PROTOCOL SPECIFICATION, TESTING, AND VERIFICATION, C. Sunshine (ed.)
North-Holland Publishing Company

AN ILLUSTRATION OF PROTOCOL PROJECTIONS

Simon S. Lam and A. Udaya Shankar

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

The method of protocol projections to facilitate the analysis of communication protocols is illustrated with two examples. In our model, protocol entities are connected by communication channels; messages sent by the protocol entities through the channels may be lost, duplicated and/or reordered. Protocols with several distinguishable functions are considered. Image protocols are constructed separately for each function. Image protocols are obtained by aggregating protocol entity states, messages and events using equivalence relations. As a result, image protocols are typically much simpler than the original protocol and can be analyzed more easily using available techniques. We employ two protocol examples to illustrate the construction of entity states, messages and events of some image protocols. In the first example, the protocol entities are described by a finite state machine model. The second example is a full-duplex data transfer protocol and its entities are described by a programming language model. In both cases, the image protocols constructed satisfy a well-formed property. As a result, each image protocol constructed is faithful in the sense that its logical correctness properties are the same as the logical correctness properties of the original protocol with respect to the projected function.

1. INTRODUCTION

The method of protocol projections is intended to facilitate the analysis of nontrivial protocols that are too complex to be analyzed with one of the basic approaches [1-5]. We observe that real-life protocols typically have several distinguishable functions. For example, the HDLC protocol has at least three functions: connection management, and one-way data transfers in two directions. We would like to ask questions regarding the logical correctness behavior of the protocol concerning these functions. Instead of asking such questions all at the same time, we may ask them with respect to one function of the protocol at a time. The analysis approach is to construct from the given protocol an <u>image protocol</u> for each of the functions that are of interest to us. These functions will be referred to as the <u>projected functions</u>. The entity states, messages, and events of an image protocol are obtained from those of the original protocol using certain equivalence relations. Entity states, messages and events that are equivalent with respect to a projected function are aggregated in the image protocol. As a result, image protocols are simpler than the original protocol. Each image protocol can thus be more easily analyzed using available means. For example, an image protocol for the function of one-way data transfer of HDLC will be of the same complexity as Stenning's data transfer protocol which has been successfully analyzed [3,4].

A communication protocol system consists of protocol entities connected by communication channels (real or virtual) over which messages are exchanged. A communication protocol system can be modeled abstractly as a transition system specified by the pair (G,τ) where G is the set of global states and τ is a binary relation in G called the set of transitions [6]. The global state of such a model of interacting protocol entities is specified by a joint description of the states of the protocol entities and communication channels. State transitions correspond to the occurrence of various events: an entity sending or receiving messages, errors in channels, an entity receiving signals from its user and timers, etc.

Given an initial state g_0, τ determines the reachability tree (space) R. R is a directed graph with nodes being elements of G and arcs being elements of τ. R contains all available information on logical correctness properties of the protocol. Let R_s denote the set of reachable states in G. R_s, which may be obtained from R, determines the safety (partial correctness) properties of the protocol. Liveness properties, however, require knowledge of the set of paths in R.

The protocol projection idea can be illustrated by the following example. Consider a protocol model with the state description (x, y, z) and the set R_s of reachable states. Suppose that we are only interested in a safety assertion that involves only the variables x and y. To determine whether the assertion is true, it is sufficient for us to know the image of R_s on the (x, y) plane. Obviously, if R_s is known, its image on the (x, y) plane is readily available. However, the complexity of R (and thus R_s) is the basic source of difficulty in protocol analysis. Thus, we would like to take the original protocol and construct from it an image protocol whose set of reachable states duplicates the image of R_s on the (x, y) plane. In fact, we want to construct image protocols that are faithful with respect to both safety and liveness properties. In general, an image protocol is said to be <u>faithful</u> if its logical correctness properties are the same as the logical correctness properties of the projected function in the original protocol.

The method of protocol projections was first described by these authors in [7]. Subsequently, the theory has been extended to a protocol model with communication channels that may lose, duplicate or reorder messages [8,9]. The main result in [8,9] is that if an image protocol is constructed to satisfy a well-formed property then it is faithful. Although the well-formed property is a sufficient condition, we have found that it is the weakest condition that one can have without any knowledge of the reachability tree R of the protocol. (Note again that we cannot assume any knowledge of R since its complexity is the basic source of our difficulties.)

Given a multi-function protocol, the successful construction of well-formed image protocols that are much smaller than the given protocol depends upon the protocol's structure. One can think of a multi-function protocol as well-structured if it gives rise to "minimal" well-formed image protocols for its functions.

The application of protocol projections to the analysis of protocols is as follows. Suppose that we are given a protocol and one or more assertions that specify the correct behavior of some protocol function (service specification of the function). Suppose also that a verifier is available for checking the validity of assertions for a given protocol. Instead of feeding the assertion(s) and the original protocol into the verifier, our objective is to first construct a well-formed image protocol (which should be much simpler than the original protocol). The image protocol and assertions are then fed into the verifier for evaluation. If we are interested in several functions of the protocol, a different image protocol is generated for each function.

