C. H. West

General Technique for Communications Protocol Validation

Abstract: A technique for the validation of protocols in communications systems is described. It can be used for systems composed of processes that can be modeled as finite directed graphs. The validation exhaustively exercises the interaction domain of a system and identifies all occurrences of a number of well-defined error conditions. The method can detect when individual processes have no predefined response to incoming messages, as well as system deadlocks and potential loss of messages due to overflow conditions.

Introduction

A communications protocol may be defined as the set of rules that govern the exchange of information between processes in a communications system. A protocol provides a mechanism for system components to exchange control information that ensures coordinated system behavior. It also enables transfer of data and recovery from errors occurring in the transmission medium. The central role of a protocol in a communications system and the difficulty of testing until a system is at an advanced state of development emphasize the importance of the correctness of the protocol design. In this paper we are concerned with the problem of examining the design of a communications protocol to determine whether or not it contains errors.

For the purposes of this paper, we use the term validation to distinguish it from verification because we are concerned with determining whether or not the protocol is sound and its logical structure complete. Verification is more concerned with what the protocol is designed to do and involves a comparison of particular aspects of the protocol behavior with those intended by the designer.

A recent survey by Sunshine [1] on protocol verification points out that verification of the behavior of a protocol presupposes a clear definition of the properties to be verified. Many current protocols are so complex that it is difficult to define a general method that permits the verification of all properties of interest.

A similar situation exists in the related area of computer program verification. Establishing the correctness of a program presupposes a definition of what the program is designed to do. Luckham [2] points out the necessity of a specification on which to base a verification, and also that certain properties of a program, such as range bounds, might be automatically verified for large classes of programs as they are independent of what the program is supposed to do. Many errors in programs can be found automatically. For example, a compiler can detect references to undefined variables, an operating system can detect out-of-range addresses and other error conditions without information concerning what a program is designed to do. Errors that can be detected in this way represent violations of general rules of program behavior. Testing of complex software would be extremely difficult if compilers and operating systems were not able to detect and report many common errors of this type. A program which contains no such errors is not necessarily correct since it may not accomplish the aims of its designer.

The duologue matrix theory of Zafiropulo [3] addresses the validation of a protocol between a pair of asynchronous processes by defining a number of fundamental rules of protocol behavior that are incorporated in a validation function that can be applied to an interaction sequence to determine whether it contains errors. By applying the validation function to all possible interaction sequences defined by a protocol, design errors in the protocol can be found. The validation technique identifies design errors in the form of areas of the protocol design that are incompletely defined and thus result in unpredictable execution in a subsequent implementation.

The theory also detects deadlocks and the presence of interaction steps that can never be executed. A protocol design which is validated in this way and found to contain

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.

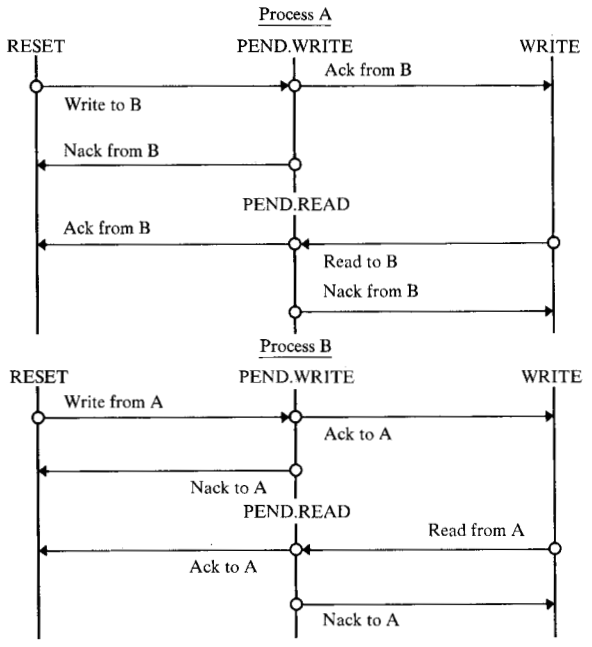


Figure 1 A loop example for a write/read protocol.

no errors will execute in a predictable way when correctly implemented. This does not mean that it necessarily accomplishes the objectives of the designer.

In order to test the theory, a reformulated version has been implemented [4] and used to validate the call establishment procedure specified by the CCITT recommended X.21 interface [5]. Our experience with this and other protocols we have studied has shown that many errors in protocol design lead to situations that can be detected by the validation procedure.

The initial theory was restricted in the range of protocols it could validate, in that it could only be applied to protocols between two processes that both return to their initial state after a finite number of interaction steps.

In this paper we present a generalization of the above validation technique that is applicable to a wider range of protocols. We show that the restrictions of the initial theory may be removed by reformulating the method used to define the interaction domain of the two processes. The initial theory does this by deriving all possible unilogues, or paths through the state diagram representation of each process that start at and return to the initial state. A set of duologues, the Cartesian product of the sets of unilogues for each process, is then systematically analyzed for a number of well defined error conditions [3].

By applying the same fundamental error conditions to a set of system states derived by an iterative perturbation of the initial state of the communicating system, we demonstrate that protocols among more than two interacting processes can be validated without the restriction that the processes need return to their initial states after a finite number of interaction steps.

This state perturbation approach is first described in terms of a few simple examples, which illustrate how it overcomes the limitations of the initial theory. An implementation of the technique is then described in detail, and finally possible extensions of the perturbation approach are discussed.

