PROTOCOL VERIFICATION VIA PROJECTIONS*

Simon S. Lam and A. Udaya Shankar†

TR-207 August 1982
Revised, June 1983

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
Tel: (512) 471-4353

*This work was supported by National Science Foundation Grants No. ECS78-01803 and No. ECS-8304734.

†Current address: Department of Computer Science, University of Maryland, College Park, Maryland 20742. Tel: (301) 454-4245.

ABSTRACT

The method of projections is a new approach to reduce the complexity of analyzing nontrivial communication protocols. A protocol system consists of a network of protocol entities and communication channels. Protocol entities interact by exchanging messages through channels; messages in transit may be lost, duplicated as well as reordered. Our method is intended for protocols with several distinguishable functions. We show how to construct image protocols for each function. An image protocol is specified just like a real protocol. An image protocol system is said to be faithful if it preserves all safety and liveness properties of the original protocol system concerning the projected function. An image protocol is smaller than the original protocol and can typically be more easily analyzed. Two protocol examples are employed herein to illustrate our method. An application of this method to verify a version of the High-level Data Link Control (HDLC) protocol is described in a companion paper.

Index Terms: Communication protocols, verification, method of projections, image protocols, protocol analysis, message-passing networks, communicating processes, distributed systems.

Table of Contents

1. INTRODUCTION

Most real-life communication protocols are very complex because they typically have to perform several distinct functions. For example, the High-level Data Link Control (HDLC) protocol has at least three functions: connection management and one-way data transfers in two directions [10, 20]. To reduce the complexity of analyzing such a multi-function protocol, an approach that appears attractive is to decompose each protocol entity into modules for handling the different functions of the protocol. For example, each protocol entity in HDLC may be decomposed into three functional modules such as shown in Figure 1. Each module communicates with a corresponding module in the other protocol entity to accomplish one of the three functions [1]. However, the decomposition approach does not seem to facilitate an analysis of the protocol. The main difficulty is that significant interaction exists among the modules. We identify two types of dependencies. First, modules interact through *shared variables* within an entity. Second, they also interact because data and control messages sent by different modules in one entity to their respective modules in the other entity are typically encoded in the same protocol message (*shared messages*).

Most communication protocols that have been rigorously analyzed and presented in the literature are concerned with a single function: either a connection management function [2, 11, 16] or a one-way data transfer function [6, 9, 23]. A one-way data transfer protocol, for example, corresponds to the interaction of a data send module and a data receive module in isolation (see Figure 1). Any interaction between these modules and other modules are not accounted for. As such, it constitutes just one function of a real-life protocol. The following question arises: Are the safety and liveness properties that are proved for the one-way data transfer protocol still valid when it is implemented as part of a multi-function protocol with the two types of dependencies mentioned above?

The method of projections provides an approach to transform the analysis of a multi-function protocol into analyses of smaller single-function protocols, called image protocols. It is different from the straightforward approach of decomposing protocol entities into functional modules. Image protocols are obtained by treating groups of entity states, messages and events in the original protocol as equivalent and aggregating them. An image protocol is specified and verified just like a real protocol.

1.1 The Abstract Model

The underlying abstract model for the development of our theory uses states to encode meaning [3, 17]. Our results are thus applicable to two widely used protocol specification formalisms: state machines [24] and programming language descriptions [1, 8, 20]. (We illustrate below the application to each formalism with one example.)

A protocol system consists of a network of protocol entities and channels. At any

time, the *global state* of the system is specified by a joint description of the states of the entities and channels. Let G denote the set of all global states of the protocol system. The states of entities and channels (and hence the global state) may change due to the occurrence of certain events: entities sending messages, entities receiving messages, timeouts, channel errors, etc. These global state transitions define a directed graph on G. Given any initial global state g_0, the portion of the graph that is reachable from g_0 is referred to as the *reachability graph* R. R contains all available information on the logical properties of the protocol system.

Specifically, assertions of liveness properties are predicates on the set of paths in G. A liveness assertion is valid if it is satisfied by the paths in R. Assertions of safety properties are predicates on G. Let R_s denote the set of states reachable from g_0. A safety assertion is valid if it is satisfied by the states in R_s.

Verification of these properties may be carried out by a brute-force state exploration (in the case of a small finite R), or by proof techniques for parallel programs. The method of projections can be used in conjunction with either verification approach.

1.2 The Projection Approach

Consider a protocol with several distinguishable functions. We would like to ask questions regarding the logical behavior of the protocol system 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 projection idea can be illustrated by the picture in Figure 2. Consider a protocol system with the state description (x,y,z) and the set R_s of reachable states. Suppose that we are interested in a safety assertion that involves only the variables x and y. To determine whether the assertion is true, it is sufficient 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 any protocol analysis. Our analysis approach avoids a characterization of R. Instead, we construct from the given protocol an image protocol for each of the functions that are of interest to us (to be referred to as the *projected functions*).

An *image protocol* is specified just like any real protocol. The states, messages and events of entities in an image protocol are obtained by aggregating groups of states, messages and events of the corresponding entities in the original protocol. Definitions needed for the construction of an image protocol are presented in Section 3.

Given an image protocol, suppose that a second image protocol is obtained by aggregating some of the entity states, messages and events of the first one. We say that the second image protocol has a lower *resolution* than the first image protocol. The original protocol can be thought of as an image protocol of itself, and obviously it has the highest resolution available.

Due to the aggregations, an image protocol is smaller than the original protocol and is typically easier to analyze. However, the reachability graph of an image protocol captures only part of the logical behavior of the original protocol. The following useful properties of image protocols are proved in Section 4.

First, any safety property that holds for an image protocol must also hold for the original protocol. Second, if an image protocol is constructed with sufficient resolution so that its events satisfy a well-formed property, then it is *faithful*: Any logical property, safety or liveness, that can be stated for the image protocol holds in the image protocol *if and only if* it holds in the original protocol. The well-formedness of an image protocol is determined by checking protocol entities individually. The well-formed property is the weakest sufficient condition for faithfulness that can be stated without any knowledge of R.

Given a protocol and an assertion A_0, stating the desired logical behavior of the protocol for the function being projected, our objective is to construct the smallest image protocol that is of sufficient resolution to verify A_0. Towards this goal, we construct a sequence of image protocols of increasing resolution by a stepwise refinement process. The initial image protocol can be determined by the resolution needed to *describe* A_0. The stepwise refinement is terminated when an image protocol with sufficient resolution to *verify* A_0 is constructed. Two stepwise refinement algorithms are presented in Section 5 with termination conditions based upon the two image protocol properties mentioned above.

Given a multi-function protocol, a faithful image protocol can always be obtained for each function by adjusting its resolution. However, the successful construction of faithful image protocols that are much smaller than the original protocol depends upon whether the protocol has a good structure. Thus, one can think of a multi-function protocol as being *well-structured* if it possesses small faithful image protocols for its functions.

Our protocol system model is described in Section 2. Various extensions to this model can be accommodated by our theory (e.g., broadcast channels, message priority classes). To streamline our presentation below, we postpone these extensions to Appendix I. Proofs of lemmas, theorems and corollaries are given in Appendix II.

Two examples are employed throughout Sections 2-5 for illustration. In the first example, the protocol entities are specified using a finite state machine description. The second example is a full-duplex data transfer protocol. Its entities are specified using a programming language description.

1.3 Related Work

The idea of projection is similar to various notions of abstraction in the design and analysis of software systems and programming languages. Within the communication protocols literature, the term "protocol projection" has been used by Bochmann and Merlin to describe an operation in their method for protocol synthesis [4]. Their basic idea of "projection onto the relevant action" is similar to ours herein, but the development and application of the idea in their work and ours are different. The idea of projection as applied to various special cases of our protocol system model has been previously explored in [12, 13, 14, 18, 19]. Extension of our results to a more structured model of protocol entities is presented in [21]. Efficient rules for constructing image protocols as well as the modeling of timing relationships in protocol systems are addressed therein. An application of the method of projections to verify a version of the HDLC protocol is presented in [20].

2. THE PROTOCOL SYSTEM MODEL

Let there be I protocol entities $P_1,P_2,...,P_I$ and K channels $C_1,C_2,...,C_K$. Let S_i be the set of states of P_i, and M_{ik} be the set of messages that P_i can send into C_k. For notational convenience, we shall assume that the message sets M_{ik}, for all i and k, are mutually exclusive. (In practice, this is achieved by encoding each entity's unique address in its messages.)

Each protocol entity can send messages into a subset of channels, called its *outgoing set* of channels. Each protocol entity can receive messages from a subset of channels, called its *incoming set* of channels. We do not require the outgoing and incoming channel sets of a protocol entity to be mutually exclusive. We consider channels that have a single destination protocol entity, although the number of protocol entities that can send into a channel can be more than one. (An extension of the model to include multi-destination or broadcast channels is given in Appendix I.)

A channel can be real or logical. In any case, all buffers and communication media between two entities connected by a channel are considered to be part of the channel. Hence, channels may have large storage capacities for messages in transit. At any time, a channel contains a (possibly empty) sequence of messages. We further distinguish channels into the following categories:

(1) Infinite-buffer channels--most communication protocols have some measure of flow control. As a result, their buffer requirements for messages in transit between two entities are bounded. Hence the assumption of an infinite buffering capacity is equivalent to being able to satisfy these buffer requirements.

(2) Finite-buffer blocking channels--with a blocking channel, protocol entities are not permitted to send into the channel whenever the channel is full.

(3) Finite-buffer loss channels--with a loss channel, the sending of a message into a full channel results in the instantaneous *bumping* (deletion) of a message. The message bumped may be the new message or a message already in the channel.

Bumping rule. Whenever bumping is necessary, the selection of a message to delete may be specified by any rule that depends only upon the channel positions of messages.

Define the set of messages that can be in channel C_k to be

$$M_k = \bigcup_{i=1}^{I} M_{ik}$$

The set of all possible sequences in C_k is a subset of

$$\underline{M}_k = \left(\bigcup_{j=1}^{J} M_k^j \right) \cup \{<>\}$$

where $<>$ denotes the null sequence, M_k^j is the Cartesian product of M_k with itself j times, and J is the maximum number (possibly infinite) of messages that C_k can accommodate. If we think of C_k as having J buffers, then any message sequence of length j in C_k occupies the first j buffers.

