicago, June 1977, p. 259.
2. C. H. West, "Computer Automated Protocol Validation," *Research Report RZ 817*, IBM Zurich Research Laboratory, 8803 Rüschlikon, Switzerland, 1977. See also C. H. West, "An Automated Technique of Communications Protocol Validation, *Proceedings of the International Communications Conference*, Vol. II (ICC 77), Chicago, June 1977, p. 264.
3. "Recommendation X.21 (Revised)," *AP VI-No. 55-E*, published by the CCITT (International Telegraph and Telephone Consultative Committee), Geneva, Switzerland, March 1976.
4. H. C. Folts, "X.21—The International Interface for New Synchronous Data Networks," *Proceedings of the International Conference on Communications*, Vol. 1 (ICC 75), San Francisco, June 1975, p. 15.

Received June 28, 1977

The authors are located at the IBM Zurich Research Division Laboratory, 8803 Rüschlikon, Switzerland.