An Exercise in Deriving a Protocol Conversion*

Kenneth L. Calvert
Simon S. Lam

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Abstract

This paper demonstrates formal techniques useful in solving and reasoning about protocol conversion problems. A simple example problem is solved and the resulting conversion system is shown to have certain desired properties, using the *projection paradigm.* The example problem is representative of some real-world problems in that the protocols involved are similar in function, and even in structure, but have fundamental differences that render them incompatible in the absence of an active translation entity (protocol converter). The use of mappings, as well as images and inverse images of properties, in comparing semantics of protocols is discussed and illustrated.

1 Introduction

A wide variety of formal methods for verifying and modeling protocols have been developed [8]. However, these methods generally are intended to deal with the correctness of individual protocols, more or less in isolation. The problem of enabling productive interoperation among entities that were *not* designed to operate together, i.e., the *protocol conversion* problem, has recently begun to receive some attention. Several recent papers have dealt with protocol conversion from various perspectives [1,2,4,7]; there is general agreement that protocol conversion is and will remain an important problem, and that formal methods for reasoning about the problem are a necessity.

In the general protocol conversion problem, two *processes*, A_1 and A_2, perform some useful function in a network environment by exchanging messages over channels according to a protocol, A; while two other processes, B_1 and B_2, have been designed to accomplish a similar function using a different protocol, B (Figure 1). The classical example of such a function is ensuring reliable transmission of data messages over unreliable channels.

Now, suppose we need to use A_1 and B_2 (or B_1 and A_2) to provide some or all of the service normally implemented by protocols A and B. To accomplish this we must consider *what* services are implemented by each protocol, and *how* the processes interoperate in each case to provide those services. In other words, we must understand the *semantics* of each of the protocols. Then we must relate the semantics of the two in order to understand what service can reasonably be provided by the conversion system, and use that relationship to specify a transformation between the *syntaxes* of the protocols. We can then implement this transformation via an intermediary, a *protocol converter* (Figure 3). The converter maps messages or sequences of messages from one protocol into messages or sequences of messages in the other protocol.

We consider the processes in isolation, as they are depicted in Figure 2. This is an abstraction from the general case, in which A and B may be part of some layered architecture; the channels connecting A_1 and B_2 in the figure imply the existence of any conversion necessary at lower levels to be able to provide a transmission path between the processes.

In this paper, we illustrate the approach described in [4,5] for reasoning about conversion systems and their correctness. The approach makes use of *protocol projection* [3], an abstraction technique for verifying properties of complex protocols. The basis of the projection paradigm is the idea that a property of a complex system can be proved by finding a property-preserving transformation to a simpler system, and then proving the property of the simpler system. The rest of the paper is organized as follows. The next section explains the model of protocol systems used for the examples, and briefly describes protocol projection. Then we introduce our example problem, a conversion between the Alternating Bit protocol and a version of the Bisync protocol. A converter is constructed, and the use of protocol projection to prove that the conversion system has certain properties is demonstrated. We conclude with some discussion about the general applicability of these ideas.

*work supported by the National Science Foundation under grant no. ECS-8304734 and under grant no. NCR-8613338

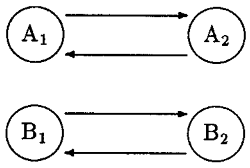

Figure 1: Protocols A and B

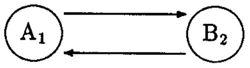

Figure 2: Conversion Configuration

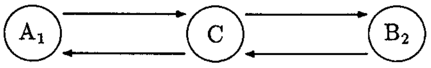

Figure 3: Conversion between A and B

2 Background

We have stated that we wish to use A_1 and B_2 to provide "some or all" of the function of the original protocols, and that the conversion system is to be proved correct somehow. Intuitively, we want the conversion system involving A_1, B_2, and possibly a converter process, to have certain *properties*, somehow related to those of the protocols A and B. In order to reason formally about the conversion, we must have a formalism for expressing such properties. In this section, we describe a simple model for protocol systems and its semantics. Then we discuss methods of relating the semantics of one protocol to those of another, including the method of protocol projection.

The Model

The protocols of our example problem are modelled using a simple *communicating finite-state machines* (cfsm) model. In this model a protocol consists of a set of *processes* connected by *channels*. Each process is defined by a finite *process state space*, a finite set of *events*, and an *initial state*, and may be represented conceptually as a directed graph with nodes corresponding to process states and labeled arcs corresponding to events. The processes of the protocols for the example are depicted using this representation in Figures 4 and 5. A channel is a means of transmission of *messages* from one process to another, and is represented by an infinite-capacity FIFO queue, to which one process may add messages while another process may remove enqueued messages from the other end. Thus communication in this model is completely asynchronous.

The *events* represent activity, or change in the state of the system. In our model, every event involves exactly one process, and directly affects only the state of that process and/or the channels connected to it. Events are considered to be indivisible: only one event may occur at a time, and events considered concurrent may occur in any order. A process event may be one of three types: a send event (represented by a label "–m"), involving the addition of the message m to the tail of a channel and a change in the state of the process; a receive event, (label "+n"), involving removal of a particular message n from the head of a channel and a state change, or an *internal* event, which involves a process state change but no messages; an internal event can be an abstraction of interactions with other entities that are not modeled. In the text, we denote events by triples: an event that changes the state of the process from a to b, and removes the message m from a channel coming from process P, is denoted $(a, b, +\mathrm{m}/\mathrm{P})$. When there is only one incoming and outgoing channel per process, the channel designation is omitted: $(a, b, +\mathrm{m})$.

The state of a channel is the ordered sequence of messages sent on the channel but not yet received on it. The set of possible states of a channel is determined by the set of all messages that may be sent on it, and is the set of all finite sequences of messages in that set. If the set of messages that may be sent on a channel is M, the set of all finite sequences of messages in M (i.e., the channel state space) is denoted by M^*.

