Protocol Conversion

SIMON S. LAM, FELLOW, IEEE

***Abstract*—Consider the problem of achieving communication between two processes across a network or an internetwork. To address this problem, we first formalize the notion of logical connectivity between processes in a protocol architecture; logical connectivity is a necessary condition for conversion-free communication. We then formulate the problem of constructing a protocol converter to achieve interoperability between processes that implement different protocols. We present a formal model, based upon the theory of protocol projection [8], for reasoning about the semantics of different protocols and conversions between them. The correctness of a conversion is a consequence of the correctness properties of image protocols obtained by projections. Two kinds of converters are presented: memoryless converters and finite-state converters. Lastly, we illustrate the construction of some finite-state converters with examples.**

***Index Terms*—Communication protocols, computer networks, internetworking, protocol architecture, protocol conversion, protocol projection, protocol verification.**

I. Introduction

With the proliferation of network architectures and communication protocols, it becomes increasingly difficult to ensure that users connected to different networks can communicate. It may be argued that the solution to this problem is simply to agree upon one worldwide standard protocol architecture, say Open Systems Interconnection [18], or one internetworking protocol, say TCP/IP or X.25/X.75, to be used by all suppliers of hardware and software [2], [4], [17]. In a recent article, Green [5] reviewed the protocol conversion problem from the architectural point of view, reviewed current ad hoc solutions, and argued convincingly that protocol conversions will be a permanent fact of life. He gave two main reasons. First, it is already too late to try to get everyone to adhere to the same standard. There is an installed base of over 20 000 IBM SNA networks, over 2000 DECnet networks, several hundred DoD TCP/IP networks, as well as many other vendor-specific networks. Second, convergence to a global standard implies that all tradeoffs are understood and all inventions are made and assimilated, which is obviously not the case in the relatively young field of computer communications.

Even in the absence of significant architectural mismatches, achieving interoperability between different variants of the same protocol is a nontrivial task. Many protocol standards developed with the intention of fostering compatibility ended up as families of different standards [1], [2]. A standard as basic as RS-232 has many variants [11]. The data link protocol standard HDLC has many siblings: SDLC, ADCCP, LAP, LAPB, LAPB Multilink, etc. Even HDLC itself defines, in addition to a basic repertoire of commands and responses, a wide variety of optional capabilities for implementors to pick and choose from (thus fostering incompatibility between independently implemented versions of the protocol) [7].

To date, there have been a few protocol conversions attempted [5], [6]. Subsequent to the state-of-the-art review of this subject by Green, some formal models for protocol conversion have been proposed [9], [12]. However, there is no general theory for understanding the protocol conversion problem. When is a conversion needed? What is meant by a correct conversion? Or a useful conversion? How do we construct a converter? In this paper, we attempt to provide some (but not all) of the answers to these questions.

In Section II, we present conditions for determining whether protocol conversion is needed, and where it is needed, in a given protocol architecture in order for a pair of processes to communicate. In Section III, we formulate the problem of constructing a converter to achieve interoperability between processes that implement different protocols. The theory of protocol projection is reviewed. Image protocols obtained by projections are shown to be useful for reasoning about the semantics of different protocols and conversions between them. The correctness and functionality of a converter are inferred from properties of image protocols. Two kinds of converters are presented: memoryless converters and finite-state converters. We illustrate our method with some examples of finite-state converters between the alternating-bit protocol and a nonsequenced protocol for data transfer.

II. Logical Connectivity

Two processes P_1 and P_2 are said to *interoperate* if they implement the same protocol. Interoperability implies that each process "understands" the syntax and semantics of the messages it receives from the other process. For a communication protocol, interoperability also implies a data transfer service provided by the processes to other processes.

Suppose that processes P_1 and P_2 implement, or *speak*, protocol P, while processes Q_1 and Q_2 speak protocol Q whose function is similar to that of P. Can P_1 and Q_2 interoperate? Or Q_1 and P_2? One approach to achieve interoperability is to construct a converter process C_1 (C_2)

Manuscript received February 15, 1987; revised October 1, 1987. This work was supported by the National Science Foundation under Grants ECS-8304734 and NCR-8613338.

The author is with the Department of Computer Sciences, the University of Texas at Austin, Austin, TX 78712.

IEEE Log Number 8718695.

0098-5589/88/0300-0353$01.00

as an intermediary between processes P_1 and Q_2 (Q_1 and P_2). The converter accepts the messages of one protocol from one process, interprets them, and delivers to the other process messages of the other protocol that are semantically equivalent. In particular, the network of three processes (P_1, C_1, Q_2), or (Q_1, C_2, P_2), provides *some* useful function, such as a data transfer service to other processes. In this case, we say that P_1 and Q_2 interoperate via C_1, and Q_1 and P_2 interoperate via C_2. Note that we have deliberately left undefined the level of useful function for a protocol conversion to be considered successful. The functional requirement of a protocol conversion can be specified formally like the functional requirement of any protocol and is to be determined by the designer of the protocol.

The problem of achieving interoperability between processes is addressed in Section III. Consider now two processes, say P_1 and P_n, that exchange messages through a network or an interconnection of networks. How do we determine if protocol conversion is needed to achieve communication between P_1 and P_n? The rest of this section is devoted to stating some conditions for determining where protocol conversion is needed, if it is needed, in a given protocol architecture in order for two processes to communicate.

The protocol architecture of a network, or an interconnection of networks, can be represented by an undirected graph (V, A), where V is a set of vertices and A is a set of arcs specified as unordered pairs of vertices. Each vertex represents a process[1] which speaks one or more protocols; in layered architectures, for example, a process may speak interface protocols to processes in upper and lower layers, as well as peer protocols to other processes in the same layer. Each arc represents a pair of physical channels for processes at the two vertices of the arc to send messages to each other. In real networks, physical channels may be internode communication links or they may be intranode channels provided by some interprocess communication facility. We will not distinguish them.

We pose the following problem. Given a protocol architecture specified as described above and two processes, P_1 and P_n, in the architecture, how do we check if protocol conversion is needed for P_1 and P_n to communicate? The following conditions are clearly necessary for conversion-free communication between P_1 and P_n:

1) P_1 and P_n interoperate, and

2) processes and physical channels in the architecture collectively provide a data transfer service for delivering messages sent by P_1 to P_n and messages sent by P_n to P_1.