Limitations of the duologue matrix theory of validation

The duologue matrix theory of protocol validation [3] has the following limitations, which we discuss in detail below.

1. The theory is restricted to the validation of protocols in which the interacting processes must return together to their initial states after a finite number of interaction steps.
2. The theory only addresses protocols between two processes.

• *Return to the initial state*

The limitation of the duologue matrix theory that requires that both processes return to the initial state after a finite number of interaction steps has two consequences. First, it limits the range of protocols that can be validated to those that do not contain embedded loops, i.e., sequences of interaction steps that can be repeated an arbitrary number of times without the processes traversing the initial state. This limitation appears because the presence of loops makes both the number and length of unilogues that each process can execute potentially infinite.

Second, both processes must traverse the initial state with the same periodicity. The division of a continuously progressing interaction into sequences of duologues requires, for example, that one process cannot execute two unilogues while the other is executing one. Such an interaction would require an extension of the theory which is not presently defined.

• *Validation of protocols containing loops*

Considering the first part of this limitation, we examine the protocol between the two processes shown in Fig. 1. This shows two processes that will execute a protocol that ensures that a sequence of write and read instructions is executed, that each instruction can be followed by either a positive or negative acknowledgment, and that in the latter case the instruction will be repeated. This simple protocol could be used, for example, to write data to a storage device and to read them back again for checking purposes.

Throughout this paper we follow closely the nomenclature used to represent protocols in the IBM Systems Network Architecture Formats and Protocols Manual [6]. Processes are represented as directed graphs. A state is shown as a vertical line with its name at the top. Transitions between states are shown as directed links, with text above the link indicating the receipt of an event or message that initiates the transition. Text below the link indicates an event that is transmitted to another process as a result of the execution of the transition. When either is omitted, the implication is that an unspecified action either has initiated the transition or occurs as a result of executing the transition.

The example shown exhibits a loop, i.e., a cycle not traversing the initial state. In principle, when process B receives a read instruction, it can always negatively acknowledge it, producing a continuous cycling of both processes between the WRITE and PEND.READ states, without ever returning to the initial (RESET) state.

Such behavior can produce an infinite number of interaction sequences starting at and returning to the initial state, each one with a different number of negative acknowledgments.

The duologue matrix theory requires extension in order to validate protocols of this type. One can validate such a simple example by induction; a reasonable upper limit is placed on the number of negative acknowledgments. However, a general solution in terms of duologues appears to be quite complex.

The basic philosophy of the perturbation approach, which we present by means of this example, is to examine the complete interaction domain of the system by defining an initial system state, comprising the state of all components of the system when the interaction starts. The interaction domain of the system is then examined by investigating all possible ways in which the initial state and all subsequent states can be perturbed.

Each state that the system can reach is analyzed to determine whether or not it represents a deadlock and whether all processes are able to receive in their current state all events that may have been transmitted to them.

This simple example (and most of the others we discuss) contains no errors. The detection of errors is discussed in detail in a subsequent section. Our current aim is to show how the complete interaction domain of the processes can be examined.

In order to define the state of the system at all times, it is important to realize that the system state not only comprises states of the individual processes of which it is composed, but also of the communications medium which may contain events being transmitted between processes.

Sunshine [7] has made use of this to verify properties of connection establishment protocols in packet switching networks. His analysis is conceptually similar to the one presented here. The state of the communications medium has also been discussed by Bochmann [8], who suggests that in certain types of transmission media it may only be necessary to consider system states in which the transmission medium is empty. For generality, we do not assume that this is the case. The state of the communications medium has been used in [4] to determine the error conditions at each vertex of the phase diagram used to analyze the errors that can arise when a duologue is executed.

In the current example, the communications medium can be modeled as two simplex channels with their states defined in terms of the events they contain. The state of the system is then specified by the states of the two processes and of the two channels linking them.

Figure 2 shows a traversal of the complete interaction domain of the example, which has been performed by generating a complete list of possible states of the system by a perturbation technique. Each system state is a one-line entry in the list, giving the states of the two processes and of the two channels that link them. The states of the channels are represented in terms of the messages they are transporting.

The initial state of the system (system state 0) is the first entry, with both processes in the RESET state and both channels empty.

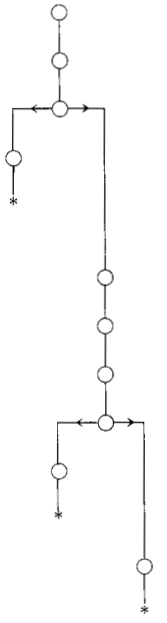
A perturbation of a system state is defined as the execution of a single transition in one process in the system, which implies a change of state of a channel if an event is transmitted or received. We refer to this as a perturbation as it represents the smallest change that can take place in the system at any instant of time.

The execution tree shown to the right indicates the execution paths possible in the system. Transitions between states are shown as lines, and the system states that are linked are shown as circles. An asterisk indicates a system state that has been previously generated by perturbation of an earlier state.

In this example, the only perturbation of the initial state is when A transmits a write instruction to B. This leads to system state 1, with A in state PEND.WRITE, the channel from A to B containing the event write, and the rest of the system unchanged. System state 2 is reached from system state 1 when B receives the write, which is thus removed from the channel.

Both processes are then in the PEND.WRITE state, and now two different perturbations of the system state are possible, according to whether or not B transmits a positive or negative acknowledgment. If a negative acknowledgment is sent, the system returns to its initial state. This need not be further perturbed as we are already investigating all execution sequences that start in the initial state.