The global state of the protocol is defined by the states of all the processes and channels; the global state space of protocol A is denoted A and is the cartesian product of the state spaces of all the processes and channels. For example, if protocol A has two processes with state spaces S_1 and S_2, and two channels with message sets M_1 and M_2, its global state space is defined by:

$$S_1 \times S_2 \times M_1^* \times M_2^*$$

and a global state $g \in A$ may be specified by a 4-tuple:

$$g = [s_1, s_2, U_1, U_2],$$

where

$$s_1 \in S_1, s_2 \in S_2, U_1 \in M_1^*, U_2 \in M_2^*.$$

The *initial global state* is defined as the global state in which all processes are in their initial process states and the channels are empty.

A receive event $(a, b, +\mathrm{m}/\mathrm{P})$ of a process is said to be *enabled* in any global state in which the state of the process is a and the head message of the channel from P is m. An internal or send event (a, b, e) of a process is enabled in any global state in which the state of that process is a. Thus, an event defines a set of transitions between global states; we say an event e *takes* the system from global state g to global state h, and write $g \xrightarrow{e} h$, if e is enabled in g and the state changes associated with e change g to h. For example, the event $(a, b, \text{-m})$ of process 1 in protocol A takes the state $[a, x, U, V]$ to state $[b, x, Um, V]$ (where "Um" represents the appending of message m to sequence U), for any x, U, and V.

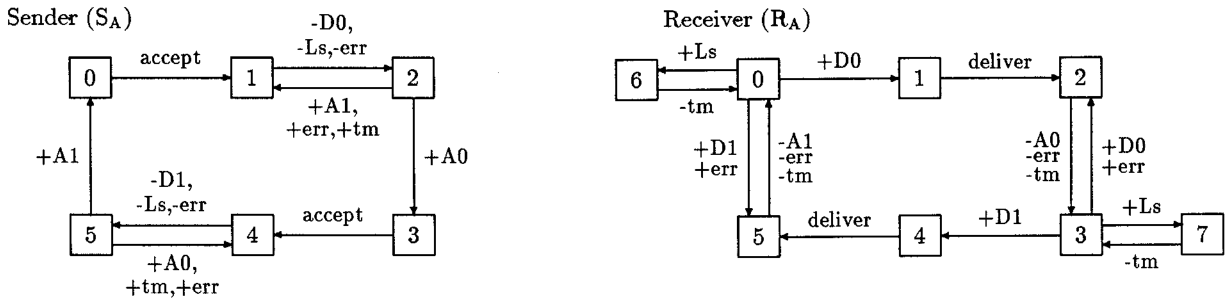

Figure 4: Alternating-Bit Protocol

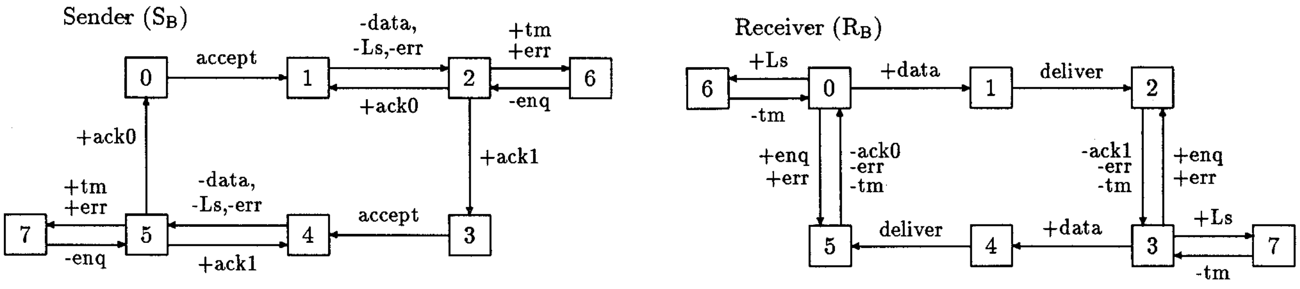

Figure 5: A version of Bisync Protocol

The behavior of the protocol over time is represented by a sequence of global states $g_0, g_1, g_2, \ldots$, such that each pair of successive states in the sequence are related by some event; that is, there exist $e_0, e_1, \ldots$ such that

$$g_0 \xrightarrow{e_0} g_1 \xrightarrow{e_1} g_2 \xrightarrow{e_2} \ldots$$

We call such a (finite or infinite) sequence of states (which may also be considered a sequence of events) a *path.* We say a path is a *computation* if and only if (i) g_0 is the initial global state and (ii) either the path is infinite or no event is enabled in its terminal state. Note that at any state in a sequence there may be several events enabled, while only one occurs to take the system to the next state.

We say a path is *fair* if and only if no event that is enabled infinitely many times in the path occurs only a finite number of times. (This definition is sometimes called "strong fairness," and reflects one criterion for deciding whether an infinite behavior is in some sense "realistic." There are others — fairness is a topic of study in itself.) Note that all finite paths are fair. It can be shown that *any finite path from the initial state can be extended to a fair computation.* That is, if there is a sequence of states and events $g_0 \xrightarrow{e_0} \ldots \xrightarrow{e_{k-1}} g_k$, where g_0 is the initial state, then there is a fair computation, the first $k+1$ states of which are $g_0, \ldots, g_k$. This fair computation may be finite if and only if no event is enabled in its final state. (The proof, which we omit in the interest of brevity, involves showing that there is a path from any global state in which the number of times any event may be enabled between occurrences is bounded.)

The set of all fair computations of a protocol is denoted R. (When we refer to a particular protocol A we write R_A.) The set R embodies the *semantics* of the protocol in the following sense: any possible behavior of the system is represented by some computation in R.

Properties of Protocols

In the model we have just described, the semantics of a protocol is defined *operationally*, by specifying a set of transitions in the global state space that determine the set R of fair computations. The behavior of a system may also be described *functionally* by characterizing the set R by means of logical assertions. We now introduce a language for simple logical assertions about the set R; such assertions can be used to specify certain basic *properties* of a protocol. A full system for stating and proving assertions about the set R is beyond the scope of this paper; we define just what is necessary to specify the properties relevant to our example protocols and the conversion. (The language introduced here is essentially a small piece of *temporal logic*; for a full treatment, see [6].) Our approach to the example will allow us to avoid proving properties of the conversion system from its operational definition. Instead, we find a relation between the conversion system and the original protocols such that it has properties corresponding to their (known) properties.