In Section 2, we shall first present our basic protocol model. In Sections 3 and 4, two protocol examples are employed to illustrate the construction of well-formed image protocols. In the first example (Section 3), the protocol entities are described by a finite state machine model. The second example in Section 4 is a full-duplex data transfer protocol whose protocol entities are described by a programming language model.

For a complete exposition of our basic protocol model, the method of protocol projections and the well-formed property, the reader is referred to [8,9,10]. In a companion paper [11], time variables and time events are added to our basic protocol model for the analysis of time-dependent communication protocols and distributed systems.

2. THE PROTOCOL MODEL

Our basic model is an extension of the model of Brand and Zafiropulo [12]. Let P_1 and P_2 be two protocol entities that communicate with each other. P_1 sends messages to P_2 through channel C_1, and P_2 sends messages to P_1 through channel C_2. (See Fig. 1.) Messages transmitted through a channel may be reordered, duplicated, and/or lost (due to corruption by noise).

The consideration of a protocol model with only two entities is strictly for notational simplicity. Our results in this paper and in [7-10] are applicable to any network of protocol entities interconnected by communication channels.

A channel from one entity to another consists of all buffers and communication media between the entities. At any time, the channel contains a (possibly empty) sequence of messages. We assume that a message can be sent into the channel without any constraint by the channel (i.e. unblocked sends). Note that most communication protocols have some measure of flow control. As a result, their buffer requirements for messages in transit between entities are bounded. Hence, the assumption of unblocked sends is equivalent to being able to satisfy those buffer requirements.

We also assume that the message flow in a channel cannot be blocked due to an entity refusing to accept messages indefinitely. The first message in a channel's sequence of messages will be deleted after some finite time duration (by the channel, the entity or some other agent).

The above two assumptions are both reasonable whether the communication channels are real or virtual.

For i = 1 and 2, let S_i be the set of states of P_i, o_i be the initial state of P_i in S_i, and M_i be the set of messages sent by P_i.

Let $\underline{m}_i$ denote a sequence of messages representing the state of channel C_i. A message reception event removes the first message in the sequence. A message send event appends a new message to the end of the sequence. The set of all possible message sequences in C_i is a subset of

$$\underline{M}_i = (\bigcup_{k=1}^{\infty} M_i^k) \cup \{\langle\rangle\}$$

where $\langle\rangle$ denotes the null sequence and M_i^k is the cartesian product of M_i with itself k times.

The state space of the global model of protocol interaction is

$$G = S_1 \times \underline{M}_1 \times \underline{M}_2 \times S_2.$$

Each global state is a 4-tuple $\langle s_1, \underline{m}_1, \underline{m}_2, s_2\rangle$ where $s_i \in S_i$ and $\underline{m}_i \in \underline{M}_i$ for i = 1 and 2. The initial global state denoted by g_0 will be assumed to be $\langle o_1, \langle\rangle, \langle\rangle, o_2\rangle$ in the rest of this paper (without any loss of generality).

The events in the protocol are either entity events or channel events. An event can occur only if certain conditions, denoted as its <u>enabling condition</u>, hold. When an <u>enabled</u> event occurs, it changes the state of one or more components of the global state.

There are three types of entity events. We describe these events for P_1.

(1) (s, r, m), where $m \in M_1$, denotes an event of P_1 due to the sending of a message m by P_1 into channel C_1. This send event is enabled when P_1 is in state s. After the event occurrence, P_1 is in state r and m has been appended to the end of the message sequence in C_1. The set of such send events is a subset of $S_1 \times S_1 \times M_1$.

(2) (s, r, m), where $m \in M_2$, denotes an event of P_1 due to the reception of message m by P_1 from channel C_2. This receive event is enabled when P_1 is in state s and m is the first message in C_2. After the event occurrence, P_1 is in state r and m has been deleted from the message sequence in C_2. The set of such receive events is a subset of $S_1 \times S_1 \times M_2$.

(3) (s, r, α), where α is a special symbol indicating the absence of a message, denotes an internal event of P_1 that does not involve a message. This internal event is enabled when P_1 is in state s; after the event occurence P_1 is in state r. Internal events model such events as timeouts and interactions between the entity and its local user. The set of internal events is a subset of $S_1 \times S_1 \times \{\alpha\}$.

The three types of events for entity P_2 are similarly defined, but with the send events in $S_2 \times S_2 \times M_2$, receive events in $S_2 \times S_2 \times M_1$, and internal events in $S_2 \times S_2 \times \{\alpha\}$.

Our model defines 3 types of events in communication channels (loss, duplication and reordering events). These events transform the sequence of messages in a communication channel and are enabled whenever the channel has messages. Furthermore, the following axiom about the behavior of channels is assumed.

Channel error axiom. If channel C_i loses (duplicates, repositions) messages, then any message in any message sequence in C_i may be lost (duplicated, repositioned).