System state	*State of A*	*Channel A to B*	*State of B*	*Channel B to A*	*Execution tree*
0	RESET	—	RESET	—	
1	PEND.WRITE	Write	RESET	—	
2	PEND.WRITE	—	PEND.WRITE	—	
3	PEND.WRITE	—	RESET	Nack	
0	RESET	—	RESET	—	
4	PEND.WRITE		WRITE	Ack	
5	WRITE	—	WRITE	—	
6	PEND.READ	Read	WRITE	—	
7	PEND.READ	—	PEND.READ	—	
8	PEND.READ	—	WRITE	Nack	
5	WRITE	—	WRITE	—	
9	PEND.READ	—	RESET	Ack	
0	RESET	—	RESET	—	



*Indicates a state already validated.

Figure 2 Analysis of loop example.

A positive acknowledgment leads on to further states which have not yet been examined. When system state 7 is reached, a situation similar to that of system state 2 exists. Two alternative perturbations are possible: a negative acknowledgment leading back to system state 5, which has already been traversed; a positive acknowledgment leading on to more unexplored states until the initial system state is reached and all possible states of the system have been traversed.

If each of the traversed states of the system is systematically examined for error conditions, and all system states are traversed, then all errors that can occur during system execution can be found, irrespective of the particular sequence of transitions that leads to a system state that manifests an error.

A complete discussion of the errors that can be found in this way is given in a later section.

• *Analysis of protocols between unmatched processes*

The requirement of the duologue matrix theory that both processes return to the initial state after a finite number of interaction steps excludes from validation protocols such as the one shown in Fig. 3. Here another write/read protocol is shown, but for simplicity, without any negative acknowledgments. A significant difference from the previous example in Fig. 1 is that, whereas the first process is constrained to transmit a read after each write instruction, the second can respond to any arbitrary sequence of write and read instructions, since both write and read can be received in their initial states.

The protocol is simple enough for inspection to show that it contains no errors, yet the duologue matrix theory would indicate an error. When process A executes the unilogue resulting in the transmission of read and write to process B, B executes two unilogues with an intermediate traversal of the initial state. An error would be indicated because B returns to the initial state without having received all events transmitted during the execution of the unilogue in A.

The perturbation approach can analyze such a protocol, as is shown in Fig. 3. When the second process returns to its initial state, the system as a whole is in a state that has not been previously executed, so that the perturbation continues until the system as a whole returns to its initial state.

The significance of protocols of the type shown in Fig. 3 is that the two processes have not been specifically designed to operate together. Process B is a more general process that may operate correctly with a number of different processes. In computer systems, a single process is often designed to operate in several different environments, as an alternative to having different, special-purpose processes for each. It is important that a general validation technique be able to address such protocols.

• *Validation of protocols between multiple processes*

Zafiropulo [9] has pointed out that an extension of the duologue matrix theory to validate protocols among more than two processes is possible if transitions between states are labeled with the name of the process with which an event is exchanged, as well as the event. However, the details of such an extended theory have not been developed. We have used such a labeling convention in the examples discussed above.

The disadvantage of such an extension of the duologue matrix theory would appear to lie in the interaction domain being defined by the Cartesian product of unilogue sets, which becomes very large if many processes are involved.

Figure 4 shows a simple protocol among three processes, which is examined to illustrate how multiprocess protocols can be analyzed. In the simple protocol shown, A may represent a program, C a file, and B an intermediate access mechanism. Figure 4 also shows how the system can be validated.

A system is defined which consists of the three processes and four channels that link the process pairs as defined by the specified event exchanges. Starting with all processes in the RESET state, successive perturbations traverse the interaction domain in the same way as in the previous examples. An examination of each system state traversed for errors enables a complete validation of the system to be performed. Note that the number of system states traversed is less than the number of states in the Cartesian product of the three processes. The perturbation technique only explores the accessible states of the system, and so is intrinsically efficient.

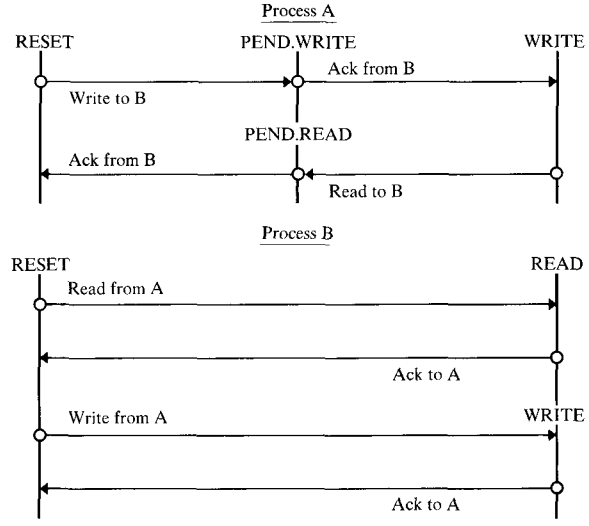
Error conditions detected by validation

In the previous sections we have shown by a series of examples how the perturbation technique can overcome most of the limitations of the duologue matrix theory of protocol validation by changing the method of traversing the interaction domain of the processes executing the protocol. In order to concentrate on this, we have only considered examples that contain no errors. In this section we discuss the error conditions that the validation procedure can detect, and then in the following section we show how the error detection and interaction domain traversal can be combined in a validation procedure.

The errors that can be detected by the duologue matrix theory of protocol validation are of two types when expressed in terms of the process and channel states [4].