The global state space of the protocol system is

$$G = (S_1 \times S_2 \times \ldots \times S_I) \times (\underline{M}_1 \times \underline{M}_2 \times \ldots \times \underline{M}_K)$$

Each global state in G is an (I+K)-tuple

$$(s_1, s_2, \ldots, s_I; \underline{m}_1, \underline{m}_2, \ldots, \underline{m}_K)$$

where $s_i \in S_i$ for each i and $\underline{m}_k \in \underline{M}_k$ for each k.

The dynamics of protocol entities and channels are described by events. Each event is specified by its enabling condition and its action. The *enabling condition* of an event is a predicate in the components of the global state of the protocol system. The action of an event specifies an update to the components of the global state. An event can occur only if the protocol system is in a state where the enabling predicate of the event is true. Its occurrence, consisting of the execution of its action, is treated as an indivisible transition from one global state to another. Each event corresponds to a set of transitions in the global state space G. The union of these sets of transitions over all events specified for the protocol system is denoted by τ. We make the assumption that the simultaneous occurrence of multiple events in the protocol system can be treated as occurrences of the same events in some arbitrary order.

Both entities and channels behave as event-driven processes: i.e. their life cycle consists of a continuous series of events. We will next describe the types of events in our (abstract) protocol system model.

2.1 Entity Events

There are three types of entity events. The events of entity P_i are specified as follows:

(1) Send events
Let (s, r, -m) denote the event of P_i sending message m into channel C_k where $m \in M_{ik}$ and C_k is in the outgoing channel set of P_i. This send event is enabled when P_i is in state s. With a finite-buffer blocking channel, it is also required that the channel is not full. After the event occurrence, P_i is in state r and m has been appended to the end of the message sequence in C_k. If C_k is a finite-buffer loss channel and is already full, then a message is bumped instantaneously according to the bumping rule of C_k. The set of such send events that can be specified is a subset of $S_i \times S_i \times M_{ik}$.

(2) Receive events
Let (s, r, +m) denote the event of P_i receiving message m from channel C_k where $m \in M_k$ and C_k is in the incoming channel set of P_i. This receive event is enabled when P_i is in state s and m is the first message in channel C_k. After the event occurrence, P_i is in state r and m is deleted from the channel. The set of such receive events that can be specified is a subset of $S_i \times S_i \times M_k$.

(3) Internal events
Let (s, r, α) denote an internal event of P_i where α is a special symbol indicating the absence of a message. This internal event is enabled when P_i is in state s. After the event occurrence, P_i is in state r. Internal events model timeout occurrences internal to P_i, as well as interactions between the entity and its local user. (The latter can be treated as internal events if message exchanges between P_i and its local user are not explicitly modeled.) The set of internal events that can be specified is a subset of $S_i \times S_i \times \{\alpha\}$.

Note that each send or receive event may alter the states of the entity and channel involved in the event. Internal entity events do not affect the state of any channel. We shall use T_i to denote the set of events specified for P_i, i=1,2,...,I. Note that the behavior of P_i may be nondeterministic. For example, we may specify T_i to contain both (s, r, +m) and (s, u, +m) where $r \neq u$.

2.2 Channel Events

Let E_k denote the set of channel events specified for channel C_k. The occurrence of a channel event in E_k depends on and changes only the state of C_k; no other channel or protocol entity is involved. We use such channel events to model various types of channel errors.

The following three types of channel error events are considered: loss events, duplication events, and reordering events. Each event is defined for a position or pair of positions in C_k. Let the sequence of messages in C_k be $\underline{m}_k$. A loss event for the i-th position of C_k is enabled if $\underline{m}_k$ has a message in that position; its occurrence deletes that message from $\underline{m}_k$. A duplication event for the i-th position of C_k is enabled if $\underline{m}_k$ has a message in that position; its occurrence inserts a duplicate of that message immediately behind it in $\underline{m}_k$. A reordering event from the i-th to the j-th position of C_k is enabled if $\underline{m}_k$ has messages in both these positions; its occurrence moves the message in the i-th position to immediately behind the message in the j-th position.

E_k can be specified to contain any collection of events of each type.

For consistency, we require that all duplication events are inhibited if C_k is a finite-buffer blocking channel and it is full. If C_k is a finite-buffer loss channel, the bumping rule applies when a duplication event results in a message sequence whose length exceeds the buffering capacity of the channel.

2.3 Paths in the Global State Space

The protocol system is completely specified by the following:

$$S_i,\ M_{ik},\ T_i,\ E_k,\ g_0 \quad \text{for } i=1,2,\ldots,I,\ k=1,2,\ldots,K \tag{1}$$

where g_0 denotes the initial global state of the protocol system. Define

$$E = \left(\bigcup_{i=1}^{I} T_i\right) \cup \left(\bigcup_{k=1}^{K} E_k\right)$$

Given event $e \in E$ and global states g and h (not necessarily distinct) we say that e — *can take* the protocol system from g to h, if e is enabled when the protocol system is at state g and its occurrence results in the protocol system entering state h.

Recall that τ is the set of global state transitions due to the events in E. The pair (G,τ) defines a directed graph whose nodes are elements of G and whose arcs are elements of τ. We now formally define a *path in* (G, τ) to be a sequence of global states $f_0,f_1,\ldots,f_n$ in G, such that there exist events $e_1,e_2,\ldots,e_n$ in E and e_i can take the protocol system from f_{i-1} to f_i for $i=1,2,\ldots,n$. Since R denotes that portion of the graph (G, τ) that is reachable from the initial global state g_0, a *path in* R is a path in (G, τ) that starts from the initial state g_0. The *length of a path* is defined to be the number of elements (global states) in it. For notational convenience, we consider any global state $g \in G$ as a path of length 1 in (G, τ). Thus, g_0 is the only path of length 1 in R.

Given two sequences $x = f_0,f_1,\ldots,f_n$ and $y = g_0,g_1,\ldots,g_m$, we denote by x,y the sequence $f_0,f_1,\ldots,f_n,g_0,g_1,\ldots,g_m$ obtained by concatenating x and y. A path x is *extendable* by sequence y to $z = x,y$ if z is also a path.

2.4 Two Protocol Examples

Finite state machines example

We first illustrate the specification of protocol entities with an example of two interacting finite state machines. (See Figure 3.) Let C_1 and C_2 be two channels connecting P_1 to P_2, and P_2 to P_1 respectively. Protocol entity P_1 has state space $S_1 = \{0,1,2,3,4,5,6\}$ and sends messages into channel C_1 from $M_1 = \{a_1,a_2,a_3\}$. P_1 receives messages from C_2. Protocol entity P_2 has state space $S_2 = \{0,1,2,3,4,5,6\}$ and sends messages into channel C_2 from $M_2 = \{b_1,b_2,b_3\}$. P_2 receives messages from C_1. The events of entity P_1 are shown in Figure 3(a). An arc from node i to node j with a label x specifies event (i,j,x), where x is α for an internal event, x is $-a_i$ for sending message a_i, and x is $+b_i$ for receiving message b_i. The events of entity P_2 are similarly shown in Figure 3(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, and is useful for representing certain features in real protocol systems.

We have not given meaning to the protocol entity states and messages. The example in Figure 3 will be used expressly to illustrate definitions for the construction of image protocols to be presented in Sections 3 and 4.

Full-duplex data transfer example

We next illustrate the specification of protocol entities using a programming language model. A full-duplex data transfer protocol is considered.

Let C_1 and C_2 be infinite-buffer channels connecting P_1 to P_2, and P_2 to P_1 respectively. Let variables CHANNEL1 and CHANNEL2 denote the sequence of messages in C_1 and C_2 respectively.

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. (SOURCE and SINK should be interpreted as history variables that are not actually implemented.) Additionally, the following variables are used in P_1: VS and VR which are non-negative 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 indicating that P_1 is given access to send into channel C_1.

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 type of message, and it can take the (character string) values DATA, ACK and DATA&ACK, corresponding to three message types. 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

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

The message set M_2 for P_2 is the same as M_1; each message has a sender identity field which will not be explicitly indicated whenever there is no ambiguity.

The set of entity events for P_1 is presented in Table 1. (Note that each event in Table 1 corresponds to a collection of events as defined in Section 2.1.) Events 1-3 are send events for the 3 message types of M_1. Events 6-8 are receive events for the 3 message types of M_2. Events 4 and 5 are internal events caused by an agent locally connected to P_1 but which is not explicitly modeled (e.g., a 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 system 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.

We will come back to both of the above examples when we use them to illustrate the construction of image protocols in the following section.

3. CONSTRUCTING AN IMAGE PROTOCOL SYSTEM

An image protocol system is constructed by first partitioning each of the sets

S_i, M_{ik}, T_i, E_k, for all i and k

in the original protocol system specification. Protocol quantities (entity states, messages or events) in each partition subset are *treated as equivalent* and are *aggregated* to form a single quantity, called their *image*, in the image protocol system. Since partition subsets are mutually exclusive and collectively exhaustive, quantities that are treated as equivalent have the same image; quantities not treated as equivalent have different images.

For a protocol quantity x, we use x' to denote its image. The same notation x' will also be used to refer to the set of protocol quantities in the original protocol system that have the x' image. For a set A of protocol quantities in the original system specification, we use A' to denote the set of image quantities derived from A, i.e., $A' = \{x' : x \in A\}$. Note that an image protocol system, defined by

$$S_i', M_{ik}', T_i', E_k', g_0' \quad \text{for } i = 1,2,\ldots,I \quad k = 1,2,\ldots,K \tag{2}$$

is specified just like any real protocol system. We cannot tell from inspecting (2) that it is an image of another protocol system.

Since an image protocol is obtained from the original protocol by aggregations, it captures only part of the logical behavior of the original protocol system. First, global states of the image protocol system correspond to aggregations of global states of the original protocol system. Second, in the global state space of the image protocol system, the observable effect of different events in the original protocol system may be identical (these events will become the same image event) or nil (these will be eliminated).

Section 3 is devoted to definitions needed for the construction of an image protocol given a partitioning of the entity state spaces of the original protocol. Stepwise refinement algorithms for obtaining an image protocol of sufficient resolution to verify some logical assertion are presented in Section 5. Properties of image properties are presented in Section 4.

3.1 Aggregation of Entity States

We start with a given partitioning of the state space S_i of protocol entity P_i, for all i. All entity states in a partition subset are aggregated to the same image. Let S_i' denote the set of images of states in S_i, i.e., $S_i' = \{s' : s \in S_i\}$. S_i' is said to be the image state space of P_i. Elements of S_i' are also referred to as *image entity states.*

Note that if a protocol entity does not participate in the projected function, then the entire state space of that protocol entity may be aggregated to a single degenerate image state.

Finite state machines example (cont.)