3. PROTOCOL PROJECTION ILLUSTRATED BY AN EXAMPLE

The first example consists of two interacting finite state machines shown in Figure 2. Protocol entity P_1 has state space $S_1 = \{0, 1, 2, 3, 4, 5, 6\}$ and sends messages from $M_1 = \{a_1, a_2, a_3\}$. Protocol entity P_2 has state space $S_2 = \{0, 1, 2, 3, 4, 5, 6\}$ and sends messages from $M_2 = \{b_1, b_2, b_3\}$. The events of entity P_1 are shown in Figure 2 (a). An arc from node i to node j with a label m specifies event (i,j,m), where m is α for an internal event, m is $-a_i$ for sending message a_i, and m is $+b_i$ for receiving message b_i. The events of entity P_2 are similarly shown in Figure 2 (b).

Note that from state 4 in S_2, the reception of a_2 can cause a transition to either state 3 or 5. This nondeterministic behavior is allowed in our model, and is useful for representing certain features in real protocols and systems.

Projections are achieved using equivalence relations. The image of a protocol quantity is obtained by aggregating all those protocol quantities that are equivalent. Images of protocol entity states, messages, and events are next defined and should be obtained in the order shown below.

Projection of entity states

We start by examining the entity states in S_1 and S_2 that are equivalent with respect to the projected function. For $i = 1$ and 2, for any entity state $s \in S_i$, the image of s is the set of entity states in S_i that are equivalent to s. We denote the image of s by s'. s' is also referred to as an image entity state. Let S_i' denote the set of images of states in S_i. S_i' is the image state space of P_i.

Consider the example in Figure 2. Suppose that for a particular function, states 0, 1, 2, 3 and 4 are equivalent, and states 5 and 6 are equivalent. This equivalence relation corresponds to the partition $\{ \{0,1,2,3,4\}, \{5,6\} \}$ of S_1 indicated by Figure 3 (a). Let image state $0'$ denote the partition cell $\{0,1,2,3,4\}$, and image state $5'$ denote the partition cell $\{5,6\}$. $0'$ is the image of states 0, 1, 2, 3 and 4 in S_1. $5'$ is the image of states 5 and 6 in S_1. The image state space of P_1 is $S_1' = \{0', 5'\}$.

For the same function, suppose that in S_2 states 0, 3 and 4 are equivalent, states 1 and 5 are equivalent, and states 2 and 6 are equivalent. This equivalence relation corresponds to the partition $\{ \{0,3,4\}, \{1,5\}, \{2,6\} \}$ of S_2 indicated in Figure 3 (b). Let image states $0'$, $1'$ and $2'$ respectively denote the partition cells $\{0,3,4\}$, $\{1,5\}$ and $\{2,6\}$. Then, the image state space of P_2 is $S_2' = \{0', 1', 2'\}$.

Projection of messages

The above equivalence relation is next extended to messages in M_1 and M_2. Two messages m and n in M_i are said to be equivalent if and only if they cause identical changes in both image state spaces S_1' and S_2'. For any message $m \in M_i$, the image of m is the set of messages in M_i that are equivalent to m. The image of m, denoted by m', is also referred to as an image message. Messages that cause only changes internal to image entity states in both S_1' and S_2' are said to have the null image. The image message sets are defined by

$$M_i' = \{m' : m' \text{ is not null, } m \in M_i\}, \text{ for } i = 1 \text{ and } 2.$$

Consider the example protocol. In Figures 3 (a) and 3 (b), if we relabel each state by its image and collapse all identically labelled states, without deleting any events, then we obtain Figures 4 (a) and 4 (b). From these figures we can observe the state changes in S_1' and S_2' caused by the messages in M_1 and M_2.

We will first examine the message set M_1. The state transitions caused by message a_1 are $(0',0')$ in S_1', and $(1',1')$ in S_2'. Hence a_1 has a null image. Both messages a_2 and a_3 cause the state transition $(0',5')$ in S_1' and the state transition $(0',1')$ in S_2'. However, a_2 is not equivalent to a_3 because a_2 causes state transition $(0',0')$ in S_2' while a_3 does not.

This equivalence relation corresponds to partitioning M_1 as $\{\{a_1\}, \{a_2\}, \{a_3\}\}$. The image of a_1 is null. Let the image messages a_2' and a_3' denote the partition cells $\{a_2\}$ and $\{a_3\}$ respectively. The image message set M_1' is $\{a_2', a_3'\}$.

We will now examine the message set M_2. The state changes caused by message b_1 are $(5',0')$ in S_1' and $(2',0')$ in S_2'. The state changes caused by message b_3 are $(5',0')$ in S_1' and $(2',0')$ in S_2'. Hence b_1 is equivalent to b_3. The state changes caused by message b_2 are $(0',0')$, $(5',5')$ in S_1', and $(0',0')$ in S_2'. Hence b_2 has a null image. This equivalence relation corresponds to the partition $\{ \{b_1,b_3\}, \{b_2\} \}$ of M_2. The image of b_2 is null. Let image message b_1' denote the partition cell $\{b_1,b_3\}$. The image message set M_2' is $\{b_1'\}$.