First, there are deadlock errors, such that the two processes are both in states from which further execution is dependent on the reception of an event, but there are no events underway in the channels that link them.

Second, there are reception errors, which can occur when either process is in a state such that there is no departing transition which corresponds to the reception of an event which is underway to it in a channel.



Nonmatching example analysis

System state	*State of A*	*Channel A to B*	*State of B*	*Channel B to A*	*Execution tree*
0	RESET	—	RESET	—	
1	PEND.WRITE	Write	RESET	—	
2	PEND.WRITE	—	WRITE	—	
3	PEND.WRITE	—	RESET	Ack	
4	WRITE	—	RESET	—	
5	PEND.READ	Read	RESET	—	
6	PEND.READ	—	READ	—	
7	PEND.READ	—	RESET	Ack	
0	RESET	—	RESET	—	*

*Indicates a state already validated.

Figure 3 Example with nonmatching processes.

These two types of error conditions can be simply generalized within the framework of the perturbation approach and are formally stated in the next section.

A third error condition that can also be detected is overflow. Any communications medium has a storage capacity that limits the amount of information it can contain at any instant. An attempt to exceed this capacity may result in loss of information being transferred between processes.

Bochmann [8] has discussed the representation of the communications medium in terms of queues of messages in transit and points out that the number of messages in

transit may be large. He is able to simplify the verification of properties of protocols by using regular expressions to characterize the contents of message queues but points out that this does not capture all of the properties of the communications medium. One such property is that a communications medium has a finite capacity for information that it can be transporting at any time. In a network or computer system this is governed by the capacity of the buffers that the information must pass through on its way between processes and possibly by the overall level of traffic in the network. In communications hardware it is governed by the number and size of intermediate registers.

An important function of a protocol is to ensure that no overflow of the storage capacity takes place that may result in loss of information.

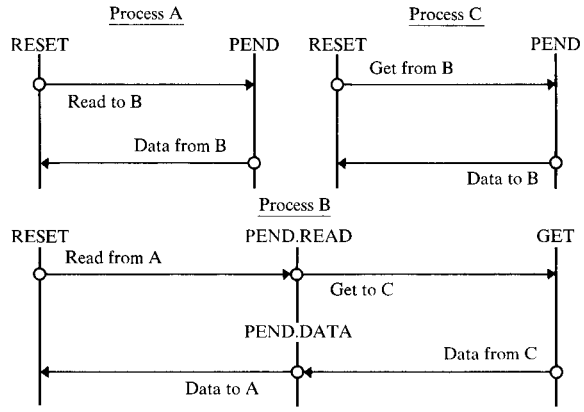
Protocols generally do this by controlling the transfer of information into the communications medium. Two control mechanisms are available. The first is to make use of timing constraints. If the communications medium and receiving process can accept information at a well-defined rate, the transmitting process can be physically designed to transmit at a compatible rate. In other circumstances, transmission is made conditional on acknowledgments, which indicate the successful completion of information transfer.

This latter mechanism can be directly expressed in the protocol representation we are using. The former implies reference to timing constraints, which are not currently considered as part of our validation procedure. However, the timing constraints are generally of such a nature that their verification can be treated independently of the validation of the logical structure of the protocol.

In the current context, the main interest of protocols that rely on timing to avoid overflow is their behavior when their logical structure is validated by a perturbation technique.

Consider the simple example shown in Fig. 5. Here a simplex protocol is shown by means of which one process sends a continuous sequence of messages to the other. The protocol obviously contains no deadlock. Also the protocol is such that the receiver is always in a state such that it can accept the first incoming message; therefore, it does not manifest the first two error conditions discussed above. The protocol as shown may cause overflow in the communications medium since the sender is always in a state from which it can transmit, and there is nothing in the logical structure of the processes that prevents it from transmitting an infinite number of messages before any are received. A traversal of the interaction domain will

Figure 4 Multiprocess example.



Multiprocess example analysis

System state	*State of A*	*Channel A to B*	*State of B*	*Channel B to A*	*Channel B to C*	*State of C*	*Channel C to B*	*Execution tree*
0	RESET	—	RESET	—	—	RESET	—	○
1	PEND	READ	RESET	—	—	RESET	—	○
2	PEND	—	PEND.READ	—	—	RESET	—	○
3	PEND	—	GET	—	GET	RESET	—	○
4	PEND	—	GET	—	—	PEND	—	○
5	PEND	—	GET	—	—	RESET	DATA	○
6	PEND	—	PEND.DATA	—	—	RESET	—	○
7	PEND	—	RESET	DATA	—	RESET	—	○
0	RESET	—	RESET	—	—	RESET	—	*

*Indicates a state already validated.

not terminate since there is an infinite number of system states, differing from one another in the number of messages underway in the communications medium.

The addition of a timing constraint that prevents the overflow has the effect of putting an upper bound on the number of messages underway, thus limiting the number of distinct states of the system. Conversely, the assumption of an upper bound on the number of messages underway is equivalent to assuming a timing constraint.

The assumption of a finite storage capacity for a channel linking a pair of processes is thus useful to bound the interaction domain when validating the logical structure of protocols that have certain implied timing constraints. In protocols which rely on acknowledgments to prevent overflow, the generation during validation of an overflow that exceeds a predefined storage capacity indicates a protocol error. It is also possible to use overflow to verify particular properties of protocols. For example, Synchronous Data Link Control [10] normally requires no more than seven messages to be sent without acknowledgment. A validation with a storage capacity of seven messages would verify this design rule.