Consider the example in Figure 3. Suppose that the partition { {0,1,2,3,4}, {5,6} } of S_1 has been chosen; see Figure 4(a). Let image state 0' denote the image of states 0, 1, 2, 3 and 4 in S_1. Let image state 5' denote 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 image protocol, suppose that the partition { {0,3,4}, {1,5}, {2,6} } of S_2 has been chosen; see Figure 4(b). Let image states 0' 1' and 2' respectively denote the images of the states in the partition subsets {0,3,4}, {1,5} and {2,6}. Then, the image state space of P_2 is $S_2' = \{0',1',2'\}$.

Full-duplex data transfer example (cont.)

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 are encountered when one attempts to decompose protocol entities into functional modules. First, the variable BUSY is shared by both functions of the protocol. Second, messages of type DATA&ACK are also shared. Such dependencies present difficulties for protocol analysis using a decomposition approach, but not when using a projection approach.

Consider the function of one-way data transfer from P_1 to P_2. Two desirable properties of the protocol with respect to this function may be stated as follows:

DP1 : $SINK_2[i] = SOURCE_1[i]$ for $0 \leq i < VR_2$

DP2 : $VS_1 \geq VR_2 \geq VS_1 - 1$

where the subscripts indicate which entity the variables belong to. This assertion can be described using just the variables SOURCE and VS in P_1 and the variables SINK and VR in P_2. However, image protocol entities specified using only these variables do not have sufficient resolution to verify the above assertion. The necessary resolution is obtained by a stepwise refinement procedure (see Section 5). For now, suppose that we retain the following variables in the image protocol: VS, D_OUT and SOURCE in P_1, and VR, ACK_DUE and SINK in P_2.

At any time, the image state of P_1 is given by the value of <VS, D_OUT, SOURCE>. 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, SINK and BUSY.

At any time, the image state of P_2 is given by the value of <VR, ACK_DUE, SINK>. 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, SOURCE and BUSY.

3.2 Aggregation of Messages

The partitioning and aggregation of messages in the message set M_{ik} depend upon the partitioning of the entity state spaces. Two messages m and n may be treated as equivalent only if their receptions cause identical state changes in the image state space of the receiver of C_k. This equivalence relation partitions M_{ik} and messages within the same partition subset may be aggregated to form an *image message*. The image message sets are given by

$$M'_{ik} = \{m' : m \in M_{ik}\} \quad \text{for all i and k}$$

In particular, messages in M_{ik} whose receptions do not cause any state change in the image state space of the receiver are said to have a *null image*. The null image message in M'_{ik} is denoted by β. Define

$$M'_k = \bigcup_{i=1}^{I} M'_{ik}.$$

The null image messages from different M'_{ik} are all denoted by β and not distinguished in M'_k.

Sometimes it is desirable to have a finer partitioning of a message set. A higher resolution may be needed to describe and/or verify the behavior of the protocol system. A stronger criterion for treating messages as equivalent is to require also that their send events cause identical state changes in the image state space of their sender.

We shall deal with a null image message β in two different ways depending upon whether C_k is a finite-buffer channel or an infinite-buffer channel. With a finite-buffer channel, β is included in M'_k, as described above. If C_k is an infinite-buffer channel, it is possible to ignore β because even though β occupies a buffer in C_k, it can never cause C_k to be full. Specifically, β may be excluded from M'_k if C_k is an infinite-buffer channel and is characterized by the following *uniform error model*:

a) Either E_k contains no loss events, or the only loss event in E_k is for the first position in C_k, or E_k contains a loss event for every position in C_k.

b) Either E_k contains no duplication events, or E_k contains a duplication event for every position in C_k.

c) Either E_k contains no reordering events, or E_k contains a reordering event for every pair of positions in C_k.

The uniform error model is a realistic characterization of most conventional physical and logical communication channels. Henceforth in this paper, an infinite-buffer channel C_k will be considered to be characterized by the uniform error model and β will be deleted from M_k'.

If E_k does not satisfy the uniform error model then C_k is said to be characterized by a nonuniform error model.

Finite state machines example (cont.)

In Figures 4(a) and 4(b), if we relabel each state by its image and collapse all identically labelled states, without deleting any events, then we obtain Figures 5(a) and 5(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. Assume that C_1 and C_2 have adequate buffers (infinite-buffer channels).

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 treated as equivalent to a_3 because a_2 causes state transition (0',0') in S_2' while a_3 does not. Thus, we partition M_1 to be $\{\{a_1\}, \{a_2\}, \{a_3\}\}$. The image of a_1 is null. Let the image messages a_2' and a_3' denote the images of 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'. We will treat b_1 as 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. Hence, we partition M_2 to be $\{ \{b_1, b_3\}, \{b_2\} \}$. The image of b_2 is null. Let image message b_1' denote the image of messages b_1 and b_3. The image message set M_2' is $\{b_1'\}$.

Full-duplex data transfer example (cont.)

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 (see [14]). The message (ACK) has a null image. The messages (DATA,d) and (DATA&ACK,d) may be treated as equivalent; let their image be denoted by (DATA',d). Thus $M_1' = \{(\text{DATA}',d) : d \in \text{DATASET}\}$.

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

3.3 Images of Channel States

The preceding equivalence relation defined for messages is now extended to channel states. The image $\underline{m}'_k$ of channel state $\underline{m}_k$ is obtained by taking the image of each message in $\underline{m}_k$. For infinite-buffer channels, messages with a null image are deleted from $\underline{m}_k$ to form $\underline{m}'_k$. All channel states with the same image are treated as equivalent. Let $\underline{M}'_k$ denote the set of images of channel states in $\underline{M}_k$. Then

$$\underline{M}'_k = \left(\bigcup_{j=1}^{J} M_k'^{j} \right) \cup \{<>\}$$

where $M_k'^{j}$ denotes the cartesian product of M'_k with itself j times, and J is the number of buffers in C_k (possibly infinite). The image of a channel state is also referred to as an *image channel state*. $\underline{M}'_k$ is said to be the image state space of C_k.

3.4 Images of Global States

The above equivalence relations defined for entity states and channel states are now extended to global states. The image of global state $g = (s_1,s_2,...,s_I;\underline{m}_1,\underline{m}_2,...\underline{m}_K)$ is defined to be $g' = (s'_1,s'_2,...,s'_I;\underline{m}'_1,\underline{m}'_2,...,\underline{m}'_K)$. All global states with the same image are treated as equivalent. Let G' denote the set of images of global states in G. Thus, we have

$$G' = (S'_1 \times S'_2 \times \ldots \times S'_I) \times (\underline{M}'_1 \times \underline{M}'_2 \times \ldots \times \underline{M}'_K).$$

The image of a global state is also referred to as an *image global state*. G' is said to be the image global state space of the protocol system.

3.5 Image of a Sequence of Global States

The preceding equivalence relation defined for global states is now extended to sequences of global states in G. Given w, a sequence of global states in G, its image w' is obtained as follows: first, take the image of each global state in w; second, any consecutive occurrences of the same image state in the sequence are replaced by a single occurrence of the image state. All sequences having the same image are treated as equivalent. Recall that paths in (G, τ) are sequences of global states in G. Thus, the image of a path in (G, τ) is defined as above for sequences.

3.6 Aggregation of Entity Events

The above equivalence relations defined for entity states and messages are now extended to entity events. Entity events that have the same observable effect in the image global state space are aggregated to the same *image entity event*. The images of the three types of entity events (s,r,+m), (s,r,-m) and (s,r,α) are (s',r',+m'), (s',r',-m') and (s',r',α) respectively.

An image event whose occurrence does not have any observable effect in the image

global state space is said to be a *null image event.* An image internal event (s',r',α) is a null image event if s' = r'. Image send events involving null image messages and infinite-buffer channels are treated as image internal events in T_i'; in this case the image event (s',r',-β) is represented as (s',r',α), and it is a null image event if s' = r'. Finally, image receive events involving null image messages must have s' = r' by definition; hence such receive events are null image events if the channel involved is an infinite-buffer channel.

The set of image entity events for P_i is defined by

T_i' = {(s',r',x') : (s,r,**x**) ϵ T_i where x = +m, -m or α,
and the image of (s,r,x) is not a null image event}

Finite state machines example (cont.)

Recall that messages a_1 and b_2 have null images. From Figures 4(a) and 5(a), the events in P_1 having null images are (0,1,+b_2), (1,2,α), (2,0,-a_1), (0,3,α), (3,4,-a_1), (4,3,α), 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 treated as 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 4(b) and 5(b), the events in P_2 having null images are (0,3,α), (3,4,-b_2), and (1,5,+a_1). The image of (4,3,+a_2) is (0',0',+a_2'). Events (0,1,+a_2), (3,1,+a_2), and (4,5,+a_2) are treated as equivalent and have the image (0',1',+a_2'). Event (4,5,+a_3) has the image (0',1',+a_3'). Events (1,2,α) and (5,6,α) have the image (1',2',α). Events (2,0,-b_1) and (6,4,-b_3) have the image (2',0',-b_1'). Therefore,

T_2' = {(0',0',+a_2'), (0',1',+a_2'), (0',1',+a_3'), (1',2',α), (2',0',-b_1')}.

Full-duplex data transfer example (cont.)

The entity events of the image protocol for the projected function of one-way data transfer from P_1 to P_2 are shown in Tables 2 and 3. They can be derived using the above definition as illustrated in [14, 21]. In Table 2 for example, SEND_DATA' is the image of events SEND_DATA and SEND_DATA&ACK of P_1; REC_ACK' is the image of REC_ACK and REC_DATA&ACK. The events SEND_ACK, START_BUSY, STOP_BUSY and REC_DATA of P_1 have null images.

3.7 Channel Events

The image of a channel event should describe the effect of the event that can be observed in the image channel state space. We now show that for every channel event e in E_k, we need to specify an image channel event e' in E_k' which is identical to e. Hence, we do not aggregate events in E_k. We need to consider two possible cases.

First, C_k is a finite-buffer channel (null image messages are included in M_k'). Let

$\underline{m}_k$ be a sequence of messages in C_k. If we observe in the image channel state space an error event occurring to $\underline{m}_k$, its effect is identical to that of the same channel event occurring to $\underline{m}'_k$. Hence, for a finite-buffer channel, E'_k must be identical to E_k.