Condition 1) is satisfied if P_1 and P_n implement the same protocol; otherwise, protocol conversion is needed to achieve interoperability between P_1 and P_n. Both conditions 1) and 2) would be sufficient, as well as necessary, if the data transfer service provided by the architecture meets the functional specifications for message delivery between P_1 and P_n.

Given a graph representation of a protocol architecture, condition 2) cannot be easily checked. We next introduce two weaker conditions, namely, physical connectivity and logical connectivity. Physical connectivity is a necessary condition for logical connectivity, while logical connectivity is a necessary condition for conditions 1) and 2) above. Logical connectivity and physically connectivity between a pair of processes can be checked algorithmically.

Definition: P_1 and P_n are *physically connected* in a protocol architecture if, and only if, they are the head and tail of a sequence of processes in the architecture wherein adjacent processes are directly connected by physical channels.

If P_1 and P_n are not physically connected, some physical channels and processes will have to be added to the architecture to complete the physical path.

Definition: P_1 and P_n are *logically connected* in a protocol architecture if, and only if,

1) $P_1 = P_n$ (each process is logically connected to itself), *or*

2) P_1 and P_n interoperate and are directly connected by physical channels, *or*

3) P_1 and P_n interoperate and there exist processes P_2 and P_{n-1} such that P_1 and P_2 are logically connected, P_2 and P_{n-1} are logically connected, and P_{n-1} and P_n are logically connected, where P_2 and P_{n-1} may be the same process. (See Fig. 1.)

Logical connectivity is a stronger condition than physical connectivity and is useful for checking if there are processes along the physical path between P_1 and P_n which must interoperate but which do not implement a common protocol. (Hence protocol conversion is needed to make these processes interoperate.) The recursive nature of the definition makes it easy to check logical connectivity between process pairs in a protocol architecture represented as an undirected graph. Application to a layered protocol architecture is especially easy. In Figs. 2–4, we illustrate the definition of logical connectivity by considering several internetwork architectures.

Fig. 2 illustrates the basic architecture of X.25/X.75 internetworks [2], [17]. The two user processes in the hosts are logically connected if, and only if, they interoperate, the X.25 processes in the hosts are logically connected and, in each host, the user process interoperates with the X.25 process. The X.25 processes in the hosts are logically connected if, and only if, the X.25 processes in networks 1 and 2 are logically connected. The X.25 processes in networks 1 and 2 are logically connected if, and only if, the X.25 and X.75 processes are logically connected. Logical connectivity between the X.25 and X.75 processes in each network requires that the X.25 and X.75 protocols are sufficiently close to each other for the processes to interoperate.

Fig. 3 illustrates the basic architecture of TCP/IP internetworks [4]. The two user processes are logically con-

[1] What we call a process may be implemented either as a sequential process or a network of processes.

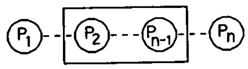


Fig. 1. Logical connectivity definition.

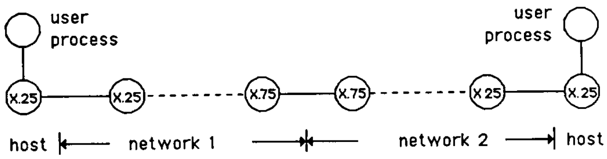


Fig. 2. Basic architecture of X.25/X.75 internetworks.

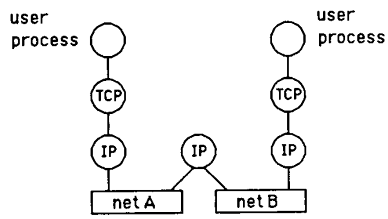


Fig. 3. Basic architecture of TCP/IP internetworks.

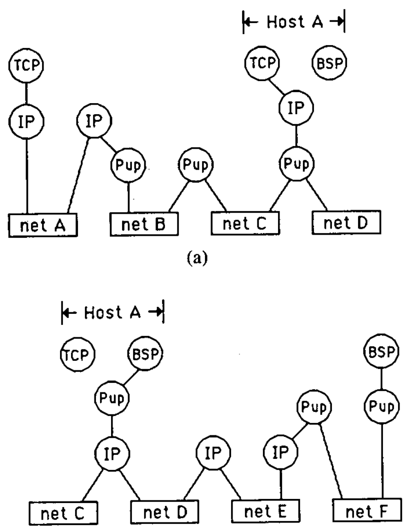


Fig. 4. An illustration of mutual encapsulation. (a) Logical connectivity between two TCP processes. (b) Logical connectivity between two BSP processes.

nected if, and only if, they interoperate, the TCP processes are logically connected and, in each host, the user process interoperates with the TCP process. The TCP processes are logically connected if, and only if, the IP processes in the hosts are logically connected and, in each host, the TCP process interoperates with the IP process. Finally, the IP processes in the hosts are logically connected because there exists an IP process (in a gateway node) to which each host IP process is logically connected. Such logical connectivity between two IP processes requires that each IP process interoperates with each network it is connected to (by some network interface protocol),[2] as shown in Fig. 3; note that logical connectivity between different pairs of IP processes can be provided by networks with very different protocol architectures.

Fig. 4 illustrates the method of mutual encapsulation to achieve logical connectivity in an interconnection of networks where some hosts implement one internetwork protocol, say the IP protocol of Darpa, some hosts implement another internetwork protocol, say the Pup protocol of Xerox, and some hosts implement both internetwork protocols [16]. Fig. 4(a) illustrates how to achieve logical connectivity between two TCP processes by having the Pup processes provide logical connectivity to two of the IP processes. Fig. 4(b) illustrates how to achieve logical connectivity between two BSP processes by having the IP processes provide logical connectivity to two of the Pup processes. Note that at host A, the Pup and IP processes must interoperate with both net C and net D by the respective network interface protocols. The Pup and IP processes must also interoperate with each other by two interface protocols, one for the IP process to access the Pup protocol and the other for the Pup process to access the IP protocol.