For any protocol A, a *global state predicate* is a mapping of the global state space A into the set $\{true, false\}$.

We say "p holds at g" or "p is true for g", if and only if $p(g) = true$. A predicate may be considered equivalent to a particular subset of A, namely the set of all states g for which $p(g) = true$. Conversely, any set of states in A may be considered as a predicate; the predicate is true for a state if and only if the state is in the set. In what follows, we use the concepts "subset of A" and "predicate on A" interchangeably.

The expression "P **at** s," where P is a process of protocol A and s is a process state of P, denotes a predicate that is true for all global states in which the state of P is s. For example, if we represent a global state of protocol A (as in Figure 1) by $[a_1, a_2, U_1, U_2]$, then the set of states denoted by "A_1 **at** s" contains exactly those states in which $a_1 = s$. If S is a *set* of process states of P, "P **at** S" denotes the set of global states in which process P is at one of the process states in S. Because the process P is at *exactly one* process state in every global state, if S is the entire process state space of P, then P **at** $S = true$.

If the expressions p and q denote sets on A, then $p \wedge q$ denotes their intersection, and $p \vee q$ denotes their union. The expression $\neg p$ denotes the set complement of the set denoted by p, so $\neg p(g) = true$ if and only if $p(g) = false$. Thus, a boolean combination of **at** expressions defines a predicate. Two such expressions are equivalent if and only if they define the same predicate. From the above definitions, it follows that "P **at** $x \vee P$ **at** y" is equivalent to "P **at** $\{x, y\}$."

We define two types of properties of protocols. A *safety property* expresses the fact that "the system is always in a good state," where "goodness" is defined by a predicate. If p is an expression defining a state predicate, we define the assertion "p is a safety property of A" to be equivalent to "for every state g in every computation in R_A, $p(g) = true$." In other words, every state of every fair computation of A is in the set denoted by p. Since every fair computation is a path from the initial state, if p holds at every state of every path from the initial state of A, then p is a safety property. Also, since any finite path from the initial state can be extended to a fair computation, if $\neg p$ holds at some state on some path from the initial state, then p cannot be a safety property. Thus, "p is a safety property of A" is also equivalent to "p holds at every state of every path from the initial state of A."

A liveness property is an assertion that "something good will eventually happen" in any possible behavior of the system. If p and q are predicates, then "$p \leadsto q$ (read 'p leads to q') is a liveness property of A" is equivalent to "in any fair computation in R_A, every state at which p holds is followed by some state at which q holds." This expresses the characteristic that, at any point in a computation, if p is true, q will eventually become true.

Mappings and Images

In this section, we consider how the properties of one protocol may be related to those of another by relating the global state spaces of the two protocols through a *functional mapping*; such a mapping associates each global state of one protocol with a unique global state of the other. Such a mapping may be considered to define a semantic equivalence between a global state and the state to which it maps.

Suppose we have two protocol systems, A and B, defined operationally as described above. Let A represent the global state space of A, and B the global state space of B. Now consider any functional mapping, $\mathrm{F} : A \mapsto B$. For any global state g in A, $\mathrm{F}(g)$ is a (unique) global state in B, called the *image of g under the mapping* F. (Note that F may be many-to-one, so that we may have $\mathrm{F}(g) = \mathrm{F}(h)$ for some $g \neq h$.) The image of a subset $p \subseteq A$ under the mapping is the set of all the $\mathrm{F}(g)$'s such that $g \in p$. In what follows, the image of a quantity x under the mapping F will be denoted by x'. So for example $g' = \mathrm{F}(g)$ and $p' = \{g' : g \in p\}$.

If t is a state in B, the *inverse image* of t is the *set* of states g in A such that $\mathrm{F}(g) = t$. We denote the inverse image of a quantity x by $\hat{x}$. The inverse image of a subset q of B is a subset $\hat{q} = \{x : x' \in q\}$. From these definitions and the fact that each $g \in A$ has a unique image, it follows that the *inverse image* of the intersection of two sets in B is the same as the intersection of their inverse images:

$$\begin{aligned}
\widehat{(p \cap q)} &= \{g : g' \in p \cap q\} \\
&= \{g : g' \in p \wedge g' \in q\} \\
&= \{g : g' \in p\} \cap \{g : g' \in q\} \\
&= (\hat{p} \cap \hat{q})
\end{aligned}$$

Similar properties hold for union and complementation: the inverse image of $p \cup q$ is $\hat{p} \cup \hat{q}$, and $\widehat{\neg p} = \neg \hat{p}$.

The image of a computation $g_0 \xrightarrow{e_0} g_1 \xrightarrow{e_1} \ldots$ is defined to be the sequence of image states $g'_0, g'_1, \ldots$, with adjacent occurrences of the same image state consolidated into one image state, so that the same state does not occur twice in succession. Thus the image of an infinite computation of protocol A might be a finite sequence of states in B, if all states beyond a certain point in the computation have the same image under the mapping F. Note that the image of a computation of A is a sequence that is *not necessarily a computation* of B, because there might be some point in the image sequence where for no event e of B is it the case that $g'_k \xrightarrow{e} g'_{k+1}$; or, the image of the computation might be finite, with some event enabled in the terminal state.

The inverse image of a computation of B (or any sequence of global states in B) is the set of sequences of states having that computation as image. That is, if α is any sequence of global states of B, $\alpha = \alpha_0, \alpha_1, \ldots$, the inverse image of α, denoted by $\hat{\alpha}$, is the set of sequences of states, given by (with "$\times$" denoting cartesian product)

$$\hat{\alpha} = \hat{\alpha}_0 \times \hat{\alpha}_1 \times \ldots = \{\sigma : \sigma' = \alpha\}.$$

A sequence of states of A in the inverse image of a computation of B is not necessarily a computation.

Now, suppose we want to relate the properties of some protocol B to another protocol A; we would like to find assertions about A that correspond to the assertions of properties of B. In our simple assertion language, properties are defined by **at** expressions relating the states of the processes in the protocol. If we want to obtain assertions about A from assertions about B, we must define a correspondence between the process states of A and those of B. *Protocol projection* is one way of defining such a correspondence.