Second, C_k is an infinite-buffer channel (null image messages are deleted from M'_k). Now, if we observe in the image channel state space the effect of an error event occurring to $\underline{m}_k$, it is either nil or in the form of an error event of the same type occurring to $\underline{m}'_k$. Suppose that the error event is not a loss of the first message in $\underline{m}_k$. The uniform error model requires that E'_k contains error events of this type specified for all positions or pairs of positions of C_k. On the other hand, if the first message in $\underline{m}_k$ is lost, the observable effect in the image channel state space is either nil or that of an identically defined event. Hence, for an infinite-buffer channel also, E'_k must be identical to E_k.

3.8 An Image Protocol System

The specification

$$S'_i,\ M'_{ik},\ T'_i,\ E'_k,\ g'_0 \quad \text{for } i = 1,2,\ldots,I,\ k = 1,2,\ldots,K$$

defines an image protocol system. Here, g'_0, the image of g_0, is the initial state of the image protocol. Due to aggregations, an image protocol system is always smaller than the original protocol system.

Finite state machines example (cont.)

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 $T'_2 = \{ (0',0',+a'_2),\ (0',1',+a'_2),\ (0',1',+a'_3),\ (1',2',\alpha),\ (2',0',-b'_1) \}$. These quantities define the image protocol system which is shown in Figures 6(a) and 6(b). The channels C_1 and C_2 in the image protocol system have the same characterization as in the original protocol system. (We have not yet specified the event sets E_1 and E_2 for these channels.)

Full-duplex data transfer example (cont.)

For P_1 and P_2, we have already obtained their image state spaces to be defined by the state vectors <VS, D_OUT, SOURCE> and <VR, ACK_DUE, SINK> respectively, their image message types to be (DATA',d) and (ACK') respectively, and their image events shown in Tables 2 and 3. These quantities define the image protocol system. The channels C_1 and C_2 in the image protocol system have the same characterization as in the original protocol system.

4. PROPERTIES OF IMAGE PROTOCOLS

For the original protocol system with global state space G and initial state g_0, assertions of its safety properties are predicates on G, while assertions of its liveness properties are predicates on the set of paths in G. The validity of such assertions can be decided by examining the reachability graph R.

Recall that an image protocol is specified just like any real protocol. It has the following set of events

$$E' = \left(\bigcup_{i=1}^{I} T_i'\right) \cup \left(\bigcup_{k=1}^{K} E_k'\right).$$

E' gives rise to a set of transitions τ' in the global state space G'. The transition system (G',τ') governs the logical behavior of the image protocol system. Let R' denote the reachability graph from g_0'. Let R_s' denote the set of global states reachable from g_0'. Informally, an image protocol system is faithful if any logical property, safety or liveness, that can be stated for the image protocol system holds in the image protocol system if and only if it holds in the original protocol system.

When an assertion stated for the image protocol system is considered in the context of the original protocol system, each image protocol quantity x' (entity state, message or event) appearing in the assertion denotes any protocol quantity in the original system whose image is x'.

An image protocol system as defined in Section 3 is in general not faithful. However, we show in Section 4.1 below that image protocols have the following nice property: Any safety property that holds in the image protocol system also holds in the original protocol system. Given a safety assertion to verify for a protocol function, this property of image protocols can be used as a terminating condition for the stepwise refinement process to find an image protocol with adequate resolution. In Section 4.2, we formally define faithfulness for an image protocol system. This definition involves an examination of the reachability graphs R and R'. In Section 4.3, we give sufficient conditions for an image protocol system to be faithful. These conditions can be checked by examining individually the entity and channel events of a protocol system. These conditions are the weakest sufficient conditions that can be stated without any knowledge of R and R'.

4.1 Safety Properties of Image Protocols

The results below apply to any image protocol system obtained according to the definitions in Section 3. Let the original protocol system be characterized by (G,τ), and the image protocol system by (G',τ').

Lemma 1. Given a protocol system and channel states $\underline{m}$ and $\underline{n}$ in $\underline{M}_k$, for any k, if a channel event in E_k can take C_k from $\underline{m}$ to $\underline{n}$, then in an image protocol system, ei-

ther $\underline{m}' = \underline{n}'$ or a channel event in E_k' can take C_k from $\underline{m}'$ to $\underline{n}'$. (A proof of Lemma 1 is given in Appendix II.)

Lemma 2. For any two global states g and h of the original protocol system, if g is extendable to g,h, then in an image protocol system, either g' = h' or g' is extendable to g',h'. (A proof of Lemma 2 is given in Appendix II.)

Theorem 1. The image of every path in (G,τ) is a path in (G',τ').

Coroliary 1. Given any initial global state g_0, the image of every path in R is a path in R'.

Corollary 2. $\{g' : g \in R_s\} \subseteq R_s'$.

The above theorem and corollaries follow readily from Lemma 2 (proofs are given in Appendix II).

Observation. Suppose that a safety assertion has been found to be invariant in an image protocol system. Note that a safety assertion is invariant if it is true on a superset of R_s' in G'. By Corollary 2, the safety assertion must also hold invariantly in the original protocol system.

Finite state machines example (cont.)

Given the initial state (0',0';<>,<>) for this image protocol system, and assuming channels C_1 and C_2 to be error-free (E_1 and E_2 are empty sets), it is easy to verify that the following assertion holds for the image protocol system.

P_1 is in state 5' $\Rightarrow$ 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 in the original protocol P_1 is in any state whose image is 5'. Similarly, message b_1' is in C_2 means that in the original protocol any message whose image is b_1' is in C_2.

We can verify whether this assertion is valid for the original protocol by examining the reachability graph of the original protocol system in Figure 3. However, by Corollary 2, we know that the above assertion is valid for the original protocol system.

The assumption of error-free channels is necessary for the above assertion to hold in both the original and image protocol systems. This assumption is not required in order for the nice property of image protocols stated in Corollary 2 to hold.

Full-duplex data transfer example (cont.)

Assuming C_1 and C_2 to be error-free channels, the following safety assertion has been found to hold for the one-way data transfer image protocol constructed in Section 3.8:

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

2. $VS \geq VR \geq VS\text{-}1$.

3. (DATA',d) in CHANNEL1 $\Rightarrow$ (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 $\Rightarrow$ (D_OUT)
   and (no DATA' message in CHANNEL1)
   and (VS = VR) and (no ACK' message in CHANNEL2)

5. ACK' in CHANNEL2 $\Rightarrow$ (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 $\Rightarrow$ VS = VR

By Corollary 2, the above assertion holds in the original full-duplex data transfer protocol.

Again, the assumption of error-free channels is needed for the above assertion to hold because the protocol is relatively simple and is not designed to handle errors. Without this assumption, no meaningful assertion can be stated. For larger protocol examples involving error-prone channels, the reader is referred to [19, 20].

4.2 Conditions for Faithfulness

Liveness properties of a protocol system are concerned with the future behavior of paths in the reachability graph R. Consider an image protocol system with reachability graph R'. Our definition of faithfulness is formally stated as follows:

Definition (Faithful Image Protocol). For every path w in R and every path u' in R' such that w'=u', the following two conditions hold:

(F1) If w can be extended to a path x in R then u' can be extended to a path v' in R' such that x'=v'.

(F2) If u' can be extended to a path v' in R' then w can be extended to a path x in R such that v'=x'.

The above two conditions guarantee *faithfulness of the image protocol's safety properties* as a result of the following implications:

(F1) $\Rightarrow R'_s \supseteq \{g' : g \in R_s\}$

(F2) $\Rightarrow R'_s \subseteq \{g' : g \in R_s\}$

where R'_s is the set of reachable states of the image protocol system, and $\{g' : g \in R_s\}$ is the set of images of reachable global states of the original protocol system.

To be able to characterize the liveness properties of R given the liveness properties of R', we assume that the original protocol system satisfies the following *fairness assumption*: No event will be indefinitely delayed if it is enabled infinitely often. Given the fairness assumption, conditions (F1) and (F2) guarantee *faithfulness of the image protocol's liveness properties.*

We have proved that all image protocols as constructed in Section 3 satisfy condition (F1). However, an image protocol system satisfies condition (F2) only if it has adequate resolution. To check if (F2) holds requires an examination of R and R'. However, the complexity of R is to be avoided unless it has a special structure that facilitates the checking of (F2). We give in Section 4.3 sufficient conditions for an image protocol to satisfy (F2). These conditions can be checked by examining individually the entity and channel events of the original protocol system, without any knowledge of R or R'.

4.3 Sufficient Conditions Without Knowledge of R or R'

To satisfy condition (F2) above, we require that channels in the original protocol system satisfy a finite lifetime assumption, and each event in the image protocol system is well-formed. These requirements are now explained below.

4.3.1 Assumption about the original protocol system

Finite lifetime assumption: The first message residing in a channel will be deleted in finite time.

The finite lifetime assumption is satisfied by protocol systems in several ways. First, this assumption is satisfied by protocol systems which have been carefully designed to completely avoid "unspecified receptions" [24]. In any reachable state of such a system, if m is the first message in channel C_k whose receiver P_i is in state s, there exists a receive event (s,r,+m) in T_i for some r.

We shall refer to such protocol systems as having a *complete* set of receive events.

If we do not know whether the original protocol system has a complete set of receive events, then unspecified receptions can be avoided by augmenting each entity event set T_i with a receive event set L_i containing receive events of the form (s,s,+m). Two cases are of interest:

Case 1. Exhaustive specification

$L_i = \{(s,s,+m) : s \in S_i, m \in M_k$ for all C_k in the incoming set of $P_i\}$

Case 2. Complementary specification

$L_i = \{(s,s,+m)$: $s \in S_i$, $m \in M_k$ for all C_k in the incoming set of P_i,
such that (s,r,+m) is not in T_i for some r$\}$

Since occurrences of events in L_i do not change the state of P_i, they can be interpreted not as entity receive events but as events that enforce a finite lifetime for messages in a channel. Specifically, a message residing at the head of a channel will be deleted by some "channel controller" within a finite time. Finite lifetimes for messages are highly realistic. First, messages propagating within a physical channel have a small transit time. Second, if a channel is logical and messages are actually travelling within a store-and-forward communication network, they are often subject to mechanisms that enforce bounds on their lifetimes [22].

Whether we assume Case 1 or Case 2 above depends upon the magnitude of message lifetimes compared to the magnitude of a receiving entity's reaction time to handle receive events. If a receiving protocol entity can always execute an enabled receive event for a message prior to the expiration of the message's lifetime, then the complementary specification of L_i is realistic; otherwise, the exhaustive specification should be assumed.

Observation. An exhaustive L_i is logically equivalent to a set of loss events specified for the first position of channel C_k for every channel in the incoming set of P_i. The finite lifetime assumption is automatically satisfied if such loss events are already specified in the channel event sets.