<u>Projection of entity events</u>

The above equivalence relations are next extended to entity events. The image of an event (s,r,m) is given by (s',r',m'), where the image of α is still α for internal events. Recall that if the image message m' is null, then by definition $s'=r'$. Consequently, if m' is null, the event (s,r,m) does not change the image global state of the protocol model. Such an event is said to have a <u>null image</u>. Similarly, an internal event (s,r,α) with $s'=r'$ has a null image. The set of image events at P_i for $i = 1$ and 2 is defined as

$$T_i' = \{(s',r',m') : (s',r',m') \text{ is not null, } (s,r,m) \in T_i\}.$$

Consider the example protocol. Recall that messages a_1 and b_2 have null images. From Figures 3 (a) and 4 (a), the events in P_1 having null images are $(0,1,b_2)$, $(1,2,\alpha)$, $(2,0,a_1)$, $(0,3,\alpha)$, $(3,4,a_1)$, $(4,3,\alpha)$, and $(5,6,b_2)$. Event $(3,5,a_2)$ has the image $(0',5',a_2')$. Event $(4,5,a_3)$ has the image $(0',5',a_3')$. Events $(5,0,b_1)$, $(5,4,b_3)$, $(6,4,b_1)$, and $(6,2,b_3)$ are equivalent and have the image $(5',0',b_1')$. Hence,

$T_1' = \{ (0',5',a_2'), (0',5',a_3'), (5',0',b_1') \}$.

From Figures 3 (b) and 4 (b), the events in P_2 having null images are $(0,3,\alpha)$, $(3,4,b_2)$, and $(1,5,a_1)$. The image of $(4,3,a_2)$ is $(0',0',a_2')$. Note that the image of $(4,3,a_2)$ is not null because a_2' is not null. Events $(0,1,a_2)$, $(3,1,a_2)$, and $(4,5,a_2)$ are equivalent and have the image $(0',1',a_2')$. Event $(4,5,a_3)$ has the image $(0',1',a_3')$. Events $(1,2,\alpha)$ and $(5,6,\alpha)$ are equivalent and have the image $(1',2',\alpha)$. Events $(2,0,b_1)$ and $(6,4,b_3)$ are equivalent and have the image $(2',0',b_1')$. Therefore,

$T_2' = \{(0',0',a_2'), (0',1',a_2'), (0',1',a_3'), (1',2',\alpha), (2',0',b_1')\}$.

Image protocol

Our objective is to define a new protocol that is faithful to the projected function, i.e., such that its logical correctness properties are the same as the logical correctness properties of the projected function in the original protocol. We call this protocol an image protocol. A natural candidate is a protocol between entities P_1' and P_2', using channels C_1 and C_2, obtained as follows. For i = 1 and 2, let S_i' be the entity state space of P_i', let M_i' be the set of messages sent by P_i', and let T_i' be the set of entity events for P_i'. Given the channel error axiom, the events of channel C_i remain the same as before.

In general, the image protocol system may not be faithful to the projected function in the original protocol system. We present next sufficient conditions for the image protocol to be faithful to the projected function [8,9]. The objective in the construction of an image protocol is therefore to find the most coarse partitions of entity states (in other words, the smallest image protocol) which satisfy the sufficient conditions.

Definition. For i = 1 and 2, for a and b in S_i, b is internally reachable from a if $a'=b'$ (they have the same image), and there is a sequence of internal events in T_i with state changes inside a' that will take P_i from a to b.

Note that in our current model, all send events are unblocked. As a result, send events involving messages of null images can be regarded as internal events for the above definition.

Definition. For i = 1 and 2, an image send or internal event $(s',r',m') \varepsilon T_i'$ for $m' \varepsilon M_i' U\{\alpha\}$ is well-formed if, for every a whose image is s', the following condition holds: for some n whose image is m' there is some $b \varepsilon S_i$ such that b is internally reachable from a, and $(b,c,n) \varepsilon T_i$ for some $c \varepsilon r'$.

Definition. For $i = 1$ and 2, an image receive event $(s',r',m') \in T_i'$ for $m' \in M_j'$ $(j \neq i)$ is well-formed if, for every a whose image is s', the following condition holds: for every n whose image is m' there is some $b \in S_i$ such that b is internally reachable from a, and $(b,c,n) \in T_i$ for some $c \in r'$.

If in either of the above definitions of well-formed events, the length of the internal path is zero (i.e., $b=a$), then we say that the image event is strongly well-formed.

Note from the construction of T_1' and T_2', that for every $(s',r',m') \in T_i'$, there is an $(a,b,n) \in T_i$ such that $a'=s'$, $b'=r'$, and $n'=m'$.

Definition. The image entity state spaces S_1' and S_2' are said to be well-formed (strongly well-formed), if every image entity event $(s',r',m') \in T_1' \cup T_2'$ is well-formed (strongly well-formed).

It has been shown that if the image entity state spaces S_1' and S_2' are well-formed, then the image protocol is faithful to the projected function in the original protocol [8,9].