Protocol Projection

Protocol projection [3] is an abstraction technique that can be used to prove properties of a complex protocol by mapping it onto a simpler *image protocol.* The image protocol and the mapping are both derived from the given protocol. While projection is applicable to distributed systems with more general structure, we shall describe it in terms of the simple cfsm model used for the example. The following discussion is necessarily brief; for a complete discussion the reader is referred to [3].

Given a protocol A, a protocol projection is defined by partitioning the state spaces of each of A's processes. The idea is that process states which are to be functionally indistinguishable in the image protocol are *aggregated* into the same partition, and map to the same *image process state.* The (unique) image of any process state a is defined by this state aggregation, and is denoted by a'. The set of process states corresponding to the image process state b is denoted by $\hat{b}$. Because of the structure of a protocol, a partitioning of the process state spaces defines a corresponding partitioning of the message and event sets of the protocol. Every message (and event) maps to a message (event) in the image protocol or has a *null image.* The messages and events of the image protocol are exactly those that are the image of some message or event in the original protocol.

The image of a message is determined by the effect of its reception in the image of the receiving process' state space: messages that cause the same state changes have the same image, while any message that does not cause any state change in its receiver is said to have a null image. For example, suppose a, b, c and d are process states of some process, and m and n are messages that may be received by the process. If $(a, b, +\mathrm{m})$ and $(c, d, +\mathrm{m})$ are the only receive events involving the message m, and $a' = b'$ and $c' = d'$, then the message m has a *null image.* If $a' = c'$ and $b' = d'$ for any two events $(a, b, +\mathrm{m})$ and $(c, d, +\mathrm{n})$, then m and n have the same image. In the former case, the projection has abstracted away completely the function served by the message m, while in the latter case, the functional distinction between m and n has been removed.

The image of a process event (a, b, e) is determined by the images of a, b, and any messages involved in e. If $a' = b'$, and e is the label of an internal event or involves a message with a null image, then the event has a null image; otherwise its image is the event (a', b', e').

Thus, a projection yields a specification of image processes, which define the image protocol. The image protocol has a process and channel corresponding to each process and channel of the original protocol. The image process corresponding to P is denoted P'. Because the partitioning of the process state space defines an image for every process state and message, the projection also defines a mapping from the global state space of A to the global state space of of the image protocol: the image of a global state $g = [a_1, a_2, U_1, U_2]$ is $g' = [a_1', a_2', U_1', U_2']$; the image of a sequence of messages U is the sequence of (non-null) images of the messages in U. The initial global state of the image is the image of the initial global state.

For any event e and global states g and h in A such that $g \xrightarrow{e} h$, either $g' = h'$, or there is some image event e', such that $g' \xrightarrow{e'} h'$ in the image protocol. From this it follows ([3]) that the image of any computation of A is a path of the image protocol beginning at the initial global state. If the projection meets some additional structural conditions, then the image is said to be *well-formed*, and the image of any fair computation is a fair computation of the image protocol.

Now suppose B is an image protocol of A, and let F : $A \mapsto B$ be the functional mapping of global states defined by the projection. Then F has the following characteristic: any global state of A in which the state of process P is x maps to a global state in which the state of process P' is x'. Conversely, the only global states with images in which the state of P' is y, are those in which the state of P is one of the process states whose image is y, i.e., those in the set denoted by P **at** $\hat{y}$. Thus, the inverse image of the predicate defined by P' **at** y is defined by P **at** $\hat{y}$.

From the correspondence between boolean combinations of **at** expressions and set operations described earlier, and the fact that $\widehat{p \cap q} = \hat{p} \cap \hat{q}$, $\widehat{p \cup q} = \hat{p} \cup \hat{q}$, and $\widehat{\neg p} = \neg \hat{p}$, it follows that the inverse image of any predicate defined by a boolean combination of **at** expressions is defined by the same boolean combination of **at** expressions with each process P' replaced by P, and each process state y replaced by $\hat{y}$. For example, suppose protocol B in Figure 1 is an image of protocol A, and processes A_1 and A_2 correspond to processes B_1 and B_2, respectively. Then the inverse image of the predicate on B defined by B_1 **at** $x \Rightarrow B_2$ **at** y is defined by the predicate A_1 **at** $\hat{x} \Rightarrow A_2$ **at** $\hat{y}$. From now on we use $\hat{p}$ to represent the expression defining the inverse image of the predicate defined by p.

Now we can relate properties of one protocol to those of another. Suppose B is an image protocol of A, and p is a safety property of protocol B. Then p holds for every state in any path from the initial state of B. As noted above, the image of any computation α of A under the mapping defined by the projection is a path α' from the initial state of B. Since p holds at every state of α', it follows that $\hat{p}$ holds at every state of α. (Suppose not. Then for some α, α' is a path from the initial state of B, and there is a state g in α for which $\hat{p}(g) = \mathit{false}$. But by the definition of $\hat{p}$, we have $p(g') = \mathit{false}$. Since g' is a state in a path from the initial state of B, p cannot be a safety property of B, a contradiction.) Thus, $\hat{p}$ is a safety property of A.

If the image protocol B is well-formed, then the image of any fair computation α of A is α', a fair computation of B. If $p \leadsto q$ is a property of B, then every state in α' at which p holds is followed by some state at which q holds. But this means that every state in α at which $\hat{p}$ holds is followed by a state where $\hat{q}$ holds, so $\hat{p} \leadsto \hat{q}$ is a property of A.

If two protocols can be projected onto the same image protocol, then they share the inverse images of the safety properties of that image. Furthermore, if the images are well-formed, then they have its safety and liveness properties in common. This suggests the following approach to solving a conversion problem such as that described in the first section. We first specify the properties required of the conversion; then we look for a projection of the two protocols onto a common image with the properties we desire. If one is found, we are done, for the two projections define a correspondence between the messages of the two protocols: those with the same image are semantically equivalent.

If the protocols do not have a common image with the desired properties, a protocol converter must be obtained by considering the properties required and the structure of the processes involved in the conversion. Given a candidate converter, if the conversion system can be projected onto each of the two original protocols, the inverse images of their properties are properties of the conversion system. We illustrate these concepts in the next section with our example.