In summary, the original protocol system can satisfy the finite lifetime assumption in one of three ways. First, it has completely specified receive events; in this case, the L_i sets are null. Second, it is augmented by a complementary specification of receive events in $\{L_i\}$. Third, it is augmented by an exhaustive specification of receive events in $\{L_i\}$. It should be clear that in an actual system, the $\{L_i\}$ sets do not have to be explicitly specified and stored.

Lastly, images of events in L_i are defined in the same way as image receive events in Section 3.6.

4.3.2 The well-formed property

The well-formed property of events in an image protocol system is next defined. The image events in T_i' for entity P_i are first considered.

Definition. 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 causing state changes inside a' that will take P_i from a to b.

A send event involving P_i and C_k for some k, can be regarded as an internal event for the above definition if C_k is an infinite-buffer channel and the message being sent has a null image. Under these conditions, the send event (a,b,-m) ϵ T_i can be treated as an internal event (a,b,α) ϵ T_i.

Definition. An *image internal event* of P_i, (s',r',α) ϵ T_i' where s' $\neq$ r', is well-formed if for every a whose image is s', there is some b ϵ S_i that is internally reachable from a and (b,c,α) ϵ T_i for some c ϵ r'.

Again, in the above definition, a send event (b,c,-m) ϵ T_i where m ϵ M_{ik} for some k, can be used instead of (b,c,α) if m has a null image and C_k is an infinite-buffer channel.

Definition. An *image send event* of P_i, (s',r',-n') ϵ T_i', is *well-formed* if for every a whose image is s', there is some b ϵ S_i that is internally reachable from a and (b,c,-y) ϵ T_i for some c ϵ r' and some y whose image is n'.

Definition. An *image receive event* of P_i, (s',r',+n') ϵ T_i', is *well-formed* if for every a whose image is s', the following holds: for *every* y whose image is n' there is some b ϵ S_i that is internally reachable from a, and (b,c,+y) ϵ T_i for some c ϵ r'.

If in any of the above definitions of well-formed events, the length of the internal path is 0 (i.e., b = a), then we say that the image event is *strongly well-formed*.

Note from the construction of T_i' that for every e' ϵ T_i', there is an event e ϵ T_i whose image is e'.

Finite state machines example (cont.)

We will now show that the image events in T_1' are well-formed. (Refer to Figures 4 and 5.) First, consider event (0',5',-a_2') ϵ 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), where a_1 has a null image, state 4 is internally reachable from state 3, and hence from states 0, 1 and 2 also. From 4, event (4,5,-a_3) ϵ T_1 can be executed. Thus, event (0',5',-a_3') ϵ 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. These and the event $(4,3,+a_2) \in T_2$ make 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.

We conclude that all entity events of the image protocol shown in Figure 6 are well-formed.

We next consider image events in E_k' for channel C_k. Note that the occurrence of a channel event affects only the state of the channel. Thus, channel events are analogous to internal events of entities; the well-formed property of channel events is defined similarly.

Definition. An *image channel event* $e' \in E_k'$ is *well-formed* if given any two distinct message sequences $\underline{m}'$ and $\underline{n}'$ in $\underline{M}_k'$ such that e' can take C_k from $\underline{m}'$ to $\underline{n}'$, the following holds: for every message sequence $\underline{p} \in \underline{M}_k$ whose image is $\underline{m}'$, there is a sequence of channel events in E_k that can take C_k from $\underline{p}$, via states each with image $\underline{m}'$, to some message sequence $\underline{q}$ whose image is $\underline{n}'$.

In the above definition, if the sequence of channel events consists of only one event, then e' is said to be *strongly well-formed.*

The channel event sets defined in Section 2.2 have two properties. Firstly, we have shown that for any channel C_k, the image event set E_k' remains the same as E_k. Secondly, we have the following result.

Lemma 3. Each event in E_k' is well-formed. (A proof of Lemma 3 is given in Appendix II.)

(A channel event set E_k, that is more general than those defined in Section 2.2, can

be specified as an arbitrary subset of $\underline{M}_k$ x $\underline{M}_k$. In this case, the above definition for well-formed image channel events must be applied to check each image event in E'_k to see if it is well-formed.)

Lemma 4. Given two distinct global states g' and h' of an image protocol system with well-formed events, if g' is extendable to g',h', then for any global state f in G such that f' = g', f is extendable to a path w such that w' = g',h'. (A proof of Lemma 4 is given in Appendix II.)

Consider a protocol system characterized by (G,τ) and an image protocol system characterized by (G',τ'), the following theorem implies that condition (F2) needed for faithfulness is satisfied if all events of the image protocol in T'_i, L'_i, E'_k for all i and k are well-formed. Note that events in E'_k are well-formed by virtue of Lemma 3. Events in L'_i are well-formed if L_i is given by the exhaustive specification (or is null).

Theorem 2. Given an image protocol system with well-formed events, for any path w in (G,τ) and u' in (G',τ') such that w' = u', if u' is extendable to a path v', then w is extendable to a path x such that x' = v'.

Corollary 3. Given any initial global state $g_0 \in G$ and the corresponding image global state $g'_0 \in G'$, for any path w in R and u' in R' such that w' = u', if u' is extendable to a path v' in R', then w is extendable to a path x in R such that x' = v'.

Corollary 4. $R'_s \subseteq \{g' : g \in R_s\}$.

Theorem 2 follows readily from Lemma 4. Proofs of the preceding theorem and corollaries are given in Appendix II.

Theorem 3. An image protocol system with well-formed events is faithful.

Theorem 3 is an immediate consequence of corollaries 1-4 and the definition of faithfulness. The original protocol system is assumed to satisfy the fairness and finite lifetime assumptions.

Full-duplex data transfer example (cont.)

The image entity events in Tables 2 and 3 can be shown to be well-formed (see [14, 21] for details). Let C_1 and C_2 be error-free channels. Then, in addition to the safety assertions stated earlier for the image protocol system, it can also be easily shown that the variable VR in P_2 grows without bound. Thus, the following liveness property is established:

For any integer $n \geq 0$, the image protocol will eventually satisfy:
SOURCE[i] = SINK[i] for $0 \leq i \leq n$.

From Table 1, we see that the receive events of the original protocol system are com-

plete. Assuming that the scheduling of events in P_1 and P_2 satisfies the fairness assumption, the one-way image protocol constructed is faithful and the above liveness property holds for the original full-duplex protocol as well.

5. STEPWISE REFINEMENT ALGORITHMS

Given an assertion A_0 describing the desired logical behavior of a given protocol system, our objective is to find the smallest image protocol with sufficient resolution to verify whether A_0 holds in the original protocol system.

We start with an image protocol with sufficient resolution to describe A_0. A sequence of image protocols with increasing resolution is then constructed by stepwise refinement. Each successive protocol in the sequence is obtained by a finer partitioning of the original protocol's entity state spaces and/or message sets. Thus, each protocol in the sequence is actually an image protocol of all succeeding protocols constructed. (This property facilitates the verification of safety assertions in Section 5.2.)

Two stepwise refinement algorithms with different termination conditions are presented in the next two subsections. The choice of a termination condition depends upon whether A_0 is a liveness or a safety assertion. Assuming that A_0 is a decidable property of the original protocol system, the following algorithms always terminate. In the worst case, they terminate with the original protocol.

5.1 Termination Based Upon the Well-Formed Property

Algorithm 1. Find an image protocol for verifying a liveness or safety assertion A_0

(1) "Initial step"
partition the entity state spaces so that image entity states in $\{S_i'\}$ have enough resolution for describing A_0;

(2) "Image protocol construction"
obtain the image message sets $\{M_{ik}'\}$ based upon $\{S_i'\}$; obtain the image entity events based upon $\{S_i'\}$ and $\{M_{ik}'\}$;

(3) check each event for well-formedness;

(4) **if** all events are well-formed **then** terminate algorithm
"We have a faithful image protocol for verifying A_0"

(5) **else** "Refinement step"
the point(s) of failure in events that are not well-formed indicate refinements necessary in the partitioning of $\{S_i\}$; update $\{S_i'\}$; go to step (2)

(End of Algorithm 1)

By using the well-formedness of image entity events as the termination condition, Algorithm 1 does not involve any characterization of the reachability graph of each intermediate image protocol constructed.

In step (2), the image message sets $\{M'_{ik}\}$ are derived from the entity state space partitions $\{S'_i\}$ as defined in Section 3.2. We have assumed that this resolution of the image message sets is adequate to describe A_0. As discussed in Section 3.2, given $\{S'_i\}$ the resolution of an image protocol can also be increased by increasing the resolution of the image message sets. This provides an extra degree of freedom which may be utilized in steps (1) and (5) of the above algorithm.

Full-duplex data transfer example (cont.)

We will use this example protocol to illustrate Algorithm 1. We wanted to verify that the original protocol satisfies:

$DP_1 : SINK_2[i] = SOURCE_1[i]$ for $0 \leq i < VR_2$
$DP_2 : VS_1 \geq VR_2 \geq VS_1 - 1$

In step (1), the initial resolution of the entity state spaces is defined by the variables SOURCE and VS in P_1 and SINK and VR in P_2.

Next in step (2), we construct an image protocol. The image states of P_1 and P_2 are given by the values of <VS, SOURCE> and <VR, SINK> respectively. By examining the state changes in the image entity state spaces, we obtain the following image message sets. In M_1, (ACK) has a null image, and (DATA,d) and (DATA&ACK,d) have the same image (DATA',d). All the messages in M_2 have the null image. Thus $M'_1 = \{(DATA',d) : d \in DATASET\}$ and M'_2 is empty. The events of this initial image protocol are shown in Tables 4 and 5. In P_1, SEND_DATA and SEND_DATA&ACK have the image SEND_DATA'. In P_2, REC_DATA and REC_DATA&ACK have the image REC_DATA'. The remaining entity events have a null image.

Having obtained the above initial image protocol, we determine in step (3), whether all its events are well-formed. Consider the event SEND_DATA'. From the events of P_1 in the original protocol, we see that SEND_DATA and SEND_DATA&ACK can occur only when BUSY = D_OUT = False. If BUSY = True, then internal event STOP_BUSY can set BUSY = False. However if D_OUT = True, there is no sequence of null image send or internal events of P_1 that will set D_OUT to False. Hence SEND_DATA' is not well-formed and we go to step (5).