Throughout this paper we use the number of messages or events underway as a measure of storage capacity, as we are considering events as indivisible entities. In practice, it may be more useful to measure storage capacity in terms of bits or bytes and associate an appropriate length with each message.

An overflow is considered to take place if the transmission of an event results in a channel containing a number of events greater than a predefined maximum. The interpretation of an overflow depends on which of the two mechanisms described above is assumed to prevent overflow during protocol execution.

An overflow in a protocol that uses acknowledgments to limit transmission represents an error. On the other hand, if timing is used as part of the protocol to prevent overflow, transmission of an event which implies overflow represents a region of the interaction domain of the processes which will not be executed in practice. In this case, it does not represent an error; it is merely necessary to suppress perturbations that generate overflow conditions to avoid investigating regions of the interaction domain which have no practical significance.

Algorithms for validation by perturbation

In the previous sections we discussed how a technique of system state perturbation can be used to explore the interaction domain of communicating processes and the nature of the error conditions that may be detected.

In this section we present the algorithms that have been used in an implemented system for validating protocols for a particular model of a communicating system in order to present in a more complete way the validation concepts discussed informally in the previous sections. We first define the model of the communications system we are considering.

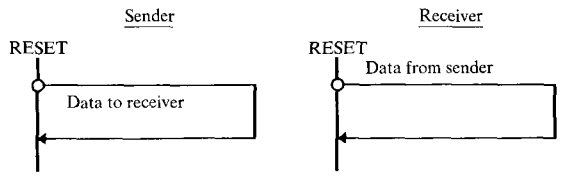


Figure 5 A simplex protocol.

• *Communications system modeled*

The communications system we consider is assumed to be composed of N processes $G(I)$, where $I = 1$ to N. Each process is represented as a directed graph whose nodes represent process states and whose arcs represent transitions between states. We denote the current state of process $G(I)$ at any time by $Gs(I)$.

Communication between processes is assumed to take place via a communications medium, which we model as a set of simplex channels, each linking a pair of processes, so that $C(I, J)$ represents a channel that can transport information from process I to process J.

The unit of information transferred is defined as an event, which is assumed to be indivisible.

The execution of a transition in any process in the system changes the current state of that process according to the topology of the directed graph. A transition may also be associated with the exchange of an event with a channel connecting the process with another, so that it can take place only within the framework of a set of rules governing the interaction of a process with the communications medium. The exchange of events that accompanies the execution of a transition is defined by a label associated with the transition.

Three types of labels are assumed for a transition in process $G(I)$:

1. "Event-name to J," which indicates that execution of the transition results in the named event being inserted into the channel leading from process I to process J.
2. "Event-name from J," indicating that the transition takes place when the named event is detected in the channel from process J to process I.
3. "No output," signifying a transition that can take place without exchange of events with the communications medium.

The state of a channel $C(I, J)$ at any time is represented as $Cs(I, J)$, an ordered set of the names of events that it is

$$
\begin{array}{ccccccccc}
Gs(1) & Cs(1,2) & \cdot & \cdot & \cdot & Cs(1,I) & \cdot & \cdot & \cdot & Cs(1,N) \\
Cs(2,1) & Gs(2) & & & & \cdot & & & & \\
\cdot & & \cdot & & & \cdot & & & & \\
\cdot & & & \cdot & & \cdot & & & & \\
\cdot & & & & \cdot & \cdot & & & & \\
Cs(I,1) & \cdot & \cdot & \cdot & \cdot & Gs(I) & \cdot & \cdot & \cdot & Cs(I,N) \\
\cdot & & & & & \cdot & \cdot & & & \cdot \\
\cdot & & & & & \cdot & & \cdot & & \cdot \\
\cdot & & & & & \cdot & & & \cdot & \cdot \\
Cs(N,1) & \cdot & \cdot & \cdot & \cdot & Cs(N,I) & \cdot & \cdot & \cdot & Gs(N)
\end{array}
$$

Figure 6 Matrix representation of a system state.

transporting between processes, where $Cs(I, J)[L]$ represents the Lth member of the set.

We assume that channels have the following properties.

1. The channel $C(I, J)$ will accept events inserted in it by process $G(I)$ and deliver them to process $G(J)$ in the same order as they are received.
2. The time taken to transport events between processes is variable and unspecified.
3. Each channel $C(I, J)$ has a predefined maximum number of events that it can be transporting at any instant of time. This we refer to as its storage capacity, $Cap(I, J)$.

• *System behavior*

At any instant of time the condition of the system can be completely represented by a system state.

We define a system state $S(M)$ as being a unique set of the current states of the processes $Gs(I)$ for all I from 1 to N and of all channels $Cs(I, J)$ for all I and J from 1 to N but not $I = J$. The index M identifies a particular system state.

System state $S(M)$ can be envisaged as a matrix of states as shown in Fig. 6.

We define a perturbation of a given system state $S(M)$ as being another system state which can be reached by executing a single transition in one process $G(I)$ from its current state $Gs(I)$. By definition, the perturbation can only change the current state of the one process $G(I)$, and it follows that the state of only one channel $Cs(I, J)$ or $Cs(J, I)$ will change, dependent on the executed transition representing a transmission to process J or a reception from process J. Any change in the system state involving simultaneous transitions of more than one process can be represented as a sequence of perturbations.