3 The Example Problem

The protocols in our example problem are the venerable Alternating-Bit (AB) protocol and a simplified version of the data transfer portion of the Bisync protocol. Each protocol provides reliable, sequenced transmission of data messages from the Sender to the Receiver over unreliable channels that may lose or corrupt messages. The protocols are represented in Figures 4 and 5. In those figures, the transitions labeled "accept" and "deliver" are internal events denoting interaction with some (higher-level) user entity. They represent acceptance of a message (e.g., higher-level protocol data) for transmission, and the delivery of a correctly received message to the receiver-side user, respectively. We are not concerned with the contents of the data messages themselves, as the service provided is a "transparent" transmission service. Thus we consider all data messages to be equivalent; in AB, the "D0" and "D1" messages represent the presence of a one-bit sequence number attached to the data message.

Modeling Lossy Channels

The example protocols are designed to function with unreliable channels; each Sender detects lost messages by means of a timeout mechanism. However, the channels in our cfsm model are perfectly reliable. To model channels that may lose or corrupt messages using ideal channels, we use "pseudo-messages" that represent loss, timeout, and corruption events. Receipt of a "tm" message represents a timeout event at the process receiving the message; receipt of an "err" message represents reception of a message corrupted in transit. "Ls" represents a message lost on its way to the receiver.

The pseudo-messages model losses as follows: each send event has two parallel "error" events, one representing the loss and one the corruption of the message sent. For example, in Figure 4, whenever the "–D0" event is enabled at the Sender, the "–Ls" and "–err" events are enabled as well. If a message is being sent *to* the process where a timeout may occur, the "tm" message may (nondeterministically) be sent instead of the intended message. If the timeout is located at the sending process, the "Ls" message may be sent instead of the intended message. Whenever a "Ls" message is received by a process, a "tm" message is returned to the sender. Thus, receipt of a "tm" message indicates loss of *either* a message or its acknowledgement. This mechanism models only non-premature timeouts, so that a timeout pseudo-message is received *if and only if* a loss has occurred.

The "err" pseudo-message carries no information other than that some message was sent and corrupted. It is not hard to see that if there is a path to a global state g, in which a message m is in some channel, then there is a path to a state h, identical to g except for the "loss" or "corruption" of m, represented by the presence of "Ls" or "err" in the channel in place of m. For example, if $[s, r, U, V]$ is a global state in some computation, where U is a sequence of messages containing m, then there is a computation in which $[s, r, U^m_{\text{err}}, V]$ is a state, where U^m_{err} is U with err substituted for m. Our fairness requirement ensures that in any fair computation, no message may be lost or corrupted infinitely many times without being correctly sent.

Properties of the Protocols

Our problem is to enable reliable transmission of data messages from the Sender of one protocol to the Receiver of the other. There are actually two different problems: enabling S_A and R_B to interoperate, and enabling S_B and R_A to interoperate. In general these problems must be solved separately; if an active intermediary (converter) is required it may be different for the two cases. However, if the two protocols have a well-formed common image protocol, the characteristics of either converter may be determined from the two projections. In the exposition that follows, we focus on the S_B-R_A conversion, the converter for which we denote by C_0. The S_A-R_B problem (with converter C_1) has an analogous solution.

Before constructing any converter, we must specify the properties required of the conversion system. Before considering those required properties, we first state the properties of the AB and Bisync protocols. The properties that we give can be proved by any of several means; we shall simply accept them as given.

Each of the following is a safety property of both AB and Bisync. ("S" represents the Sender process of either protocol, "R" the Receiver):

$$\begin{aligned} S \textbf{ at } 0 &\Rightarrow R \textbf{ at } 0 \\ S \textbf{ at } 3 &\Rightarrow R \textbf{ at } 3 \\ R \textbf{ at } 1 &\Rightarrow S \textbf{ at } 2 \\ R \textbf{ at } 4 &\Rightarrow S \textbf{ at } 5 \end{aligned}$$

These safety properties together assert that the protocol maintains synchronization; in particular, together they imply that no message is accepted for transmission unless the previous message has been delivered and acknowledged.

The following liveness properties are common to both protocols:

$$\begin{aligned} (S \textbf{ at } 1 \vee S \textbf{ at } 2) &\leadsto (S \textbf{ at } 3) \\ (S \textbf{ at } 4 \vee S \textbf{ at } 5) &\leadsto (S \textbf{ at } 0) \end{aligned}$$

These properties indicate that every message accepted for transmission will eventually be delivered and acknowledged.

The above properties may be considered to specify the "service" provided by the protocol. Since the same assertions are used to express the properties of both protocols, it is clear that state 0 of S_A corresponds (with respect to these properties) to state 0 of S_B, state 1 of R_A corresponds to state 3 of R_B, etc. Therefore it makes sense to consider the above properties to specify the service of the conversion system. We shall require that the above (with "S" representing "S_B" and "R" representing "R_A") be safety and liveness properties of the S_B-$R_A$conversion system.

Deriving the Conversion

The obvious and essential difference between the protocols is that AB includes a sequence number attached to each data message, while the Bisync protocol does not. This means that the AB Sender may safely transmit a data message to the Receiver after the Receiver has already delivered that message to the user; the sequence number attached to the message ensures that the duplicate is detected, and prevents the Receiver from delivering multiple copies of the same message. The Bisync Sender, however, may not retransmit a message unless the state of the Receiver is known; this is because the Receiver has no means to distinguish between old and new messages, and would otherwise deliver the retransmitted message as a new message. Thus, in AB, the responsibility for maintaining synchronization and ensuring correctly-sequenced delivery is essentially the Receiver's; in Bisync it resides with the Sender.

Since these two protocols seem to be so similar in structure, we first consider whether it might be possible to project one onto the other, or both onto a common image. We have already noted that the state spaces are substantially equivalent. By aggregating S_B's states 1 and 6 into a state equivalent to S_A's state 1, and similarly with 4 and 7, we can project the Bisync protocol onto an image protocol identical to AB, *except* for its messages. The image thus obtained still has only one "data" message, while AB has two semantically different ones, "D0" and "D1." In other words, the AB message set has a ***higher resolution*** than Bisync's.