The point of failure is the absence in the image protocol of variable D_OUT of P_1. Hence, we update the entity state space of P_1 to be defined by <VS, D_OUT, SOURCE>, and go back to step (2) and repeat the procedure. Note that there has been no attempt at verification of the image protocol.

In the next iteration we would include ACK_DUE of P_2. In the iteration following that, we would obtain the well-formed image protocol shown in Section 4 (and Tables 2 and 3), and terminate the algorithm.

5.2 Termination Based Upon Safety Properties of Image Protocols

Given a safety assertion A_0, we present another algorithm to find an image protocol with sufficient resolution to verify A_0. Unlike Algorithm 1, A_0 is verified for each image protocol constructed. This verification of A_0 is aided by the fact that a safety property that holds for an intermediate image protocol also holds for all succeeding image protocols. The termination condition for this algorithm is based upon Corollary 1.

In what follows, we use A_0 to denote the set of states in G on which the safety assertion is true. We use B to represent safety properties that have been found to hold for the image protocols already constructed. B can be thought of as a superset of R_s' of the current image protocol.

Algorithm 2. Find an image protocol for verifying a safety assertion A_0

(1) "Initial step"
partition the entity state spaces so that image entity states in $\{S_i'\}$ have enough resolution for describing A_0; let B be G' "nothing is known initially";

(2) "Image protocol construction"
obtain the image message sets $\{M_{ik}'\}$ based upon $\{S_i'\}$; obtain the image entity events based upon $\{S_i'\}$ and $\{M_{ik}'\}$;

(3) verify A_0 given B for this image protocol;

(4) **if** A_0 holds **then** terminate algorithm
"A_0 also holds in the original protocol system"

(5) **else** "We have found a sequence of image events $e_1', e_2', \ldots, e_n'$, referred to as a *test sequence*, that takes the image protocol system from g_0' to some $g' \notin A_0$"
consider those event sequences $e_1, e_2, \ldots, e_m$ where $m \geq n$, whose image equals the test sequence;

(5.1) **if** any of these event sequences can occur in the original protocol system **then** terminate algorithm
"A_0 does not hold in the original protocol system"

(5.2) **else** "Refinement step"
by observing how the test sequence is prevented from occurring in the original protocol system, refinements necessary in the partitioning of $\{S_i\}$ are obtained; update $\{S_i'\}$; if it is observed that the image protocol satisfies some safety property B' then update B to be $B \cap B'$; go to step (2)

(End of Algorithm 2)

In practice, since image protocols are relatively small, many safety properties can often be observed by inspection. Once a safety property has been proved, it is accumulated into B and it will hold for all succeeding image protocols. We note that the smaller B is, the easier it is to verify A_0. Methods to verify A_0 (given B) are described in Section 5.3.

As in the case of Algorithm 1, the resolution of the $\{M'_{ik}\}$ obtained from $\{S'_i\}$ in step (2) can be increased further if needed to describe A_0 in step (1) or in the refinement step.

Full-duplex data transfer example (cont.)

We will use this example protocol to illustrate Algorithm 2. The initial resolution of the entity state spaces in step (1) and the initial image protocol in step (2) are the same as in Algorithm 1 (see Tables 4 and 5).

In step (3), assuming that channels C_1 and C_2 are error-free, we can easily verify that this initial image protocol satisfies DP_1. However, this image protocol does not satisfy DP_2. For instance, from the initial state the image protocol can execute SEND_DATA' twice in succession. The image protocol would then be in a state with VS=2, VR=0 and CHANNEL1=(DATA',SOURCE[1]),(DATA',SOURCE[0]). This state clearly violates DP_2.

Next, in step (5), we have to determine whether the test sequence SEND_DATA', SEND_DATA' can occur in the original protocol. From the initial state we observe that once P_1 executes a SEND-DATA' (either SEND_DATA or SEND_DATA&ACK), D_OUT is set to True, and another SEND_DATA' cannot occur. Further D_OUT is reset to False only in REC_ACK or REC_DATA&ACK. But for either of these to occur, P_2 must execute either SEND_ACK or SEND_DATA&ACK. For that to occur P_2 must have ACK_DUE=True. But ACK_DUE is set to True only when REC_DATA or REC_DATA&ACK occurs at P_2. Neither of these events has a null image. Hence we conclude that the original protocol cannot execute an event sequence whose image equals the test sequence SEND_DATA', SEND_DATA'.

This brings us to step (5.2). The test sequence is prevented from occurring due to the variable D_OUT of P_1. Hence, we update the image entity state spaces of P_1 and P_2 to be <VS,D_OUT,SOURCE> and <VR,SINK> respectively. Also, B can now be set to DP_1, which has already been established. We now go to step (2) and repeat the iterative step.

In the next iteration we would include ACK_DUE of P_2. In the iteration following that, we would have DP_2 holding for the image protocol in Tables 2 and 3. Note that property DP_1 need not be verified again for any of the future image protocols that we obtain, since it has been shown to hold at the initial image protocol considered above.

5.3 Verification Methods

Whether we use Algorithm 1 or Algorithm 2, we need to verify A_0 for an image protocol system. With Algorithm 1, this verification is done for the well-formed image protocol obtained. With Algorithm 2, we need to verify A_0 (given B) for each image protocol constructed in step (3).

One way to verify A_0 for an image protocol system is by generating the reachable set R'_s of states starting from g'_0. This can be done by brute-force (assuming that R'_s is a small finite set) or by symbolic execution [5]. Either A_0 holds at each state in R'_s or a test sequence is obtained.

Another way to verify A_0 is to use the method of weakest preconditions [7]. The objective here is to find an inductively complete set that is a subset of $A_0 \cap B$. We expand on this method here.

For a protocol system with global state space G' and initial state g'_0, A is an *inductively complete set* if the following conditions hold:

(a) $g'_0 \in A$, and

(b) $\forall\, g' \in A, \forall\, e' \in E'$, if e' is enabled at g', then $e'(g') \in A$.

The class of inductively complete sets is closed under intersection, i.e., if A and B are inductively complete sets, so is $A \cap B$. It should be obvious that R'_s is the smallest inductively complete set of the protocol system.

Algorithm 3. Verify A_0 given B for a protocol system

(1) "Initial step"
$C := A_0 \cap B$;
$i := 1$;

(2) "Find weakest preconditions"
$A_i := \{g' \in C \mid \forall\, e' \in E$, if e' is enabled at g' then $e'(g') \in C\}$;

(3) **if** $g'_0 \notin A_i$ **then**
"A_0 does not hold for the protocol system"
obtain a test sequence by examining $A_{i-1}, A_{i-2}, \ldots, A_0$; terminate algorithm

(4) **else if** ($g'_0 \in A_i$ and $A_i = C$) **then** terminate algorithm
"$A_i \subseteq A_0$ is inductively complete and A_0 holds"

(5) **else** $C := A_i$; $i := i + 1$; go to step (2)

(End of Algorithm 3)

If it is observed that the protocol system satisfies some safety property B' then C is

updated to $A_i \cap B'$ in step (5) of the above algorithm. Such observations will speed up the algorithm's search.

6. CONCLUSIONS

The method of projections is intended to reduce the complexity of analyzing multi-function protocols. We show how to construct image protocols for individual protocol functions. Image protocols are specified like any real protocol. Each image protocol is obtained by aggregating entity states, messages and events of the original protocol system. As a result, an image protocol is smaller (never larger) than the original protocol, and can typically be more easily analyzed. Our method effectively breaks up a protocol analysis problem into smaller problems. Unlike the straightforward approach of decomposing protocol entities into functional modules, our method is not handicapped by dependencies that exist between functional modules due to shared variables and shared messages.

Given an assertion A_0, stating the desired logical behavior of a protocol system with respect to a particular protocol function, we have presented two stepwise refinement algorithms for finding an image protocol with sufficient resolution to verify A_0. Algorithm termination assumes that A_0 is a decidable property of the original protocol system. Termination of these algorithms makes use of either one of the following two important properties of image protocols.

First, any safety property that holds for an image protocol system also holds for the original protocol system. Second, given that the original protocol system satisfies a fairness assumption in its event scheduling and its channels satisfy a finite lifetime assumption, an image protocol system with well-formed events is faithful in all its safety and liveness properties to the original protocol system. A faithful image protocol can always be constructed by increasing the resolution of its entity state spaces and message sets. We propose that a multi-function protocol may be considered as well-structured if it possesses small faithful image protocols for its individual functions.

In this paper a protocol system is specified by sets of entity states, messages and events. A set-theoretic notation is used for stating the definitions and properties of image protocols. These results are very general since sets have no internal structure. However, application of these definitions and properties requires operations on sets. Such application is straightforward for protocol entities specified by a finite state machines formalism, but rather cumbersome for protocol entities specified by a programming languages formalism. A more structured specification of protocol entities and messages is presented in [21]; specialization to such a model leads to some efficient rules for image protocol construction. In addition, time variables and time events useful for modeling global timing relations in a protocol system are developed therein. An application of the method of projections to verify the HDLC protocol is presented in [20]. This serves as a rigorous exercise to illustrate the applicability of the method of projections to the analysis of real-life protocols.

APPENDIX I. EXTENSIONS TO THE PROTOCOL SYSTEM MODEL

Many extensions to the protocol system model are possible without affecting the properties of image protocols presented in Section 4. Almost all such extensions are concerned with generalizing the model of channel behavior.

The channel model described in Section 2.2 is already quite general. However, in some real systems, additional features in the channel model are desirable. For example, control messages in the BSC protocol are not protected by error detection [15]. If a BSC control message has bit errors, it may be received and mistaken by the receiving entity as some other control message. Such behavior can be modeled as a new type of error event. In general, channel events can be specified as a subset of $\underline{M}_k \times \underline{M}_k$. In this case, Lemma 3 no longer applies and all image channel events must individually satisfy the well-formed property.

The method of projections allows very general channel behavior because its objective is limited to defining image protocols that preserve some logical properties of a given protocol system. We must keep in mind that if channels have many different types of error events, communication protocols have to be extremely complex in order to possess desirable logical properties. Thus for protocols with limited capabilities, such as the full-duplex data transfer example in this article, we must assume that the channels are well-behaved in order for them to possess desirable properties. The assumption of error-free channels in our examples is due to the simplicity of our example protocols and not required by the method of projections.

Multi-Destination Channels

A channel can be in the incoming channel set of more than one protocol entity. It is a broadcast channel if it is in the incoming set of every protocol entity. When one of these protocol entities executes a receive event, the message being received is not deleted from the channel after the event occurrence. For a multi-destination channel, we assume that the message at the head of the channel has a finite lifetime which is enforced by some "channel controller". (See [20, 21] for discussions on the modeling of real-time and global timing relationships in a system.)