• *Rules governing transitions*

1. When an event arrives at a process, it must be immediately received, and as a consequence it is removed from the channel. Because the time taken for a channel to transport events from one process to another is not specified, it follows that the current state of a process must at all times be able to absorb the first incoming event in all channels incident on the process.
2. At any time any transition from the current state of a process which does not correspond to the reception of an event may be executed.
3. No transition may take place that results in an overflow of a channel's storage capacity.

• *Errors*

There are three classes of errors, which correspond to violations of the above rules.

Reception errors

An incoming channel $C(J, I)$ to process $G(I)$ contains an event, but there is no departing transition from the current state $Gs(I)$ by which the first incoming event $Cs(J, I)[1]$ can be received, i.e., no transition having a label equivalent to "$Cs(I, J)[1]$ from J."

Deadlock errors

For all I, the current state $Gs(I)$ of $G(I)$ is such that there are no transitions from it corresponding to the transmission of an event or no output, and the state of all incident channels $Cs(J, I)$ is an empty set for all J.

Overflow error

For a given process $G(I)$, there exists a transition from the current state with a label "Event-name to J" such that its execution will result in the storage capacity $Cap(I, J)$ of channel $C(I, J)$ being exceeded.

• *Algorithm for validation by system state perturbation*

The validation algorithm may be stated as follows.

Step 1. Define a set of system states S, initially containing only the initial state $S(0)$ of the system when execution commences.

Step 2. Find a member $S(N)$ of the set of system states whose perturbations have not been determined. If no such member exists, terminate as the validation is completed.

Step 3. Calculate the set of system states Sp that can be reached by a perturbation of $S(N)$. There will be one member of Sp for each executable transition of all processes from their current states as defined in $S(N)$.

Step 4. If Sp is an empty set, report $S(N)$ as a terminal (or deadlock) state of the system.

Step 5. Remove all states manifesting a reception error or overflow condition from Sp, and report the errors found.

Step 6. Add all remaining members of Sp which are not already members of S to the set S.

Step 7. Repeat from step 2.

During the validation a record of each perturbation executed can be maintained. Each perturbation can be characterized by parameters identifying the two system states

linked by the perturbation, the process executing the transition, and the actual transition executed by the process.

This record specifies the interaction domain of the system validated in the form of a single directed graph whose nodes are system states and whose arcs identify possible transitions between system states. This is a state machine for the system as a whole; it has similar properties to a token machine that can be derived from a Petri net description of a protocol [11].

It completely specifies the communications system behavior and can be used as a basis for further analysis. Bochmann [12] has discussed the suitability of such state machines for the verification of particular protocol properties. Cyclic behavior and synchronization, for example, can be investigated, the latter by means of the concept of adjoint states [13].

Performance of the perturbation validation system

The validation procedure defined in the previous section has been implemented in APL; validation of a simple protocol is discussed in the Appendix to show how the system can be used. It is of interest to discuss briefly the computer time required for validation when the procedure presented above is used, as this is an important parameter which limits the complexity of protocols that can be validated.

In this context we are primarily concerned with the complexity of a protocol or part of a protocol that must be validated as an entity. Many protocols can be considered as separate layers or be otherwise subdivided into parts that may be validated separately.

We repeated the validation of the X.21 protocol discussed in [5] using the perturbation approach. The results obtained were equivalent to those obtained in the previous validation, except that a few problems identified with limitations of the earlier validation technique were removed.

The validation required approximately 30 seconds of cpu time on an IBM System 370, model 158. An implementation of the perturbation method in a compilable language and suitably optimized should require an order of magnitude less time. A protocol of the order of complexity of the X.21 may thus be validated in a few seconds.

The principal limitation of the validation algorithm is the necessity to check whether or not each state reached by a perturbation has already been validated. The time required depends on the total number of possible system states, the number of parameters required to represent each state, and the efficiency of the search algorithms used. Experience with the X.21 validation suggests that protocols which are several orders of magnitude more complex might be validated in this way before computer time or storage become serious limitations.

A difficulty in trying to foresee what limits the complexity of protocols that might be validated is that it depends on the size of the system interaction domain rather than the number and complexity of the processes in the system. The number of system states is strongly dependent on the degree of coupling between processes and the possible number of events that may be underway in the channels.

Extensions to the validation procedure

A number of extensions to the validation procedure as described above can be envisaged.

The validation method and communications model used do not currently address the validation of protocols where specific timing constraints are important. These are commonly associated with timeouts, which can only be of unspecified length in the current model. Merlin [14] has discussed the modeling of time constraints in time Petri nets as applied to an analysis of the ability to recover from failures. A similar extension to the perturbation approach should be possible.

A number of extensions to the communications model are of interest to permit the validation of a wider range of systems.

A channel as currently defined in the model links only a single pair of processes and does not reorder the events it is transporting. Both of these limitations can be removed by suitable reformulation of the procedure and channel model.

Processes are represented as directed graphs, their function in the model being to generate and accept events in predefined sequences. The directed graph representation is inconvenient for modeling processes containing counters or internal logic which is crucial to the correct operation of the protocol. Such processes are more conveniently modeled in terms of procedures rather than directed graphs. Both Bochmann [12] and Danthine [15] use procedural modeling in terms of a programming language for protocol definition and verification. Similar modeling techniques could also be used in the validation procedure. It should, however, be noted that errors in the procedures would only be detected if they resulted in the error conditions discussed in previous sections.

Conclusions

In this paper we have discussed an extension of the duologue matrix theory of communications protocol validation which overcomes a number of limitations of the original theory.