Given that two processes, say P_1 and P_n, are connected in a protocol architecture, there are generally additional requirements which must be satisfied by protocols in the physical path between P_1 and P_n in order for P_1 and P_n to communicate. (Logical connectivity is a necessary but not a sufficient condition.) Consider the configuration in Fig. 1. Let P_1 and P_n interoperate by protocol L_0, P_1 and P_2 interoperate by protocol L_1, P_{n-1} and P_n interoperate by protocol L_2, and P_2 and P_{n-1} interoperate by protocol L_3. The protocols L_1, L_2, and L_3 provide a data transfer service to P_1 and P_n.

The service data units of different protocols have size constraints which need to be checked. In order for protocol messages of L_0 to be delivered from P_1 to P_n and from P_n to P_1, each of the protocols, L_1, L_2, and L_3, must provide a data transfer service with sufficiently large service data units. Suppose the service data units of L_3 are too small. Then to achieve communication between P_1 and P_n, two processes will have to be inserted between P_1 and P_2 and between P_{n-1} and P_n to implement a protocol that provides the functions of segmentation and reassembly of messages.

Also, the protocols L_0, L_1, L_2, and L_3 may have dependencies that must be satisfied. For example, if the processes P_1, P_2, P_{n-1} and P_n implement the routing function in the network layer of a protocol architecture, then L_0, L_1, L_2, and L_3 must necessarily be the same routing protocol. Another example: if L_1 is the X.25 protocol then L_2 and L_3 must be either the X.25 or X.75 protocol.

Lastly, given two processes that are logically connected in an architecture, the data transfer service provided to them may not meet the functional specifications that are required for their interoperation. A good example is the

[2]A network can be viewed as a process.

one discussed by Green [5] about the use of an X.25 virtual circuit as a data link in an SNA path. An extra protocol layer was added *on top of* the X.25 layer to provide (among other things) packet sequence numbers that have the same meaning as SNA's link-level SDLC sequence numbers. Green refers to such insertion of an extra protocol layer into a layered protocol architecture as *protocol complementing*. (Adding the IP protocol layer in host and gateway nodes to achieve logical connectivity across a TCP/IP internetwork, as shown in Fig. 3, is also protocol complementing according to Green [5].)

III. Conversion to Achieve Interoperability

Suppose two user processes are physically connected but not logically connected in an architecture. By the definition of logical connectivity, either the two user processes do not interoperate or there exist some processes in the architecture which implement different protocols but which must interoperate to provide logical connectivity between the user processes. In each case, protocol conversion is needed to achieve interoperability between some processes that implement different protocols.

A. *The Correctness Problem*

In what follows, we shall introduce a formal model that can be used for specifying conversions and for reasoning about the correctness of conversions. We consider protocols in which processes interact by exchanging messages. Protocol mismatches in this context refer to differences in the syntax and semantics of messages that are sent and received in different protocols. The conversion problem can be stated as follows. Consider two protocols P and Q (see Fig. 5). In the first protocol, the sets of messages that can be sent by entities P_1 and P_2 are M_1 and M_2, respectively. In the second protocol, the sets of messages that can be sent by entities Q_1 and Q_2 are N_1 and N_2, respectively. Now suppose we want P_1 to interoperate with Q_2 with the help of a protocol converter C_1 as shown in Fig. 6(a). (The converter may be a process or a protocol layer in the path between P_1 and Q_2.) One task of the converter is to perform syntax transformations of messages that P_1 and Q_2 can send to each other. But how does the converter map messages in M_1 and N_2 into N_2 and M_2, respectively? Messages that are related by the mapping have to be semantically equivalent in the two protocols. What does semantic equivalence mean? And how does one check it? Obviously, the level of functionality that can be achieved by the protocol conversion is determined by the sets of semantically equivalent messages.

B. *Protocol Verification and Projection*

The semantics of messages in a message-passing protocol such as P or Q can be found in the reachability graph of the protocol. To avoid the generation of reachability graphs (which may be infinite for many protocols), we propose the use of image protocols for reasoning about semantic equivalence. Before proceeding further with the

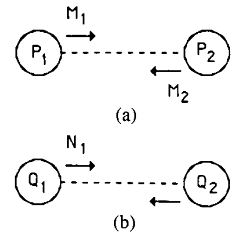


Fig. 5. Protocols P and Q.

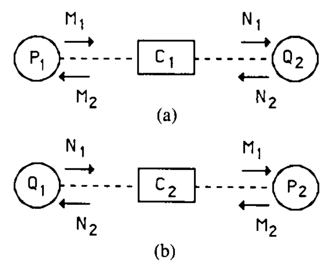


Fig. 6. Protocol conversions.

conversion problem, we shall digress and give an overview of the theory of protocol projection. The following presentation is new and hopefully is more enlightening than that in [8]. The reader is referred to [8] for additional definitions and details.

Our discourse shall be based on the abstract state machines model for protocols. Consider Fig. 5(a). Let S_1 (S_2) denote the set of states of process P_1 (P_2). S_1 and S_2 may be finite or infinite. (Thus our model is applicable to protocols specified by state variables and a programming language notation, as in [14].) Each process is event-driven. Events of P_1 are specified for sending messages in M_1 and for receiving messages in M_2. Additionally, some events not associated with the sending and receiving of messages but whose occurrences cause state transitions in P_1 may also be specified. These are called internal events and they are used to model timeouts and the protocol's interaction with its user processes which are not explicitly modeled. Events of P_2 are similarly defined. If the channels can have errors, then they are modeled as processes; errors are modeled by specifying internal events for these channel processes.

The state of the protocol is described by the four-tuple $(s_1, s_2; \boldsymbol{m}_1, \boldsymbol{m}_2)$ where s_1 is the state of P_1 and $\boldsymbol{m}_1$ is the sequence of messages in the channel from P_1 to P_2; s_2 and $\boldsymbol{m}_2$ are similarly defined. Let G denote the global state space of the protocol. When the protocol is in state g, a set of events are enabled; the occurrence of one of these enabled events, chosen nondeterministically will *take* the protocol to some state h in G. Events are considered to be indivisible. Only one event may occur at a time. Concurrent events may occur in any order.

Given a set G_0 of initial states, define a *path* in G to be a finite or infinite sequence of states and events denoted by

$$\sigma: g_0 \xrightarrow{e_0} g_1 \xrightarrow{e_1} g_2 \xrightarrow{e_2} \cdots$$

where $g_0 \in G_0$, event e_i is enabled in state g_i, and the occurrence of e_i takes the protocol from state g_i to state g_{i+1}. Also, σ is finite and terminates in g_k only if no event is enabled in state g_k. (What we call a path here is often referred to as a computation in the distributed programming literature.)