On the other hand if we try to project AB onto Bisync, R_A can be projected onto R_B by a simple mapping of messages, but we cannot project S_A onto S_B, because S_B has a larger state space. That is, Bisync's state space has a ***higher resolution*** than AB's. So neither protocol is an image of the other. Although it is always possible to come up with a common image for any two protocols, it is not hard to see that in this case the common image (obtained by projecting out the semantic difference between the data messages in AB, i.e., by "folding" the state space in half) does not have the properties desired of the conversion.

When the protocols do not have a common image with the desired properties, an active converter (a finite state machine) is required. We therefore consider the necessary characteristics of such a converter. Now, we would like to be able to use the properties of the original protocols to prove that the conversion system involving our converter has the desired properties. We may do this using projection as described in [5] by making the combined function of the converter and S_B correspond to the behavior of S_A (with respect to communications with R_A), while similarly the interactions of C_0 with S_B correspond to what R_B would do. Figure 6 illustrates half of the idea: the communications between C_0 and R_A are projected onto the AB system.

So C_0 should "emulate" R_B when interacting with S_B, and S_A when interacting with R_A; the question is how to relate those two behaviors in a single machine. Obviously, data messages received from S_B will have a sequence number attached and be forwarded to R_A. But when should an acknowledgement be sent to S_B? How should retransmissions be handled?

Observe that a message may be lost or corrupted *after* being correctly received by C_0 but *before* being received by R_A. In this case it must be retransmitted. But if the communications between S_B and C_0 are to be projected onto those of Bisync, C_0 ***must*** send a ***positive*** acknowledgement after correctly receiving a data message; therefore, if a data message must be retransmitted to R_A, it must be retransmitted by C_0 without interaction with S_B. In other words, C_0 must "accept responsibility" for delivery of a message to R_A, once it is received from S_B. So C_0's behavior should repeat the pattern:

> emulate R_B until receiving a message correctly; then
>
> emulate S_A until the message is acknowledged by R_A; then
>
> resume emulation of R_B by acknowledging message

The stop-and-wait nature of our two protocols makes it a simple matter to construct a converter with the above

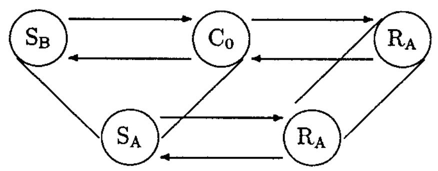

Figure 6: Projection of 3-process system onto 2-process system

structure; it is illustrated in Figure 7. In order to emulate S_A, it is necessary for C_0 to have a timeout mechanism of its own. To distinguish the pseudo-messages for the two timeout events, S_A's is designated "tm0" and C_0's "tm1." Note also that receipt of a correct data message from S_B corresponds to the "accept" event of S_A, while receipt of a positive ack from R_A corresponds to the "deliver" event of R_B. The converter for the S_A-R_B system is similar and is shown in Figure 8.

Correctness of the Conversion System

We have constructed C_0 so that AB and Bisync will be images of the conversion system. We now show that this is so, and also that the Bisync projection has *well-formed events*. If no message can remain in a channel forever in the original system (as is the case for the conversion system), then the well-formedness of the events in a projection is a sufficient condition for the image of any fair computation of the original system to be a fair computation of the image system.

We project the three-process system (S_B,C_0,R_A) onto each of the two-process original systems by considering two processes, and the channels between them, as a single machine, as shown in Figure 6. In order to project (S_B,C_0,R_A) onto (S_B,R_B), we define an aggregation of the states of the subsystem (C_0,R_A). Then we show that the image machine defined by the aggregation is R_B, so that the image of (S_B,(C_0,R_A)) is (S_B,R_B). A similar approach applies to AB: we project states of (S_B,C_0) onto those of S_A, and the image system is (S_A,R_A).

In what follows, we represent an arbitrary global state of the conversion system by $g = [s, U, V, c, X, Y, r]$, where s, c, and r are the states of S_B, C_0, and R_B respectively, and U, V, X, and Y are the states of the channels. Thus when we mention c, we refer to the state of the converter, r refers to the state of R_A, etc. In the projection onto Bisync, the image of g is a global state $g'_B = [s, U, V, r']$, where r' is a state of R_B, the image of the subsystem state $[c, X, Y, r]$. The image of g under the projection onto AB is $g'_A = [s', X, Y, r]$, where s' is the image of $[s, U, V, c]$.

To define the projection, we define the correspondence between the subsystem states $[s, U, V, c]$ and the process states of S_A, and between $[c, X, Y, r]$ and the states of R_B. These subsystem states are to be aggregated according to the state of C_0. The image of $[s, U, V, c]$ in the Bisync projection (or $[c, X, Y, r]$ in the AB projection) will be a function of c, and all subsystem states with the same value of c are aggregated together and have the same image. We

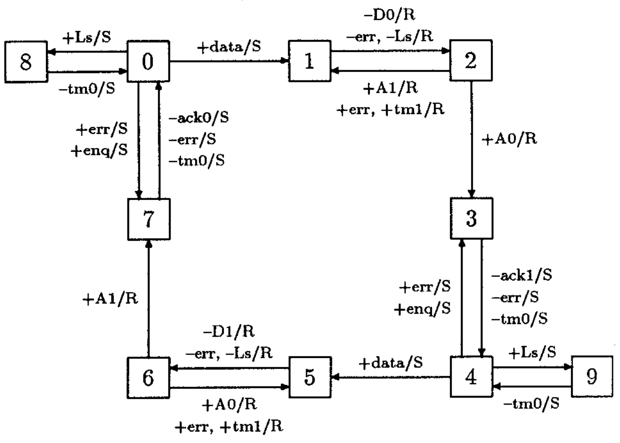

Figure 7: S_B-R_A Converter (C_0)