By defining a system state, consisting of the states of all processes in the system and the states of all channels link-

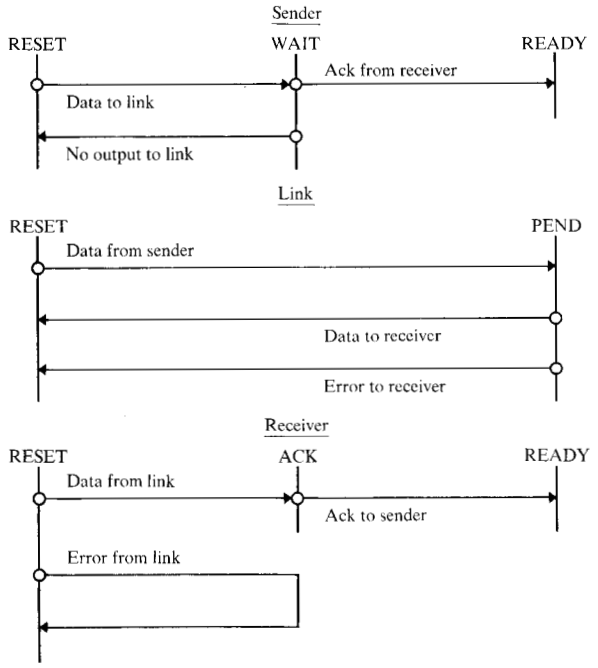


Figure 7 Protocol modeled with transmission error.

ing processes, the interaction domain of the system can be traversed by a series of perturbations of the current system state.

This permits a search of the interaction domain for error conditions similar to those detected by the duologue matrix theory and permits a unified extension of the validation technique to a larger class of protocols.

The perturbation technique can be used to validate protocols among multiple processes, protocols in which the processes do not necessarily return to the initial state after a finite number of interaction steps, and it enables certain classes of transmission errors to be modeled.

The technique has been discussed in terms of quite a simple model of a communicating system. In the model, individual channels link a single pair of processes and obey a first-in/first-out queuing discipline. The perturbation technique can be generalized to encompass the validation of a wider range of systems.

A number of extensions of the system remain to be defined. The most significant is the validation of protocols containing explicit timing constraints.

An important property of the validation technique is that it detects protocol design errors by systematically searching the interaction domain of the protocol for the violation of a few simple and universal error conditions. A protocol which has been validated without any errors being identified will behave in a manner that can be predicted from the design. The validation does not permit verification of particular protocol properties that may be desired by a designer. However, many design errors result in unpredictable protocol behavior when particular unforeseen conditions arise during execution of a subsequent implementation. Such errors are extremely difficult to find during testing, yet can be identified by the validation procedure we have described. The validation procedure, therefore, promises to be an important tool for protocol designers and will lead to enhanced reliability of communications protocols.

The validation technique can be completely automated and requires only a formal definition of a protocol in state diagram form. Errors which the validation identifies can be readily confirmed by referring to the design.

Acknowledgments

The author thanks E. Port, H. Rudin, and P. Zafiropulo for many useful suggestions and discussions during the course of this work.

Appendix: A sample validation of a positive acknowledgment with retransmission protocol

The perturbation approach to validation has been implemented in an APL based system as described in the algorithm in the body of this paper. In this Appendix we show how the system can be used to validate the simple protocol shown in Fig. 7.

Here a simple Positive Acknowledgment, Retransmission on Timeout protocol is shown, by which a sender communicates data to a receiver and thus changes the state of the receiver. The sender transmits the data, waits for an acknowledgment, and retransmits the data if no response is received after a timeout. In this example, the timeout is modeled by the transition labeled "No output." The receiver waits in its initial state for the data and ignores any erroneous information, only responding to an error-free event.

Errors are modeled by interposing a third process, link, between the sender and receiver. Link receives the data from the sender and can pass it to the receiver or transmit an error. This permits the insertion of transmission errors into the channel from the sender to the receiver and enables particular classes of errors to be modeled during the validation. This is an alternative to modeling errors by inserting additional transitions in the transmitting process as described in [3]. It has the advantage of localizing the error modeling and is formally equivalent to the technique described in [3].

Obviously, loss or duplication of events, and any other specific error mechanisms, can be modeled in this way by appropriately defining the intermediate process.

Figure 8 shows the definition of the example which is used as input to the validation system. The user creates an APL function which contains a definition of the system to be validated; the name of the function, in this case PAR,

is arbitrary. The function contains a number of statements which define different aspects of the system. It starts with a SYSTEM statement defining the names of the processes composing the system, followed by one or more EVENTS statements defining the names of events which may be exchanged between processes.

Each channel is then defined by a separate CHANNEL statement, which defines the processes linked by the channel and the channel storage capacity, i.e., the maximum number of events it may be transporting at any instant. This is defined as negative if overflows in the channel are not to be recorded as errors.

A PROCESS statement initiates the definition of a process, with one or more STATES statements defining the names of the process states. A state name starting with an asterisk indicates a transient state, an internal state in which the process cannot receive events. This is a useful mechanism that permits the modeling of interrupt-driven processes and indicates states where interrupts are disabled. In this example, it prevents the link from accepting an incoming event before it has processed a previous one. The validation system treats a transient state as one that must be immediately perturbed when entered and, by definition, will not manifest reception errors.

Each transition is defined by a TRANSITION statement giving the states it links and the event and destination/source process involved in the communication. An END statement terminates the definition.