We denote by $\mathcal{S}(P)$ the set of paths of protocol system P. We have adopted the approach that $\mathcal{S}(P)$ specifies the *semantics of P*. (This approach is espoused by researchers who advocate the use of linear time temporal logic to express properties of time sequences [10], [13].)

So far in this paper, the meaning of "specifying" a protocol for a set of processes to interoperate has been deliberately vague. The existence of such a protocol implies that the processes "understand the meanings of each other's messages." Specifying the message sets of P_1 and P_2 as well as the events of P_1, P_2, and the channels defines operationally the protocol's behavior. It is generally useful to specify a protocol functionally in addition to defining its behavior operationally. The functional specification of a protocol can be in the form of invariant and liveness assertions about the behavior of the protocol. *Invariant assertions* are specified by state formulas. A state formula is a formula written in some first-order language that describes a property of a protocol state; a state formula is evaluated at a single state to yield a truth value. An invariant assertion holds for a protocol if the state formula evaluates to true over all reachable states of the protocol system. *Liveness assertions* are specified by temporal formulas. A temporal formula is a formula constructed from state formulas and one or more temporal operators. A temporal formula is interpreted over individual paths in the set $\mathcal{S}(P)$. A liveness assertion holds for a protocol if the temporal formula is satisfied by every path in $\mathcal{S}(P)$. (See [13] for an excellent treatment of this subject.)

It will not be necessary for us to adopt a language for state formulas or for temporal formulas in this paper. Our objective herein, as well as in [8], is not to specify and verify properties of specific protocols. The set $\mathcal{S}(P)$ will be an adequate vehicle for reasoning about semantic equivalence of different protocol systems. The reader is referred to [10], [15] for some examples of languages for state formulas and temporal formulas.

We are now in a position to introduce the concept of the *resolution* of a protocol that is central to the theory of projections. Consider again the protocol in Fig. 5(a). Since the protocol state is the tuple $(s_1, s_2; \boldsymbol{m}_1, \boldsymbol{m}_2)$, the resolution of the protocol is, roughly speaking, given by the number of states in S_1 and S_2 and the number of messages in M_1 and M_2. Suppose the protocol performs many functions, but we are only interested in verifying an assertion about the protocol's performance of one or a subset of its functions. Then, in the verification, the *observation resolution* of the protocol can be much smaller than the protocol's actual resolution. This gives rise to the idea of constructing an image protocol with a resolution lower than that of the original protocol for verifying the assertion.

Let P' denote an image protocol of P and P_i' denote a process of P' ($i = 1$ and 2). Process P_i' has a set of states S_i' obtained as follows. Partition S_i of process P_i in some fashion. Each partition subset of states in S_i defines a single state in S_i'. (We shall sometimes refer to this operation as *aggregation*.) Given an assertion to be proved, exactly how to do the partitioning of S_1 and S_2 requires ingenuity and insights into the meanings of the process states. If the state of P_i is specified by the values of a set of state variables, then one way to realize a partitioning of S_i is by retaining in the image protocol a subset of the state variables in the original protocol. Generally, the meanings of state variables in protocols specified by a programming language are more self-evident than the meanings of states in state machines. (See [8], [14] for illustrations.)

Aggregating states in S_i to define states in S_i' induces an equivalence relation on the message sets M_1 and M_2. Specifically, two messages in M_1 are equivalent if their receptions cause identical state changes in the image state space S_2'; a similar definition applies to messages in M_2. Furthermore, messages in M_i whose receptions do not cause any state change in the image state space of the receiving process are said to have a *null image*. Let E be the set of all events specified for protocol system P. The aggregation of states in S_i and messages in M_i, for $i = 1$ and 2, also induces an equivalence relation on E. Events that are equivalent in E are also aggregated in order to form the event set E' for the image protocol P'. There are some more definitions and details necessary for defining the following mappings to construct image protocol P':

$$S_i \rightarrow S_i' \qquad i = 1, 2$$

$$M_i \rightarrow M_i' \qquad i = 1, 2$$

$$E \rightarrow E'$$

We refer the reader to [8] for these definitions and details. Elements in the sets S_i', M_i', and E' of the image protocol will be referred to as images of elements in the sets S_i, M_i, and E, respectively, of the original protocol. Note, however, that some elements in M_i and E may be mapped to null images which are not included in the sets M_i' and E'. Lastly, it is important to note that an image protocol is specified like any other protocol, i.e., it can be implemented.

By its very definition, an image protocol has a resolution lower than that of the original protocol. Given an image protocol, suppose that a second image protocol is obtained by partitioning S_1' and S_2' of the first image protocol. Then, the second image protocol has a lower resolution than the first. Thus, we can talk about a sequence of image protocols with decreasing (or increasing) resolution.

For a global state $g = (s_1, s_2, \boldsymbol{m}_1, \boldsymbol{m}_2)$ of protocol P, the image of g is defined to be $g' = (s_1', s_2', \boldsymbol{m}_1', \boldsymbol{m}_2')$ where each process state in g is replaced by its image, and each message in $\boldsymbol{m}_i$ in g is replaced by its image; also, null images are not included in $\boldsymbol{m}_i'$. For a path $\sigma \in \mathcal{S}(P)$, the image σ' of σ is obtained as follows: first, each state in σ is replaced by its image; second, any consecutive occurrences of the same image state in the resulting sequence is replaced by a single occurrence of the image state.

Suppose we replace every path in $\mathcal{S}(P)$ by its image. In particular, all paths having the same image are treated as equivalent and replaced by a single image path. The resulting set of paths is said to be the projection of $\mathcal{S}(P)$ to be denoted by $\mathbf{proj}[\mathcal{S}(P)]$.

$\mathbf{proj}[\mathcal{S}(P)]$ represents the behavior (semantics) of the protocol system P as observed by someone who cannot distinguish between process states, messages, and events having the same images (the observation resolution is lower than that of protocol P). There are fewer properties of protocol P that such an observer can verify, because of the lower resolution in its observations. Specifically, it can only interpret state formulas and temporal formulas that contain references to elements of sets S_i', M_i' and E' rather than elements of S_i, M_i, and E. Such observations are useful as long as the resolution offered by S_i', and M_i', and E' is adequate for the assertion to be verified. Unfortunately such an observer cannot really interpret the state and temporal formulas without knowing $\mathcal{S}(P)$ since $\mathbf{proj}[\mathcal{S}(P)]$ is obtained from $\mathcal{S}(P)$ by definition.