We shall refer to protocol entities that can receive messages from channel C_k as its eligible receivers. The equivalence relation used in the definition of image messages is modified as follows: two messages m and n in M_{ik} are treated as equivalent only if their receptions cause identical state changes in the image state spaces of *all eligible receivers* of C_k. Also, messages in M_{ik} have a null image if (and only if) their receptions do not cause any state change in the image state space of each of the eligible receivers of C_k.

Lastly for a multi-destination channel, any image receive event (s',r',+m') where s'=r' is a null image event since m is not deleted from the channel by receive events.

The proofs in Appendix II are given with this extension incorporated into the protocol system model.

Channel Insertions/Deletions

For the sake of clarity, we have specified that a send event appends a message to the end of a channel's message sequence, and only the first message in the channel is eligible for reception. This requirement is necessary only when null image messages are deleted from the image message set of a channel (i.e. infinite-buffer channel). If null image messages are included in the image message set of C_k, a send event can insert anywhere into the message sequence in C_k, and any message in C_k can be made eligible for reception. The proofs in Appendix II can be trivially extended to accommodate this generalization. Note also that relaxing the assumption that only the first message in the channel is eligible for reception will also allow us to relax the finite lifetime assumption in Section 4.3.1.

Message Priority Classes

Messages in the message sets M_{ik} of C_k may be specified as belonging to different priority classes. Two types of behavior may depend upon a message's priority class. First, error events of a channel C_k can be defined for each position of C_k as well as the priority class of the message residing in that position. Messages in some classes can be made invulnerable to a particular type of error. For example, the following specification can be made: A loss event for the i-th position of C_k is enabled if $\underline{m}_k$ has a message in that position *and* it belongs to a class that is vulnerable to a loss.

Second, the bumping rule in Section 2 can be generalized to the following: The selection of a message to delete may be specified by any rule that depends upon the positions and priority classes of the messages in the channel.

In the construction of an image protocol, messages in M_{ik} can be aggregated to form the same image message only if they have the same priority class and invulnerablility to errors. Note that if M'_k includes null image messages, we have to define different β's, one for each combination of priority class and invulnerability.

The proofs in Appendix II are given with this extension incorporated into the protocol system model.

Initial State of the Protocol System

For notational simplicity, we have assumed that the initial state g_0 of the protocol system is uniquely specified and R is the reachability graph from g_0. Note that all lemmas and theorems in this article do not depend upon an initial state g_0. Thus instead of specifying a single initial state, a set G_0 of possible initial states can be specified. Redefine R to be the reachability graph from G_0 and R' to be the reachability graph from $G'_0=\{g':g\epsilon G_0\}$. Proofs of the four corollaries can be trivially extended to accommodate this generalization.

APPENDIX II. PROOFS

The proofs are given with some of the extensions in Appendix I incorporated into the protocol system model.

Proof of Lemma 1

For message sequences $\underline{m}$ and $\underline{n}$ in $\underline{M}_k$, suppose that a loss event in E_k can take $\underline{m}$ to $\underline{n}$. Then, $\underline{n}$ is obtained by deleting a message y from $\underline{m}$. If C_k is a finite-buffer channel, then the corresponding image message y' is in $\underline{m}'$, and the same loss event in E_k' can delete y' and take the channel to state $\underline{n}'$. If C_k is an infinite-buffer channel, then null images are not included in $\underline{m}'$. If y has a null image, then $\underline{m}' = \underline{n}'$; otherwise, y' is in $\underline{m}'$ and there is a loss event in E_k' that can delete y' and take the channel to state $\underline{n}'$.

Similar arguments can be used for duplication and reordering events.

(End of proof of Lemma 1.)

Proof of Lemma 2

Since g is extendable to g,h , there is an event e in E that takes the protocol system from g to h. The event e can be either a channel event or an entity event. If e is a channel event, then the statement in Lemma 2 is implied by Lemma 1, since a channel event changes the state of the channel only. We next consider the three types of entity events. In the rest of this proof, let $g = (s_1,s_2,...,s_I; \underline{m}_1,\underline{m}_2,...,\underline{m}_K)$ with the image $g' = (s_1',s_2',...,s_I'; \underline{m}_1',\underline{m}_2',...,\underline{m}_K')$.

Case 1. e is an internal event of P_i. Since e is enabled at g, $e = (s_i,r_i,\alpha)$ for some $r_i \in S_i$. h is obtained from g by replacing s_i with r_i. Now, consider the image state g' and the image event $e' = (s_i',r_i',\alpha)$. If $s_i' = r_i'$, then g' = h'. If $s_i' \neq r_i'$, then e' is not null and belongs to T_i'. e' is enabled at g' and takes the image protocol system to a state f', where f' is obtained from g' by replacing s_i' with r_i'. It is easy to see that f' is the same as the image of h.

Case 2. e is a send event involving P_i and C_k. Since e is enabled at g, $e = (s_i,r_i,-n)$ for some $r_i \in S_i$ and some $n \in M_{ik}$. Then, h is obtained from g by replacing s_i with r_i and appending n to the tail of $\underline{m}_k$. (Assume no bumping for now.) Consider the image state g' and the image event $e' = (s_i',r_i',-n')$. If e' is a null image event, we have g' = h'. Otherwise, e' belongs to T_i'. e' is enabled at g' and takes the image protocol system to the image global state f' which is obtained from g' by replacing s_i' with r_i' and appending n' to the tail of $\underline{m}_k'$. (For an infinite-buffer channel, if n' is a null image message, its insertion has no effect on the channel state. The image channel state of C_k in f' remains the same as $\underline{m}_k'$.) It is easy to see that f' is the same as h', the image of h.

Now suppose that bumping occurred. Let the message bumped be y; y is either the message sent or a message in $\underline{m}_k$. Consider the image message y' corresponding to y. Note that each message in $\underline{m}_k$ and its image in $\underline{m}_k'$ must have the same priority class

and occupy the same position in C_k. Hence y' can be bumped in the image protocol system. Again, it is easy to see that f' is the same as h'.

Case 3. e is a receive event involving P_i and C_k. Since e is enabled at g, the following two conditions hold. First, $\underline{m}_k$ is not empty; let $n \in M_k$ be the first message in $\underline{m}_k$. Second, $e = (s_i, r_i, +n)$ for some $r_i \in S_i$. Then, h is obtained from g by replacing s_i with r_i and removing n from $\underline{m}_k$. (Let C_k be a single-destination channel.) Consider the image state g' and the image event $e' = (s_i', r_i', +n')$. If e' is a null image event, then $g' = h'$. Otherwise, e' belongs to T_i'. (If C_k is an infinite-buffer channel, we also know that n' is not a null image message.) e' is enabled at g' and takes the image protocol system to the image global state f', which is obtained from g' by replacing s_i' with r_i' and removing n' from the head of $\underline{m}_k'$. It is easy to see that f' is the same as h', the image of h. Lastly, if C_k is a multi-destination channel, the occurrence of e does not remove n from $\underline{m}_k$ and the occurrence of e' does not remove n' from $\underline{m}_k'$. f' is still equal to h'.

(End of proof of Lemma 2.)

Proof of Theorem 1

The proof is by induction on path length. A unit-length path in (G,τ) corresponds to a global state g in G. The image of g is g', which is a unit length path in (G',τ').

For some positive integer n, assume that the image of every path in (G,τ) of length n is a path in (G',τ'). We will show that the image of any path of length n+1 is a path in (G',τ'). Let $w = u,f_n$ be a path of length n+1 in (G,τ), where $u = f_0,f_1,\ldots,f_{n-1}$. Obviously, u is a path of length n in (G,τ). From the above hypothesis, u' is a path in (G',τ') and the last state in u' is f_{n-1}'. If $f_n' = f_{n-1}'$, then $w' = u'$ and the image of w is a path in (G',τ'). If $f_n' \neq f_{n-1}'$, then by Lemma 2, f_{n-1}' is extendable to f_{n-1}',f_n'. Hence u' is extendable to u',f_n' which is w', and the image of w is a path in (G',τ').

(End of proof of Theorem 1.)

Proof of Corollary 1

The first element of any path w in R is g_0. Let the image of w be w'. From Theorem 1, w' is a path in (G',τ'). The first element of w' is g_0'. Hence, w' is a path in R'.

(End of proof of Corollary 1.)

Proof of Corollary 2

For any reachable state $g \in R_s$, there is a path w leading from g_0 to g. From Corollary 1, w' is a path in R'. The last element of w' is g'. Hence, g' is in R_s'.

(End of proof of Corollary 2.)

Proof of Lemma 3

Consider message sequences $\underline{m}'$ and $\underline{n}'$ in $\underline{M}_k'$ and an error event in E_k' that can take $\underline{m}'$ to $\underline{n}'$. Suppose that null image messages are included in M_k', so that each element y' in $\underline{m}'$ has a corresponding element y at the same position in any $\underline{p}$ whose image is $\underline{m}'$. Also all messages with the same image have by definition identical priority and invul-

nerability to errors. Thus, the same error event in E_k can take any $\underline{p}$ whose image is $\underline{m}'$ to some $\underline{q}$ whose image is $\underline{n}'$. Each event in E_k' is strongly well-formed.

Let C_k be an infinite-buffer channel and E_k specified by the uniform error model. Suppose that a loss event is specified for the first position of E_k' (which is equal to E_k). Consider a message sequence $\underline{m}'$ with y' as its first element. Any message sequence $\underline{p}$ whose image is $\underline{m}'$ must contain a corresponding element y that is preceded by null image messages (if any). The same loss event in E_k can delete the null image messages one by one and then delete y in $\underline{p}$. Hence, this loss event is well-formed (but not strongly well-formed). Other error events (loss, duplication or reordering) in E_k, when present, are specified for all positions of C_k for each type of events. Consider message sequences $\underline{m}'$ and $\underline{n}'$ in $\underline{M}_k'$ and an error event in E_k that can take $\underline{m}'$ to $\underline{n}'$. Each element y' in $\underline{m}'$ has a corresponding element y in any $\underline{p}$ whose image is $\underline{m}'$. The position of y in $\underline{p}$ may be different from that of y' in $\underline{m}'$. Since the uniform error model specifies this type of error event for all positions of C_k, an error event of the same type exists that can take $\underline{p}$ to some $\underline{q}$ whose image is $\underline{n}'$. Such error events are strongly well-formed.

With duplication events, remember that all messages with the same image must have the same priority class when applying the bumping rule.