Execution of the defined function checks the definition for trivial errors and formats the definition for subsequent validation. Figure 9 shows a typical error report generated when the example was validated. It shows the error condition detected, the states of the processes and channel contents when it can occur, and the sequence of execution steps that each process has executed that have led to the error. The latter are derived from the system state machine defined by the traversal of the protocol interaction domain.

The sample error report shows one problem of this protocol when used in an environment where the transmission delay between the sender and receiver is not well defined. Referring to Fig. 7, the report shows that the sender can timeout back to its initial state, while an acknowledgment is underway. The acknowledgment can then be received in the reset state where it is not provided for.

```
PAR
SYSTEM 'SENDER,LINK,RECEIVER'
EVENTS 'DATA,ERROR,ACK'
'SENDER TO LINK' CHANNEL -1
'LINK TO RECEIVER' CHANNEL -1
'RECEIVER TO SENDER' CHANNEL -1
PROCESS 'SENDER'
STATES 'RESET,WAIT,READY'
'RESET TO WAIT' TRANSITION 'DATA TO LINK'
'WAIT TO RESET' TRANSITION 'NO OUTPUT TO LINK'
'WAIT TO READY' TRANSITION 'ACK FROM RECEIVER'
PROCESS 'LINK'
STATES 'RESET,* PEND'
'RESET TO * PEND' TRANSITION 'DATA FROM SENDER'
'* PEND TO RESET' TRANSITION 'ERROR TO RECEIVER'
'* PEND TO RESET' TRANSITION 'DATA TO RECEIVER'
PROCESS 'RECEIVER'
STATES 'RESET,ACK,READY'
'RESET TO RESET' TRANSITION 'ERROR FROM LINK'
'RESET TO ACK' TRANSITION 'DATA FROM LINK'
'ACK TO READY' TRANSITION 'ACK TO SENDER'
END
```

Figure 8 Definition of the PAR protocol for validation.

```
SENDER CANNOT RECEIVE ACK FROM RECEIVER IN SYSTEM STATE 21
-----------------------------------------------------------

PROCESS STATES
--------------

SENDER   -RESET
LINK     -RESET
RECEIVER -READY

CHANNEL CONTENTS
----------------

RECEIVER TO SENDER  -ACK

EXECUTION SEQUENCE-

SENDER
------

RESET  TO WAIT   EVENT DATA        TO  LINK
WAIT   TO RESET  EVENT NO OUTPUT   TO  LINK

LINK
----

RESET  TO * PEND  EVENT DATA        FROM SENDER
* PEND TO RESET   EVENT DATA          TO  RECEIVER

RECEIVER
--------

RESET  TO ACK     EVENT DATA      FROM  LINK
ACK    TO READY   EVENT ACK         TO   SENDER
```

Figure 9 Sample error report for PAR protocol.

References

1. C. A. Sunshine, "Survey of Communication Protocol Verification Techniques," presented at *Trends and Applications 1976: Computer Networks*, Gaithersburg, MD, November 1976, sponsored by the IEEE Computer Society Technical Committee on Computer Communications and the National Bureau of Standards.
2. D. C. Luckham, "Program Verification and Verification-Oriented Programming," *Information Processing 77, Proceedings of IFIP Congress 77* (Toronto), North-Holland Publishing Co., Amsterdam, 1977, p. 783.
3. P. Zafiropulo, "Protocol Validation by Duologue Matrix Analysis," *IEEE Trans. Commun.*, to be published.
4. C. H. West, "An automated Technique for Communications Protocol Validation," *IEEE Trans. Commun.*, to be published.
5. C. H. West and P. Zafiropulo, "Automated Validation of a Communications Protocol: the CCITT X.21 Recommendation," *IBM J. Res. Develop.* **22,** 60 (1978).

6. *Systems Network Architecture. Format and Protocol Reference Manual: Architecture Logic*, SC 30-312-0, IBM Corporation, White Plains, NY, 1976.
7. C. A. Sunshine, "Interprocess Communication Protocols for Computer Networks," *Technical Report 105*, Digital Systems Laboratory, Stanford University, Stanford, CA, December 1975.
8. G. V. Bochmann, "Finite State Description of Communications Protocols," Publication No. 236, Département d'Informatique, Université de Montreal, July 1976.
9. P. Zafiropulo, private communication.
10. R. A. Donnan and J. R. Kersey, "Synchronous Data Link Control: A Perspective," *IBM Syst. J.* **13,** 140 (1974).
11. P. M. Merlin, "A Methodology for the Design and Implementation of Communications Protocols," *IEEE Trans. Commun.* **COM-24,** 614 (1976).
12. G. V. Bochmann and J. Gecsei, "A Unified Method for the Specification and Verification of Protocols," *Information Processing 77, Proceedings of IFIP Congress 77* (Toronto), North-Holland Publishing Co., Amsterdam, 1977, p. 229.
13. G. V. Bochmann, "Communications Protocols and Error Recovery Procedures," *ACM SIGOPS Operating Syst. Rev.* **9,** 45 (1975).
14. P. M. Merlin and D. J. Farber, "On the Recovery of Communications Protocols," *Proceedings of the International Communications Conference (ICC76)* (Philadelphia), June 1976, p. 20.
15. A. A. S. Danthine and J. Bremer, "An Axiomatic Description of the Transport Protocol of Cyclades," presented at the Professional Conference on Computer Networks and Teleprocessing, Aachen, Germany, March 1976.

Received September 30, 1977

The author is located at the IBM Zurich Research Division Laboratory, 8803 Rüschlikon, Switzerland.