The objective of constructing a separate image protocol system P' is to allow the low resolution observer to interpret its formulas by observing the behavior of the image protocol system, i.e., observing $\mathcal{S}(P')$. Obviously, one would like to employ an image protocol with the lowest resolution possible (yet adequate for proving the given assertion).

Let $R_s(P)$ be the set of reachable states of protocol P. Let $\mathbf{proj}[R_s(P)]$ be obtained by replacing each set of global states in $R_s(P)$ that have the same image by the image state. It is proved that for any image protocol P' constructed as defined in [8],

$$\mathbf{proj}[R_s(P)] \subseteq R_s(P'). \qquad (*)$$

Image Protocol Property 1: If an invariant assertion holds for image protocol P', it also holds for protocol P.

This property is an immediate consequence of result $(*)$. However, keep in mind that the invariant assertion is restricted to state formulas containing references to elements of S_i' and M_i' rather than S_i and M_i.

For the observer to make correct interpretations of temporal formulas (to determine if they are satisfiable by P) by observing the behavior of P', it is necessary and sufficient that

$$\mathbf{proj}[\mathcal{S}(P)] = \mathcal{S}(P'). \qquad (**)$$

The following three conditions (A1)–(A3) are sufficient for condition $(**)$ to hold [8]. (A1) and (A2) are assumptions about the protocol system P.

(A1) Paths in $\mathcal{S}(P)$ satisfy the following fairness assumption: An event enabled infinitely often in path σ will occur infinitely many times in σ [13].

(A2) Each message sent into a channel will eventually be received or deleted.

The following condition is a requirement of the image protocol system P':

(A3) Each event in the image protocol is well-formed.

The definition of a well-formed event in an image protocol is given in [8]. It is useful to note that checking events to be well-formed does not require any knowledge of $\mathcal{S}(P)$ or $\mathcal{S}(P')$. Also, (A3) can always be satisfied by increasing the resolution of the image protocol.

Image Protocol Property 2: Given (A1)–(A3), a liveness assertion holds for image protocol P' if, and only if, it holds for protocol P.

This property is an immediate consequence of $(**)$. Again keep in mind that the liveness assertion is restricted to temporal formulas containing references to elements of S_i', M_i', and E'.

When we infer that protocol P has invariant and temporal properties proved for protocol P', we should interpret the assertions as follows. Each reference to x' in an assertion is interpreted as *some x whose image is x'*, where x denotes a process state, a message, or an event. Since some messages and events have null images which are not included in M_i' and E', an assertion about a sequence $\mathbf{m}'$ of messages should be interpreted as *some sequence* $\mathbf{m}$ *whose image is* $\mathbf{m}'$; a similar interpretation applies to a sequence of event occurrences.

It is beyond the scope of this paper to define an assertion language for protocol verification. Instead of specifying formal rules for interpreting assertions about image protocol P' to describe the behavior of protocol P, we give a few examples for illustration:

1) *event e' eventually occurs* interpreted as *some event e, whose image is e', eventually occurs.*

2) *state g' is reachable* interpreted as *some state g, whose image is g', is reachable.*

3) *the number of messages in the channel is bounded by 3* interpreted as *the number of messages in the channel with nonnull images is bounded by 3.*

4) *event e' takes the protocol from state g' to state h'* interpreted as *some sequence of events* $e_1, e_2, \ldots, e_n$, *whose image is e', takes the protocol from some state g, whose image is g', to some state h, whose image is h'.*

C. Memoryless Converters

Consider a protocol that is an image protocol of protocol P and also an image protocol of protocol Q. Such a protocol is called a *common image protocol* of P and Q. We can now state a simple approach to solving the protocol conversion problem formulated earlier: find a common image protocol of P and Q having the highest resolution. (By contrast, in protocol verification, we desire an image protocol with the lowest resolution adequate for proving an assertion.) Suppose such an image protocol common to both protocols P and Q is found. Let us consider the protocol conversion in Fig. 6(a). What C_1 provides is a mapping function. A message sent by P_1 with a nonnull image, say m' in M_1', is transformed by C_1 into a message in N_1 with image m' for delivery to Q_2; similarly, messages in N_2 are mapped into M_2. What this conversion accomplishes is an implementation of the image protocol. If this image protocol is one common to both P and Q with the highest resolution, then it implements the most functionality that is common to both P and Q.

Since image protocols P' and Q' are identical, we have

$$\mathbf{proj}[R_s(P)] \subseteq R_s(P') = R_s(Q')$$
$$\mathbf{proj}[R_s(Q)] \subseteq R_s(P') = R_s(Q')$$

By image protocol property 1, invariant properties of P' $(= Q')$ are also invariant properties of P and of Q. If all events in both P' and Q' are well-formed, then we have

$$\mathbf{proj}[\mathcal{S}(P)] = \mathcal{S}(P') = \mathcal{S}(Q') = \mathbf{proj}[\mathcal{S}(Q)].$$

By image protocol property 2, liveness properties of P' $(= Q')$ are also liveness properties of P and of Q.

Thus the correctness of the conversion is well-defined. It is also a meaningful and rigorous definition of correctness. The advantage of this approach is that it is possible to establish semantic equivalence without having to generate any of the reachability graphs.

This approach requires a heuristic search for an image protocol with useful properties. Note that an image protocol common to both P and Q can always be found. Specifically, if we aggregate the set of states in each process in Fig. 5 to a single state, we have an image protocol common to P and Q. But it is an image protocol with no useful property. Any difficulty in the heuristic search, however, may not necessarily be the fault of the method; it could be due to the fact that the protocols P and Q have very little in common to begin with. Obviously, the job of synthesizing a conversion will be easier if protocols P and Q are quite similar to each other, such as, they are variants of the same protocol.

Consider those messages in M_1 (N_2) with a null image in the common image protocol. It is not necessary for P_1 (Q_2) to be aware that it is interacting with a partner implementing a different protocol and that null-image messages should not be sent. The conversion can be made *transparent* to P_1 (Q_2) by having the converter intercept null-image messages sent by P_1 (Q_2) and simply discard any such message received.