Consider the example protocol introduced earlier. We have already obtained the image state spaces $S_1'=\{0',5'\}$ and $S_2' = \{0',1',2'\}$, the image message sets $M_1' = \{a_2',a_3'\}$ and $M_2' = \{b_1'\}$, and the image event sets $T_1' = \{ (0',5',a_2'), (0',5',a_3'), (5',0',b_1') \}$ and $T_2' = \{ (0',0',a_2'), (0',1',a_2'), (0',1',a_3'), (1',2',\alpha), (2',0',b_1') \}$. Using these quantities, we construct the image protocol system shown in Figures 5 (a) and 5 (b).

We will now show that the image events in T_1' are well-formed. First, consider event $(0',5',a_2') \in T_1'$. Because of the paths $1 \xrightarrow{\alpha} 2 \xrightarrow{-a_1} 0 \xrightarrow{\alpha} 3$ and $4 \xrightarrow{\alpha} 3$, state 3 is internally reachable from states 1, 2, 0 and 4. From state 3, the event $(3,5,a_2)$ can be executed. Hence $(0',5',a_2')$ is well-formed.

Because of event $(3,4,a_1)$, state 4 is internally reachable from state 3, and hence from states 0, 1 and 2 also. From 4, event $(4,5,a_3) \in T_1$ can be executed. Thus, event $(0',5',a_3') \in T_1'$ is well-formed.

Because of the events $(5,0,b_1)$, $(5,4,b_3)$, $(6,4,b_1)$, and $(6,2,b_3)$ in T_1, image event $(5',0',b_1') \in T_1'$ is strongly well-formed.

Next, we will show that the image events in T_2' are well-formed. Consider image event $(0',0',a_2') \in T_2'$. Because of events $(0,3,\alpha)$ and $(3,4,b_2)$ in T_2, state 4 is internally reachable from 0 and 3. This, and the event $(4,3,a_2) \in T_2$ makes image event $(0',0',a_2')$ well-formed.

Image event $(0',1',a_2') \in T_2'$ is strongly well-formed because of events $(0,1,a_2)$, $(3,1,a_2)$, and $(4,5,a_2)$ in T_2.

Image event $(0',1',a_3') \in T_2'$ is well-formed because of event $(4,5,a_3)$ in T_2, and because 4 is internally reachable from 0 and 3.

Image event $(1',2',\alpha) \in T_2'$ is strongly well-formed because of events $(1,2,\alpha)$ and $(5,6,\alpha)$ in T_2.

Image event $(2',0',b_1') \in T_2'$ is strongly well-formed because of events $(2,0,b_1)$ and $(6,4,b_3)$ in T_2.

Thus, we have shown that the image protocol in Figure 5 is well-formed.

A safety assertion

Given the initial state $(0',\langle\rangle,\langle\rangle,0')$ for this image protocol system, and assuming channels C_1 and C_2 are error-free, it is easy to verify that the following assertion holds for the image protocol:

P_1 is in state $5'$ => Exactly one of the following holds:

(a) message a_2' or a_3' is in C_1, or

(b) P_2 is in state $1'$ or $2'$, or

(c) message b_1' is in C_2, or

(d) P_2 is in state $0'$ and channels C_1 and C_2 are empty (deadlock situation).

Note that P_1 in state $5'$ means that P_1 is in any state whose image is $5'$. Similarly, message b_1' is in C_2 means that any message whose image is b_1' is in C_2.

We can verify whether this assertion is valid by examining the reachability tree of the original protocol in Figure 2. However, since the image protocol has been shown to be well-formed, we know that the above assertion is valid for the original protocol by virtue of the theorems presented in [8,9].

4. A FULL-DUPLEX DATA TRANSFER PROTOCOL

We illustrate in this section the application of protocol projection to a full-duplex data transfer protocol whose entities are described by a programming language model. Since the protocol is full-duplex, it has two basic distinguishable functions: one-way data transfers from P_1 to P_2 and from P_2 to P_1.

Let DATASET denote the set of data blocks that can be sent in this protocol. Consider protocol entity P_1. P_1 has an infinite array of data blocks, SOURCE[i] for i=0,1,2,..., destined for P_2, and an infinite array, SINK[i] for i=0,1,2,..., to store data blocks received from P_2. SINK is initially empty. Additionally, the following variables are used in P_1: VS and VR which are nonnegative integers, and D_OUT, ACK_DUE and BUSY which are Boolean variables. VS points to the data block in SOURCE to be sent next. VR points to the position in SINK to be next filled. D_OUT is true if (and only if) a data block has been sent but not yet acknowledged. ACK_DUE is true if (and only if) a received data block has to be acknowledged. BUSY can be viewed as an externally operated switch; all events of P_1 are inhibited if (and only if) BUSY is true.

The state of P_1, at any time, is given by the value of the 7-tuple <VS, D_OUT, SOURCE, VR, ACK_DUE, SINK, BUSY> of P_1. Let S_1 denote the state space of P_1. Entity P_2 has a similar set of variables. For convenience, we have omitted qualifiers (1 or 2) for these variables and we shall omit them as long as it is clear whether we are referring to P_1 or P_2. In both entities, the initial state is given by VS and VR equal to 0, D_OUT and ACK_DUE equal to false, SINK equal to empty, and SOURCE equal to some infinite array of data blocks. (In both entities, SOURCE does not change its value during the protocol interaction.)