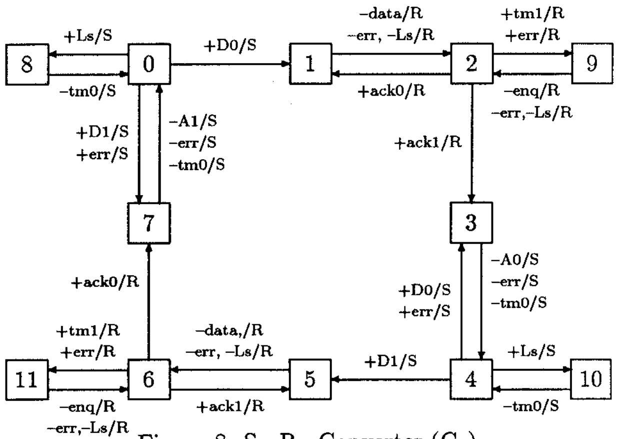

Figure 8: S_A-R_B Converter (C_1)

also aggregate further the subsystem states, by combining some states of C_0 in each projection, as shown in Figures 9 and 10. Comparison of Figures 5 and 9 shows that state 0 of C_0 corresponds to state 0 of R_B, the aggregated states 1 and 2 correspond to state 1 of R_B, etc. Each event of C_0 that involves sending or receiving a message to or from S_B corresponds exactly to an event of R_B, and for each event of R_B there is a corresponding event of C_0. Events of C_0 that involve interaction with R_A do not cross image state boundaries, and therefore have null images; the exceptions are the events (2, 3, +A0/R) and (6, 7, +A1/R), which correspond to the two "deliver" internal events of R_B.

Now, events of R_A can change the state of the (C_0,R_A) subsystem, but do not involve sending messages to the rest of the system; they are considered "internal" events of the composite process. Furthermore, since they do not change the state of C_0, they cannot affect the image process state of the subsystem, and therefore have null images. Since the image of S_B is itself, it follows that (S_B,R_B) is an image of (S_B,(C_0,R_A)); by a similar argument, (S_A,R_A) is the image of ((S_B,C_0),R_A). Table 1 shows how the image process states s' and r' under the two projections correspond to c.

We now proceed to show that the Bisync image has

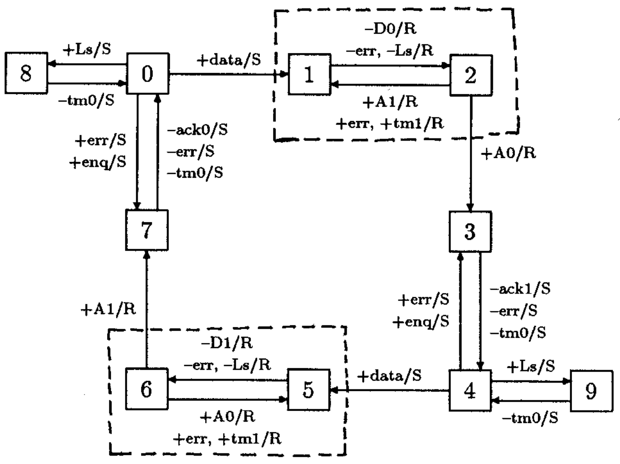

Figure 9: C_0 state aggregation in projection onto Bisync

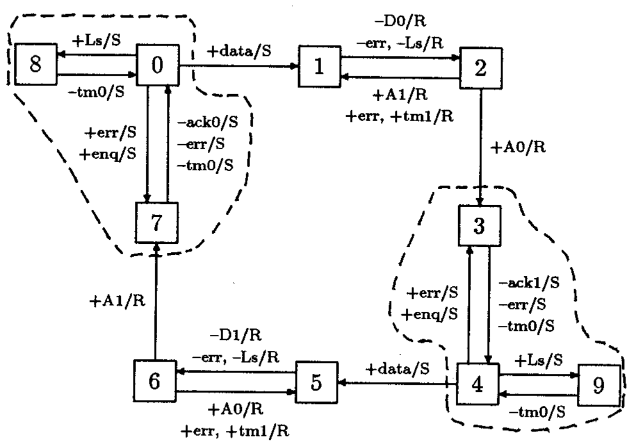

Figure 10: C_0 state aggregation in AB projection

well-formed events. An image process event e is well-formed if, for any global states g and h such that $g \xrightarrow{e} h$, and any r in $\hat{g}$, there is a path from r to some state in $\hat{h}$, such that the last event is e', and all other events are internal or send events (involving null-image messages) of that process. In a projection defined by an aggregation of process states, this condition may be checked simply by inspection of the processes' state spaces. In this case, because we are considering subsystem states as process states, we must show that the well-formedness condition is satisfied for the *subsystem's* state space, which we so far have not made explicit. However, thanks to the characteristics of C_0 (and the liveness property of AB), we can show the well-formedness of the image without having to explore the complete state space of the (C_0,R_A) subsystem.

Because no aggregation of process states of S_B is involved in the projection onto Bisync, and all messages project onto themselves, the events of S_B in the Bisync projection are clearly well-formed. Whenever an event of S_B is enabled in g'_B, the same event of S_B must be enabled in g; similarly for g'_A and events of R_A. So we just need to show the well-formedness of events of R_B in the Bisync image.

Let $g = [s, U, V, c, X, Y, r]$ be the state of the conversion system and $g'_B = [s, U, V, r']$ be its image under the projection onto Bisync. We shall show that if an event of R_B is enabled in g'_B, then a sequence of events not affecting s, U, or V can occur and take the subsystem to a state where an event whose image is the R_B event is enabled. Consider the events of R_B that might be enabled in a state $[s, U, V, r']$ when r' is 0. Figure 5 shows that these include (0, 1, +data), (0, 6, +Ls), (0, 5, +err), and (0, 5, +enq). Table 1 shows that when r' is 0, c is 0. Figure 9 shows if one of these events can occur in the image state $[s, U, V, 0]$, then the event with that image can occur in the state $[s, U, V, 0, X, Y, r]$. So all the events enabled when r' is 0 are well-formed. A similar argument applies to any events enabled when r' is 2, 3, 5, 6 or 7: only a single state of C_0 corresponds to each of these states of R_B, and the corresponding events of C_0 are enabled in each.