D. Finite-State Converters

Suppose processes P_1 and Q_2 interoperate via a memoryless converter. The common image protocol, however, may not have enough functionality for a particular application. One way to add functionality to the protocol in Fig. 6(a) is to add a state machine in C_1. An example we have tried is that of a conversion between a version of IBM's BSC protocol and an alternating-bit (AB) protocol [3]. BSC has the same basic structure as AB but differs from it in a number of details. In particular, BSC data messages do not carry a sequence number (0 or 1). But in an AB receiver, a sequence number is expected in each data message received. The common image protocol will not have a sequence number in its data messages. As a result the common image protocol does not have the desired logical property of the AB protocol. This shortcoming can be remedied by having a state machine in the converter that inserts a sequence number into each message that it sends to the AB receiver. In this case, the set of messages sent by the converter has a higher resolution than the set of messages it receives.

Consider the protocol system in Fig. 6(a) consisting of P_1, C_1, and Q_2, the channels between P_1 and C_1, and the channels between C_1 and Q_2. We shall refer to this protocol system as $C = (P_1, C_1, Q_2) = ((P_1, C_1), Q_2) = (P_1, (C_1, Q_2))$.

Consider Fig. 7. Let Q_c denote the network processes inside the rectangle in Fig. 7(a); the network Q_c can be viewed as a single process that is interacting with Q_2. A state of Q_c is defined by the tuple $(s_1, s_2, \boldsymbol{m}_1, \boldsymbol{m}_2)$ where s_1 is the state of P_1, s_2 is the state of C_1, and $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ represent the sequences of messages in the channels from P_1 to C_1 and C_1 to P_1, respectively. Events for sending messages in N_1 and events for receiving messages in N_2 that are defined for C_1, define send and receive events of Q_c. Internal events of P_1 and C_1 define internal events of Q_c. Furthermore, send and receive events of P_1 and C_1 associated with messages in M_1 and M_2 define *internal* events of Q_c.

Similarly, the network P_c in Fig. 7(b) can be viewed as a single process interacting with P_1.

Next we address the correctness problem of a finite-state converter. Suppose an image protocol C' of the (Q_c, Q_2) network in Fig. 7(a) has been found and is identical to an image protocol Q' of protocol Q. Then we have

$$\mathbf{proj}[R_s(C)] \subseteq R_s(Q') = R_s(C')$$
$$\mathbf{proj}[R_s(Q)] \subseteq R_s(Q') = R_s(C')$$

Thus an invariant property of Q' $(= C')$ is also an invariant property of Q and of C. If both C' and Q' have well-formed events, then

$$\mathbf{proj}[\mathcal{S}(Q)] = \mathcal{S}(Q') = \mathcal{S}(C') = \mathbf{proj}[\mathcal{S}(C)].$$

And a liveness property of Q' $(= C')$ is also a liveness property of C and of Q.

Next, suppose an image protocol C'' of the (P_1, P_c) network in Fig. 7(b) has been found and it is identical to an image protocol P' of protocol P. Correctness results analogous to those for C' and Q' can now be stated for C'' and P'.

Since P' and Q' are in general different, the external users of the protocol (P_1, C_1, Q_2) may not "see" the same protocol service (as in the case of a memoryless converter). The external user connected to P_1 sees the pro-

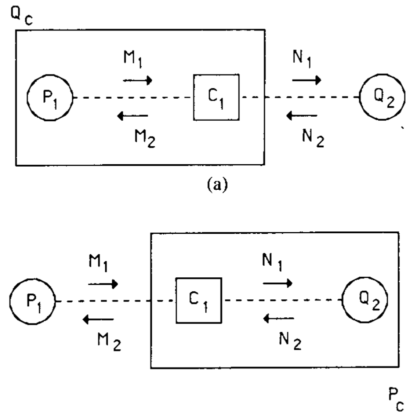


Fig. 7. Two views of the protocol system $C = (P_1, C_1, Q_2)$.

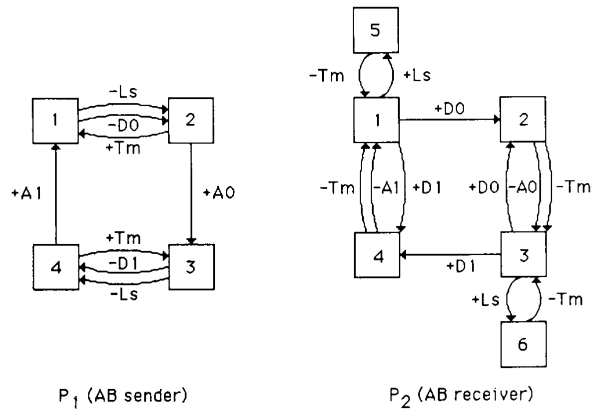


Fig. 8. The alternating-bit (AB) protocol.

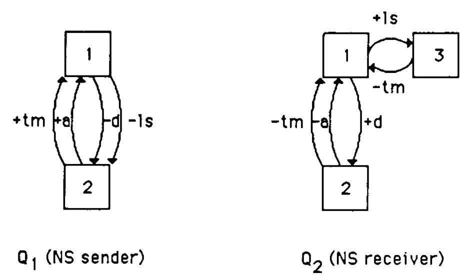


Fig. 9. The nonsequenced (NS) protocol.

tocol service of P' while the external user connected to Q_2 sees the protocol service of Q'. (See example below for an elaboration of this observation.)

E. Examples of Finite-State Converters

Consider the protocols for data transfer in Figs. 8 and 9. The alternating-bit (AB) protocol is shown in Fig. 8. The protocol in Fig. 9 does not employ any sequence numbers and is referred to as the nonsequenced (NS) protocol. In Figs. 8 and 9, a transition labeled with a plus sign denotes receiving a message from the incoming channel. A transition labeled with a minus sign denotes sending a message into the outgoing channel. In Fig. 8, $A0$ and $A1$ denote ack messages and $D0$ and $D1$ denote data messages, with sequence numbers 0 and 1, respectively. In Fig. 9, a denotes an ack message while d denotes a data message.