Each message in the protocol is a tuple with one or more components. The first component is used to identify the kind of message, and it can take the (character string) values DATA, ACK and DATA&ACK, corresponding to three kinds of messages. A DATA message is a 2-tuple (DATA, d), where d is a data block from DATASET. There are as many DATA messages as data blocks in DATASET. An ACK message is the 1-tuple (ACK), signifying a positive acknowledgement for a received data block. Unlike DATA messages, there is only one ACK message. A DATA&ACK message is a 2-tuple (DATA&ACK, d) where d is a data block from DATASET.

The set of messages that can be sent by P_1 is given by

$$M_1 = \{(\mathrm{ACK})\} \cup \{(\mathrm{DATA},d) : d \in \mathrm{DATASET}\} \cup \{(\mathrm{DATA\&ACK},d) : d \in \mathrm{DATASET}\}.$$

The message set M_2 for P_2 is the same as M_1.

The set of entity events for P_1 is presented in Table 1. Events 1-5 correspond to the sending of a message by P_1. Events 6-8 correspond to th reception of a message by P_1. Events 4 and 5 correspond to internal events caused by an agent locally connected to P_1 (e.g., user, channel controller). The enabling condition of an event defines the entity states and channel states at which the event may take place. The action of each event causes the entity to enter a new state.

SDATA and RDATA are variables taking values from DATASET. SDATA and RDATA can be thought of as temporary buffers for transmission and reception (respectively) of data blocks. In Table 1, the operation put(CHANNEL1, (DATA,SDATA)) sends a DATA message with the value of SDATA as its data block into CHANNEL1 (i.e. appends the DATA message to the end of the sequence of messages in CHANNEL1). The operations put(CHANNEL1, (DATA&ACK,SDATA)) sends into CHANNEL1 a DATA&ACK message with the value of SDATA as its data block. The operation put(CHANNEL1, (ACK)) sends an ACK message into CHANNEL1.

The function first(CHANNEL2) indicates the type of the message that is at the head of CHANNEL2 and has arrived at P_1. When a DATA message is at the head of CHANNEL2, the operation get(CHANNEL2, (DATA,RDATA)) removes the message from CHANNEL2 and assigns the data block in the message to RDATA. When a DATA&ACK message is at the head of CHANNEL2, the operation get(CHANNEL2, (DATA&ACK,RDATA)) removes the message from CHANNEL2, and assigns the data block in the message to RDATA. When an ACK message is at the head of CHANNEL2, the operation get(CHANNEL2, (ACK)) removes the message from CHANNEL2.

This example protocol has two functions corresponding to data transfers in the two directions. The protocol is extremely simple but it embodies two types of dependencies that one encounters when one attempts to decompose a protocol into functional components. First, the variable BUSY is shared by both functions of the protocol. Second, the messages of type DATA&ACK are also shared. Such dependencies present major obstacles for protocol analysis using a decomposition approach. The method of protocol projections will be used to obtain a faithful image protocol for each function.

Image protocol

Consider the function of one-way data transfer from P_1 to P_2. Observe that variables VR, ACK_DUE and SINK in P_1, and VS, D_OUT and SOURCE in P_2 are not needed for the P_1 to P_2 data transfer. Thus, the variables of the projected function (referred to as function variables) are VS, D_OUT, SOURCE and BUSY in P_1, and VR, ACK_DUE, SINK and BUSY in P_2.

At any time, the image state (i.e. state of the projected function) of P_1 is given by the value of <VS, D_OUT, SOURCE, BUSY>. Let S_1' denote the image state space of P_1. Thus, two states s and r in S_1 are equivalent if they differ only in the values of VR, ACK_DUE and SINK.

At any time, the image state of P_2 is given by the value of <VR, ACK_DUE, SINK, BUSY>. Let S_2' denote the image state space of P_2. Thus, two states s and r in S_2 are equivalent if they differ only in the values of VS, D_OUT and SOURCE.

By examining the state changes in the image spaces S_1' and S_2' due to send and receive events, the following can be shown about messages in M_1. The message (ACK) has a null image. The messages (DATA,d) and (DATA&ACK,d) are equivalent; let their image be denoted by (DATA´,d). Thus $M_1' = \{(DATA',d) : d \in DATASET\}$

Similarly, the following can be shown about messages in M_2. All (DATA,d) messages have the null image. All (DATA&ACK,d) messages are equivalent to the ACK message; denote their image by (ACK´). Thus $M_2' = \{(ACK')\}$.

The messages and events of the image protocol for the projected function are shown in Tables 2 and 3.

For this image protocol, assuming error-free channels C_1 and C_2, the following assertions stating some logical correctness properties of the projected one-way data transfer function have been found (by inspection):

1. SINK[i] = SOURCE[i] for $0 \leq i <$ VR.

2. VS $\geq$ VR $\geq$ VS-1.

3. (DATA´,d) in CHANNEL1 => (D_OUT)
   and (d = SOURCE[VS-1])
   and (exactly one DATA´ message in CHANNEL1)
   and (not ACK_DUE) and (VS = VR + 1)
   and (no ACK´ message in CHANNEL2).