If r' is 1, the event (1, 2, deliver) is enabled. We shall show that for any g in any fair computation such that $g'_B = [s, U, V, 1]$, there is a sequence of events that takes the (C_0,R_A) subsystem to a state in which (2, 3, +A0/R) is enabled (the image of that event is (1, 2, deliver)). Table 1 shows that $c = 1$ or $c = 2$ in any such g. Consider $g'_A = [s', X, Y, r]$, the image of g under the projection onto AB: from Table 1 we have $s' = 1$ in g'_A if $c = 1$ in g, and $s' = 2$ in g'_A if $c = 2$ in g. Now, the image, under the projection onto AB, of the path in the conversion system from the initial global state to g is a path from the initial state of AB to g'_A. As we noted earlier, this path can be extended to a fair computation of AB. In that fair computation, the global state g'_A must be followed by some global state in which the state of S_A is 3, by AB's liveness property. Consider the sequence of events leading from g'_A to this state. From Figure 4, the only events of S_A that may occur in the sequence are those that correspond to events of C_0 that are enabled when c is 1 or 2. Furthermore, the last event in the sequence must be (2, 3, +A0). So corresponding to the sequence of events beginning in $[s', X, Y, r]$ and culminating in a state where the event (2, 3, +A0) of S_A is enabled, is a sequence of events of C_0 and R_A beginning in $g = [s, U, V, c, X, Y, r]$ and culminating in a state where the event (2, 3, +A0/R) of C_0 is enabled. All events in this sequence have null images in the Bisync projection. Thus we have shown there is a sequence of events with null images that takes the system to a state where the event whose image is (1, 2, deliver) is

C_0 state	0	1	2	3	4	5	6	7	8	9
Image R_B state	0	1	1	2	3	4	4	5	6	7
Image S_A state	0	1	2	3	3	4	5	0	0	3

Table 1: Mapping from converter to image process state spaces

enabled, from any state g of the conversion system such that $(1, 2, \text{deliver})$ is enabled in g'_B; that event is therefore well formed. A symmetric argument applies to the other "deliver" event. Thus all the events of the image Bisync protocol are well formed.

Properties

Because AB and Bisync are both images of the conversion system, the inverse images of their safety properties hold in it. Since the Bisync image has well-formed events, the inverse image of its liveness property holds in the conversion system.

The global state predicate "R_B **at** x" defines the set of states $[s, U, V, r]$ of Bisync for which $r = x$. The inverse image of this set under the projection onto Bisync contains exactly those states g of the conversion system such that $r' = x$ in g'_B. Table 1 associates with each such x a state or states of C_0, so that we can make explicit the inverse image of the predicate "R_B **at** x" for any particular x. The inverse image of the predicate "S_B **at** x" is of course itself under this projection. Thus we have the following inverse images of the properties given earlier.

From the projection onto Bisync:

$$S_B \textbf{ at } 0 \Rightarrow C_0 \textbf{ at } 0$$
$$S_B \textbf{ at } 3 \Rightarrow C_0 \textbf{ at } 4$$
$$(C_0 \textbf{ at } 1 \vee C_0 \textbf{ at } 2) \Rightarrow S_B \textbf{ at } 2$$
$$(C_0 \textbf{ at } 5 \vee C_0 \textbf{ at } 6) \Rightarrow S_B \textbf{ at } 5$$

From the projection onto AB:

$$(C_0 \textbf{ at } 0 \vee C_0 \textbf{ at } 7 \vee C_0 \textbf{ at } 8) \Rightarrow R_A \textbf{ at } 0$$
$$(C_0 \textbf{ at } 3 \vee C_0 \textbf{ at } 4 \vee C_0 \textbf{ at } 9) \Rightarrow R_A \textbf{ at } 3$$
$$R_A \textbf{ at } 1 \Rightarrow C_0 \textbf{ at } 2$$
$$R_A \textbf{ at } 4 \Rightarrow C_0 \textbf{ at } 6$$

The safety properties desired of the conversion system follow immediately from the properties above:

$$\begin{aligned} S_B \textbf{ at } 0 &\Rightarrow R_A \textbf{ at } 0 \\ S_B \textbf{ at } 3 &\Rightarrow R_A \textbf{ at } 3 \\ R_A \textbf{ at } 1 &\Rightarrow S_B \textbf{ at } 2 \\ R_A \textbf{ at } 4 &\Rightarrow S_B \textbf{ at } 5 \end{aligned}$$

The inverse image of the Bisync liveness property is

$$(S_B \textbf{ at } 1 \vee S_B \textbf{ at } 2) \leadsto S_B \textbf{ at } 3,$$

which is the desired property.

4 Summary and Conclusions

We have illustrated the use of the projection paradigm both in reasoning about protocol conversions and proving them correct. The notion of the inverse image of a predicate and of simple properties was formalized, and used in obtaining properties of an example conversion system from those of the two original protocols. The importance of formal specifications in solving problems of this kind is clear.

The example problem illustrates that even when protocols provide identical service and have substantially equivalent state spaces, a simple, stateless message mapping may not be sufficient. In this case, the essential difference between the two protocols took the form of additional messages in one protocol, and additional process states in the other. Work is continuing on understanding and formalizing the requirements for protocol conversion.

References

[1] P. E. Green, Jr., "Protocol Conversion," *IEEE Transactions on Communications*, March 1986.

[2] I. Groenbak, "Conversion between TCP and ISO Transport Protocols as a means of achieving interoperability," *IEEE Journal on Selected Areas in Communications*, March 1986.

[3] S. S. Lam & A. U. Shankar, "Protocol Verification via Projections," *IEEE Transactions on Software Engineering*, July 1984.

[4] S. S. Lam, "Protocol Conversion: Correctness Problems" *Proceedings ACM SigComm '86 Symposium*, Stowe, VT, August 1986.

[5] S. S. Lam, "Protocol Conversion," University of Texas Computer Sciences Department Technical Report TR-87-05, February 1987; to appear in *IEEE Transactions on Software Engineering*, 1987.

[6] Z. Manna & A. Pnueli, "Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Programs," *Science of Computer Programming* 4, North-Holland, 1984.

[7] K. Okumura, "A Formal Protocol Conversion Method," *Proceedings ACM SigComm '86 Symposium*, Stowe, VT, August 1986.

[8] C. S. Sunshine, ed.,"Communications Protocol Modeling," Artech House, 1981