The channels in the protocol model are assumed to be FIFO queues. Message loss and timeout events are modeled by the addition of some state transitions associated with virtual messages (*Tm* and *Ls* in the AB protocol and *tm* and *ls* in the NS protocol denoting "timeout" and "loss," respectively). Possible loss of a data message is modeled by specifying a pair of transitions *-data* and *-loss* in parallel. Possible loss of an ack message is modeled by specifying a pair of transitions *-ack* and *-timeout* in parallel. The event *+timeout* denotes a timeout occurrence; note that premature timeout occurrences are not allowed by this model, i.e., a timeout occurs only if either a data message or an ack message has been lost.

Initially, each process is in state 1 and all channels are empty. The reachability graphs of both protocols are relatively small. In fact, for both protocols the number of messages in all channels is bounded by 1. The AB protocol provides FIFO delivery of data with no loss and no duplication. However, when a timeout occurs in an NS sender, it does not know whether a data message was lost or its ack was lost. If the NS sender always retransmits old data whenever a timeout occurs, then the NS receiver will provide FIFO delivery of data with no loss but it will sometimes deliver the same data more than once to its user.

A converter process C_1 for P_1 and Q_2 is shown in Fig. 10. Note that messages in the AB and NS protocols have distinct names (NS message names are in lower-case characters only while AB message names employ upper-case characters.) Therefore we do not have to label the sender or the receiver of each message in Fig. 10.

To show that C_1 is correct and to determine what services are provided by the network (P_1, C_1, Q_2) to the users of P_1 and Q_2, we need to construct the processes P_c and Q_c as described in Section III-D. Before doing so, observe that every one of the finite-state machines, P_1, C_1, and Q_2, has only two kinds of nodes; sending nodes in which only send events can occur, and receiving nodes in which only receive events can occur. Given that each machine is initially in state 1, it is easy to show that the network (P_1, C_1, Q_2) has the following invariant property:

1) one machine in the network is in a sending node and all other machines are in receiving nodes and all channels are empty, or

2) one channel has exactly one message and all other channels are empty and all machines are in receiving nodes.

Because of this invariant property, finite-state machines for Q_c and P_c can be constructed very easily. We have constructed them but have omitted them here for the sake of brevity.

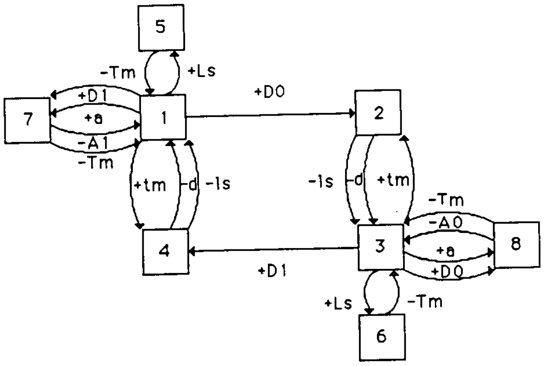


Fig. 10. Converter C_1 for P_1 and Q_2.

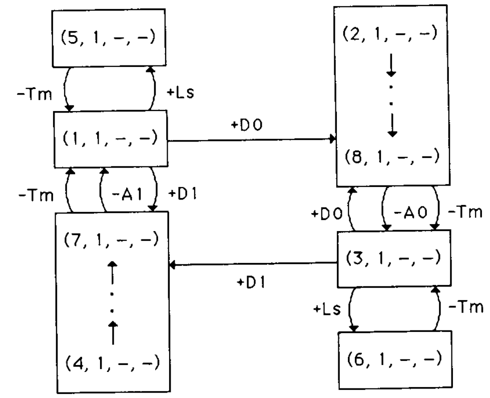


Fig. 11. Image of P_c = (C_1, Q_2).

We found that the protocol $Q = (Q_1, Q_2)$ is itself an image protocol, with well-formed events, of (Q_c, Q_2). Further, the protocol $P = (P_1, P_2)$ is also an image protocol, with well-formed events, of (P_1, P_c). In Fig. 11, we have shown P_c with its states partitioned so that when each partition subset is aggregated, the image of P_c is identical to P_2. The details of partition subsets with multiple states have been omitted. Each tuple $(s_1, s_2, \boldsymbol{m}_1, \boldsymbol{m}_2)$ in Fig. 11 denotes a state of P_c, where s_1 denotes the state of C_1, s_2 denotes the state of Q_2, $\boldsymbol{m}_1$ denotes the channel from C_1 to Q_2, and $\boldsymbol{m}_2$ denotes the channel from Q_2 to C_1. The notation "-" indicates an empty channel.

Note that the conversion is transparent to both P_1 and Q_2, such that P_1 thinks that it is interacting with P_2 while Q_2 thinks that it is interacting with Q_1. Is P_1 really providing the service of the AB protocol to its user, i.e., FIFO delivery of data with no loss and no duplication? How can this be true, when the user connected to Q_2 is only getting the service of the NS protocol, i.e., FIFO delivery of data with no loss but possible duplication?

The explanation for this apparent contradiction is as follows. The AB protocol service assumes reliable channels between P_2 and its user. In the $(P_1, P_c) = (P_1, (C_1, Q_2))$ network, however, this assumption is no longer true. When an ack from Q_2 to C_1 is lost, C_1 retransmits a duplicate data message to Q_2. These are internal event occurrences in P_c which are not observable at the resolution of P_1 and P_2.

In this example, there is an easy way to improve the service provided by (P_1, C_1, Q_2). Note that the NS protocol will provide FIFO delivery of data with no loss and no duplication if the acks sent by Q_2 are never lost (reliable outgoing channel from Q_2). This can be accomplished by placing converter C_1 in the same node as Q_2 so that they interact via some reliable interprocess communication facility instead of across unreliable communication lines.

In Fig. 12, we show a converter C_2 for Q_1 (NS sender) and P_2 (AB receiver). It is also easy to show that the protocol $Q = (Q_1, Q_2)$ is itself an image protocol, with well-formed events, of $(Q_1, (C_2, P_2))$, and the protocol $P = (P_1, P_2)$ is itself an image protocol, with well-formed events, of $((Q_1, C_2), P_2)$.