4. ACK_DUE => (D_OUT)
   and (no DATA´ message in CHANNEL1)
   and (VS = VR) and (no ACK´ message in CHANNEL2)

5. ACK´ in CHANNEL2 => (D_OUT)
   and (no DATA´ message in CHANNEL1) and (VS = VR)
   and (not ACK_DUE)
   and (exactly one ACK´ message in CHANNEL2)

6. not D_OUT => VS = VR

The image protocol can be shown to be well-formed. As a result, the above assertions are valid for the original full-duplex data transfer protocol by virtue of the theorems in [8,9].

Error-free channels have been assumed to make the protocol example small. As shown in [8,9], the method of protocol projections is applicable to protocol models in which a channel may lose, duplicate and reorder messages. An example involving channels with errors is described in the companion paper [11].

5. CONCLUSIONS

We have employed two protocol examples, a pair of interacting finite state machines and a full-duplex data transfer protocol described by a programming language model, to illustrate the method of protocol projections. We have shown how to construct the entity states, messages and events of an image protocol from the corresponding quantities of a given protocol. Since image protocols of a well-structured multi-function protocol are much simpler than the original protocol, they can be analyzed more easily using available techniques. In our examples, the image protocols constructed are simple enough that various assertions describing the logical correctness properties of the image protocols have been obtained by inspection. The image protocols can be shown to be well-formed. As a result, the assertions obtained are also valid for the original protocol examples.

The consideration of a protocol model with only two entities is strictly for notational simplicity. Our main results (Theorems 1 and 2 in [7,8]) are applicable to any network of protocol entities interconnected by communication channels. The global state of the general protocol model is again specified by a joint description of the states of the protocol entities and communication channels. The definitions of message equivalence, event equivalence, and well-formed image events are simply restated to cover all protocol entities in the model [10].

REFERENCES

[1] Bochmann, G. V. and C. A. Sunshine, "Formal Methods in Communication Protocol Design," IEEE Trans. on Commun, April 1980.

[2] Zafiropulo, P., et al., "Towards Analyzing and Synthesizing Protocols," IEEE Trans. on Commun., April 1980.

[3] Stenning, N. V., "A Data Transfer Protocol," Computer Networks, September 1976.

[4] Hailpern, B. T. and S. S. Owicki, "Verifying Network Protocols using Temporal Logic," Computer Systems Laboratories, Stanford Univ., Technical Report No. 192, June 1980.

[5] Brand, D. and W. H. Joyner, "Verification of Protocols using Symbolic Execution," Computer Networks, September/October 1978.

[6] Keller, R. M., "Formal Verification of Parallel Programs," Comm. ACM, July 1976.

[7] Lam, S. S. and A. U. Shankar, "Protocol Projections: A Method for Analyzing Communication Protocols," Conf. Rec. National Telecommunications Conference, New Orleans, November 1981.

[8] Lam, S. S. and A. U. Shankar, "Verification of Communication Protocols via Protocol Projections," Proc. INFOCOM '82, Las Vegas, April 1982.

[9] Lam, S. S. and A. U. Shankar, "Protocol Verification via Protocol Projections, Part 1: Theory," Dept. of Computer Sciences, Univ. of Texas at Austin, Technical Report (in preparation).

[10] Shankar, A. U., "Analysis of Communication Protocols via Protocol Projections," Ph.D. Thesis, Dept. of Electrical Engineering, Univ. of Texas at Austin, 1982 (in preparation).

[11] Shankar, A. U. and S. S. Lam, "On Time-dependent Communication Protocols and their Projections," Proc. Second International Workshop on Protocol Specification, Testing, and Verification, Idyllwild, California, May 1982.

[12] Brand, D. and P. Zafiropulo, "On Communicating Finite-State Machines," IBM Res. Report RZ1053, Zurich, Switzerland, Jan. 1981.

ACKNOWLEDGEMENT

The research in this paper was supported by National Science Foundation Grant No. ECS78-01803.

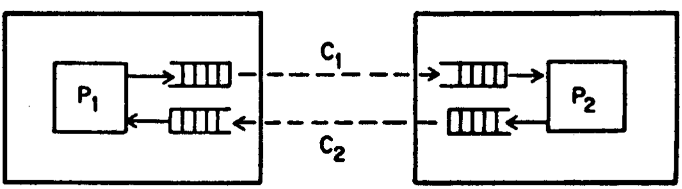


Fig. 1. Components of the protocol model.

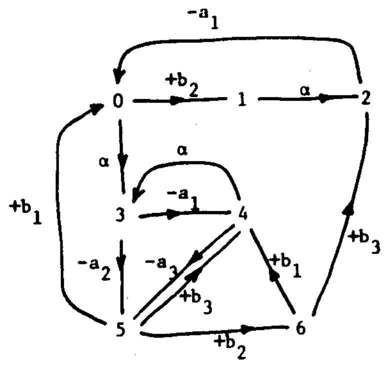


Fig. 2. Two communicating finite state machines.