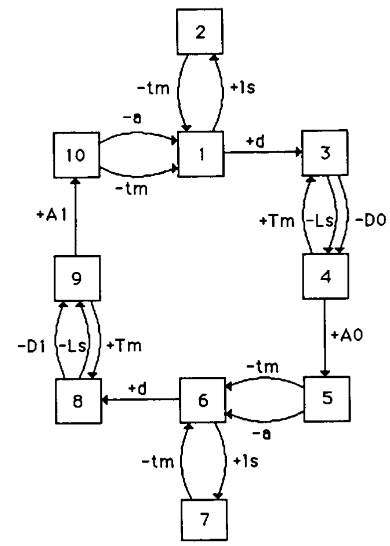


Fig. 12. Converter C_2 for Q_1 and P_2.

IV. Concluding Remarks

We have addressed two distinct aspects of the protocol conversion problem. First, we gave a definition of logical connectivity between processes in a protocol architecture. Logical connectivity is a necessary condition for conversion-free communication between processes across a network or an internetwork.

We then addressed the problem of achieving interoperability between processes that implement different protocols by means of a protocol converter. We presented a formal model, based upon the theory of protocol projection, for specifying conversions and for reasoning about the correctness of conversions. The construction of finite-state converters was illustrated with an example involving processes that implement the alternating-bit protocol and a nonsequenced protocol for data transfer.

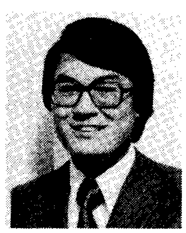
It is worthwhile noting that protocol conversion to achieve interoperability is different from protocol complementing (discussed in Section II), which is another kind of protocol conversion [5]. A protocol converter to achieve interoperability between two processes is implemented either as a process or as a lower-layer protocol in the physical path between the processes. Protocol complementing on the other hand, inserts an extra protocol layer *on top of* the processes that require "complementing"; in this case, the users of the complemented processes must subsequently interface with the inserted protocol layer. In conversions to achieve interoperability, the user interfaces of the processes requiring conversion are not affected. The two approaches to protocol conversion achieve different objectives and both approaches will find useful applications in internetworking environments.

Acknowledgment

This paper has benefited greatly from the constructive comments of the anonymous reviewers. Comments received from P. Green of IBM Research, Yorktown Heights, NY, and K. Calvert and M. Gouda of the University of Texas at Austin are also gratefully acknowledged.

References

[1] *ANSI/IEEE Standards for Local Area Networks*, IEEE Standards 802.2, 802.3, 802.4, 802.5, 1984.
[2] "Draft revised CCITT recommendation X.25," *ACM Comput. Commun. Rev.*, Jan./Apr. 1980.
[3] K. Calvert and S. S. Lam, "An exercise in deriving a protocol conversion," in *Proc. ACM SIGCOMM Workshop*, Stowe, VT, Aug. 1987.
[4] "DoD standard internet protocol and DoD standard transmission control protocol," *ACM Comput. Commun. Rev.*, Oct. 1980.
[5] P. Green, "Protocol conversion," *IEEE Trans. Commun.*, vol. COM-34, no. 3, pp. 257-268, Mar. 1986.
[6] I. Groenback, "Conversion between TCP and ISO transport protocols as a method of achieving interoperability between data communications systems," *IEEE J. Select. Areas Commun.*, vol. SAC-4, no. 2, pp. 288-296, Mar. 1986.
[7] S. S. Lam, "Data link control procedures," in *Computer Communications*, vol. 1, W. Chou, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1983, pp. 81-113.
[8] S. S. Lam and A. U. Shankar, "Protocol verification via projections," *IEEE Trans. Software Eng.*, vol. SE-10, no. 4, pp. 325-342, July 1984.
[9] S. S. Lam, "Protocol conversion—correctness problems," in *Proc. ACM Sigcomm '86 Symp.*, Stowe, VT, Aug. 1986, pp. 19-29.
[10] Z. Manna and A. Pnueli, "Adequate proof principles for invariance and liveness properties of concurrent programs," *Sci. Comput. Program.*, vol. 4, pp. 257-289, 1984.
[11] J. E. McNamara, *Technical Aspects of Data Communications*. Maynard, MA: Digital Equipment Corp., 1977.
[12] K. Okumura, "A formal protocol conversion method," in *Proc. ACM SIGCOMM '86 Symp.*, Stowe, VT, Aug. 1986, pp. 30-37.
[13] A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends," in *Current Trends in Concurrency: Overview and Tutorials* (Lecture Notes in Computer Science, vol. 224), J. W. deBakker *et al.*, Eds. New York: Springer-Verlag, 1986, pp. 510-584.
[14] A. U. Shankar and S. S. Lam, "An HDLC protocol specification and its verification using image protocols," *ACM Trans. Comput. Syst.*, vol. 1, no. 4, pp. 331-368, Nov. 1983.
[15] —, "Time-dependent distributed systems: Proving safety, liveness, and real-time properties," *Distributed Comput.*, vol. 2, pp. 61-79, 1987.
[16] J. F. Shoch, D. Cohen, and E. A. Taft, "Mutual encapsulation of internetwork protocols," *Comput. Networks*, pp. 287-300, 1981.
[17] M. S. Unsoy and T. A. Shanahan, "X.75 internetworking of Datapac and Telenet," in *Proc. 7th Data Commun. Symp.*, 1981, pp. 232-239.
[18] H. Zimmerman, "OSI reference model-The ISO model of architecture for open systems interconnection," *IEEE Trans. Commun.*, vol. COM-28, no. 4, pp. 425-432, Apr. 1980.



Simon S. Lam (S'69-M'74-SM'80-F'85) received the B.S.E.E. degree (with Distinction) from Washington State University, Pullman, in 1969, and the M.S. and Ph.D. degrees in engineering from the University of California at Los Angeles in 1970 and 1974, respectively.

From 1974 to 1977, he was a Research Staff member at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. Since September 1977, he has been on the faculty of the University of Texas at Austin, where he is a Professor of Computer Sciences and holds the Second David Bruton Jr. Centennial Professorship. His research interests are in the areas of computer networks, communication protocols, performance modeling, and the specification and verification of distributed systems.

Dr. Lam received the 1975 Leonard G. Abraham Prize Paper Award from the IEEE Communications Society. He currently serves on the editorial boards of three journals, IEEE Transactions on Communications, *Performance Evaluation*, and Proceedings of the IEEE.