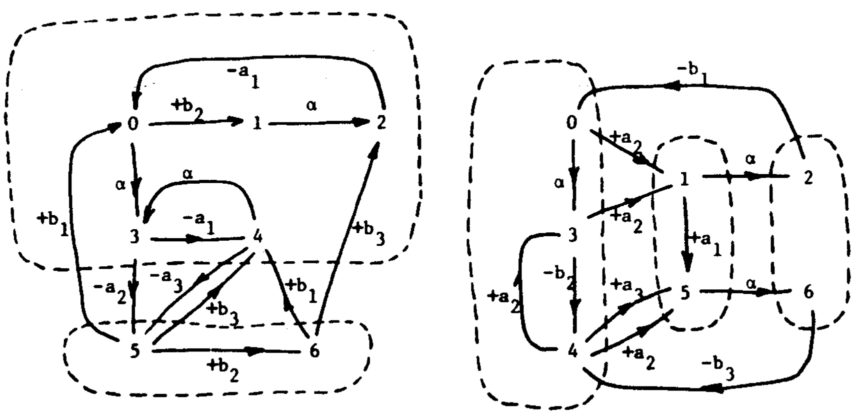
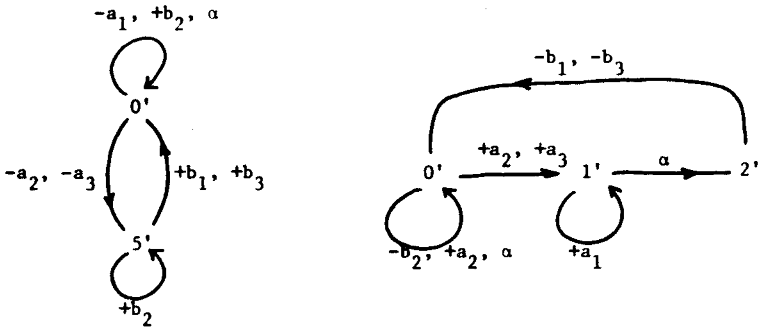
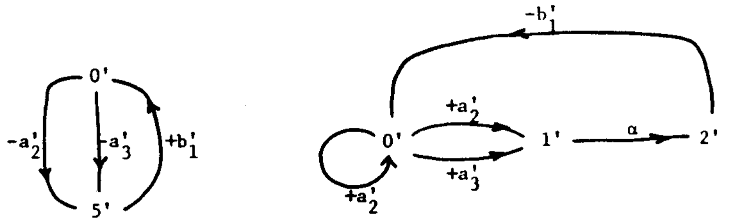


Fig. 3. Classes of equivalent entity states.



(a) (b)

Fig. 4. Transitions in image state spaces.



(a) (b)

Fig. 5. An image protocol of the communicating finite state machines.

Event Name	Enabling Condition	Action
1. SEND_DATA	not BUSY and not D_OUT	SDATA := SOURCE[VS]; put(CHANNEL1, (DATA,SDATA)); VS := VS + 1; D_OUT := true
2. SEND_DATA&ACK	not BUSY and not D_OUT and ACK_DUE	SDATA := SOURCE[VS]; put(CHANNEL1, (DATA&ACK,SDATA)); VS := VS + 1; D_OUT := true; ACK_DUE := false
3. SEND_ACK	not BUSY and ACK_DUE	put(CHANNEL1, (ACK)); ACK_DUE := false
4. START_BUSY	not BUSY	BUSY := true
5. STOP_BUSY	BUSY	BUSY := false
6. REC_DATA	first(CHANNEL2) = DATA	get(CHANNEL2, (DATA,RDATA)); SINK[VR] := RDATA; VR := VR + 1; ACK_DUE := true
7. REC_DATA&ACK	first(CHANNEL2) = DATA	get(CHANNEL2, (DATA&ACK,RDATA)); SINK[VR] := RDATA; VR := VR + 1; ACK_DUE := true; D_OUT := false
8. REC_ACK	first(CHANNEL2) = ACK	get(CHANNEL2, (ACK)); D_OUT := false

TABLE 1. Events of entity P_1 in the full-duplex data transfer protocol.

Event Name	Enabling Condition	Action
1. SEND_DATA′	not BUSY and not D_OUT	SDATA := SOURCE[VS]; put(CHANNEL1, (DATA′, SDATA)); VS := VS + 1; D_OUT := true
2. START_BUSY′	not BUSY	BUSY := true
3. STOP_BUSY′	BUSY	BUSY := false
4. REC_ACK′	first(CHANNEL2) = ACK′	get(CHANNEL2, (ACK′)); D_OUT := false

TABLE 2. Events of P_1' in the image protocol for one-way data transfer

Event Name	Enabling Condition	Action
1. REC_DATA′	first(CHANNEL1) = DATA′	get(CHANNEL1, (DATA′, RDATA)); SINK[VR] := RDATA; VR := VR + 1; ACK_DUE := true
2. START_BUSY′	not BUSY	BUSY := true
3. STOP_BUSY′	BUSY	BUSY := false
4. SEND_ACK′	not BUSY and ACK_DUE	put(CHANNEL1, (ACK′)); ACK_DUE := false

TABLE 3. Events of P_2' in the image protocol for one-way data transfer.