(End of proof of Lemma 3.)

Proof of Lemma 4

Since g' is extendable to g',h' , there is an event $e' \in E'$ that takes the image protocol system from g' to h'. Let $g' = (s_1', s_2', \ldots, s_I'; \underline{m}_1', \underline{m}_2', \ldots, \underline{m}_K')$. e' can be either a channel event or an entity event. If e' is a channel event in E_k, then h' is the same as g' except for the state of channel C_k. Let $\underline{n}_k'$ be the state of channel C_k in h'. From Lemma 3, we note that the same channel event will take any $\underline{p}$ whose image is $\underline{m}_k'$ to some $\underline{q}$ whose image is $\underline{n}_k'$. Hence, it will take any f_1 whose image is g' to some f_2 whose image is h'. We next consider e' to be an image entity event of P_i'. Three cases are considered for the three types of entity events. Since the image protocol system has well-formed events, e' is well-formed in each case. Let $f_1 = (a_1, a_2, \ldots, a_I; \underline{p}_1, \underline{p}_2, \ldots, \underline{p}_K)$ be any global state whose image is g'.

Case 1. e' is an image internal event of P_i'. Since e' is enabled at g', T_i' contains $e' = (s_i', r_i', \alpha)$ for some $r_i' \in S_i'$. h' is the same as g' except for the state of P_i'. Since e' is well-formed, we know that there exists some $b \in S_i$ that is internally reachable from a_i, and event (b,c,α) exists in T_i such that $b' = s_i'$ and $c' = r_i'$. The sequence of events that makes b internally reachable from a_i will extend f_i to some global state f_n; the image of each element in the extension is g'. In the global state f_n, the state of P_i is b and (b,c,α) in T_i will take f_n to some f_{n+1} whose image is h'.

Case 2. e' is an image send event involving P_i' and C_k. Since e' is enabled at g', T_i' contains $e' = (s_i', r_i', -n')$ for some $r_i' \in S_i'$ and $n' \in M_{ik}'$. h' is obtained from g' by replacing

s'_i with r'_i and appending n' to the tail of $\underline{m}'_k$. (Assume no bumping for now.) Since e' is well-formed, we know that there exists some $b \in S_i$ that is internally reachable from a_i, and event (b,c,-y) exists in T_i for some c and some y such that $b' = s'_i$, $c' = r'_i$ and $y' = n'$. The sequence of events that makes b internally reachable from a_i will extend f_1 to a global state f_n; the image of each element in the extension is g'. In the global state f_n, the state of P_i is b and (b,c,-y) in T_i will take f_n to some f_{n+1} whose image is h'.

Now assume that bumping occurred. Let z' be the message bumped; z' is either the message n' that was sent or a message in $\underline{m}'_k$. Consider the message z corresponding to z'. Since the bumping rule depends only on positions in C_k and message priority classes, both of which are preserved when aggregating messages, z can be bumped in the original protocol system. Again, f_{n+1} has the image h'.

Case 3. e' is an image receive event involving P'_i and C_k. Since e' is enabled at g', T'_i contains $e' = (s'_i, r'_i, +n')$ for some $r'_i \in S'_i$ and $n' \in M'_k$. h' is obtained from g' by replacing s'_i with r'_i and removing n' from the head of $\underline{m}'_k$. (If C_k is a multi-destination channel, n' is not removed from $\underline{m}'_k$.) Consider the state of channel C_k in any global state f_1 whose image is g'. If null image messages are included in $\underline{M}'_k$, then the first element in the channel state $\underline{p}_k$ has the image n'. If null image messages are deleted from M'_k (i.e., C_k is an infinite-buffer channel), then the message y in $\underline{p}_k$ corresponding to the image message n' may be preceded by messages with a null image. By the finite life-time assumption of the original protocol system, these messages having a null image will be deleted in finite time (each message deletion is an event that affects only the state of the channel). Suppose now that the first element in channel C_k is y with the image n'. Since e' is well-formed, we know that there exists some b in S_i that is internally reachable from a_i, and event (b,c,+y) exists in T_i for some c such that $b' = s'_i$ and $c' = r'_i$. The sequence of events that makes b internally reachable (as well as those events that delete null image messages in front of y in an infinite-buffer channel) will extend f_1 to some global state f_n; the image of each element in the extension is g'. In the global state f_n, the state of P_i is b and (b,c,+y) in T_i will take f_n to some f_{n+1} whose image is h'.

(End of proof of Lemma 4.)

Proof of Theorem 2

Let the last element in u' be $g' \in G'$ and the last element in w be $f \in G$. Since $w' = u'$, we must have $f' = g'$. To prove Theorem 2, it suffices to show the following: if g' is extendable to a path y' in (G',τ'), then f is extendable to a path z in (G,τ) such that $z' = y'$. This can be easily shown using Lemma 4 and applying induction.

(End of proof of Theorem 2.)

Proof of Corollary 3

The corollary follows immediately from Theorem 2.

(End of proof of Corollary 3.)

Proof of Corollary 4

Any g' in R'_s lies on a path u' from g'_0 to g'. Since g_0 is in R, from Corollary 3, g_0 can be extended to a path w such that w' = u'. Let the last element in w be f. We must have f ϵ R_s and f' = g'.

(End of proof of Corollary 4.)

REFERENCES

[1] G. V. Bochmann and R. J. Chung, "A Formalized Specification of HDLC Classes of Procedures," in Conf. Rec. Nat. Telecommun. Conf., Los Angeles, December 1977, pp. 03a..2-1-2-11.

[2] G. V. Bochmann, "Finite State Description of Communication Protocols," Computer Networks, Vol. 2, pp. 361-372, Oct. 1978.

[3] G. V. Bochmann and C. A. Sunshine, "Formal Methods in Communication Protocol Design," IEEE Trans. on Commun., Vol. COM-28, No. 4, pp. 624-631, April, 1980.

[4] G. V. Bochmann and P. Merlin, "On the Construction of Communication Protocols," in Proc. 5th ICCC, October 1980, Atlanta.

[5] D. Brand and W. H. Joyner, "Verification of Protocols using Symbolic Execution," Computer Networks, Vol. 2, pp. 351-360, Sept. 1978.

[6] D. Brand and W. H. Joyner, "Verification of HDLC," IEEE Trans. on Commun., Vol. COM-30, No. 5, pp. 1136-1142, May 1982.

[7] E. W. Dijkstra, A Discipline of Programming, Englewood Cliffs, NJ: Prentice-Hal, 1976, ch. 3, pp. 15-23.

[8] B. L. DiVito, "Verification of Communication Protocols and Abstract Process Models," PhD thesis, Dept. of Computer Sciences, University of Texas at Austin, August 1982.

[9] B. T. Hailpern and S. S. Owicki, "Verifying Network Protocols using Temporal Logic," Technical Report 192, Computer Systems Laboratories, Stanford Univ., June 1980.

[10] "Data Communications--HDLC Procedures--Elements of Procedures," Ref. No. ISO 4335, 1979, International Standards Organization.

[11] J. Kurose, "The Specification and Verification of a Connection Establishment Protocol using Temporal Logic," in Proc. 2nd Int. Workshop on Protocol Specification, Testing and Verification, May 1982, pp. 43-62, Idyllwild, CA.

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

[13] S. S. Lam and A. U. Shankar, "Verification of Communication Protocols via Protocol Projections," in Proc. INFOCOM'82, April 1982, pp. 229-240, Las Vegas.

[14] S. S. Lam and A. U. Shankar, "An Illustration of Protocol Projections," Proc. 2nd Int. Workshop on Protocol Specification, Testing and Verification, May 1982, pp. 343-360, Idyllwild, CA.

[15] S. S. Lam, "Data Link Control Procedures," ch. 3 in Computer Communications, Vol. I: Principles, W. Chou (ed.), Englewood Cliffs, NJ: Prentice-Hall, 1983.

[16] R. Razouk, "Modeling X.25 Using the Graph Model of Behavior," in Proc. 2nd Int. Workshop on Protocol Specification, Testing and Verification, May 1982, pp. 197-214, Idyllwild, CA.

[17] R. L. Schwartz and P. M. Melliar-Smith, "From State Machines to Temporal Logic: Specification Methods for Protocol Standards," IEEE Trans. on Commun. Vol. COM-30, No. 12, pp. 2486-2496, December 1982.

[18] A. U. Shankar and S. S. Lam, "On Time-Dependent Communication Protocols and their Projections," in Proc. 2nd Int. Workshop on Protocol Specification, Testing and Verification, May 1982, pp. 215-236, Idyllwild, CA.

[19] A. U. Shankar, "Analysis of Communication Protocols via Protocol Projections," PhD thesis, Dept. of Electrical Engineering, Univ. of Texas at Austin, December 1982.

[20] A. U. Shankar and S. S. Lam, "An HDLC Protocol Specification and its Verification using Image Protocols," Technical Report 212, Dept. of Computer Sciences, Univ. of Texas at Austin, September 1982 (first version), May 1983 (revised version), to appear in ACM Trans. on Computer Systems, Nov. 1983.

[21] A. U. Shankar and S. S. Lam, "Application of Projections to a Structured Model of Communication Protocols," Technical Report 214, Dept. of Computer Sciences, University of Texas at Austin, 1983 (in preparation).

[22] L. Sloan, "Mechanisms that Enforce Bounds on Packet Lifetimes," presented at the ACM SIGCOMM '83 Symposium, Austin, TX, March 1983, (to appear in ACM Trans. on Computer Systems).

[23] N. V. Stenning, "A Data Transfer Protocol," Computer Networks, Vol. 1 (2), pp. 99-110, September 1976.

[24] P. Zafiropulo, et al, "Towards Analyzing and Synthesizing Protocols," IEEE Trans. on Commun., Vol. COM-28, No. 4, pp. 651-660, April 1980.

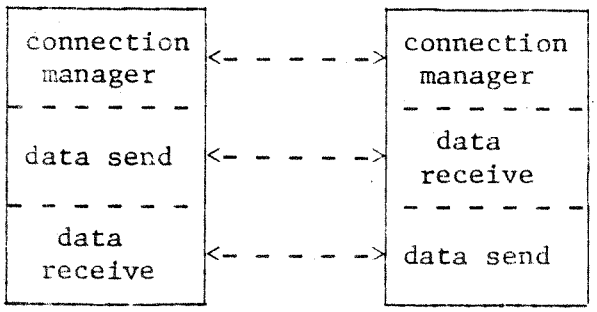

Figure 1. The three functions of an HDLC protocol.

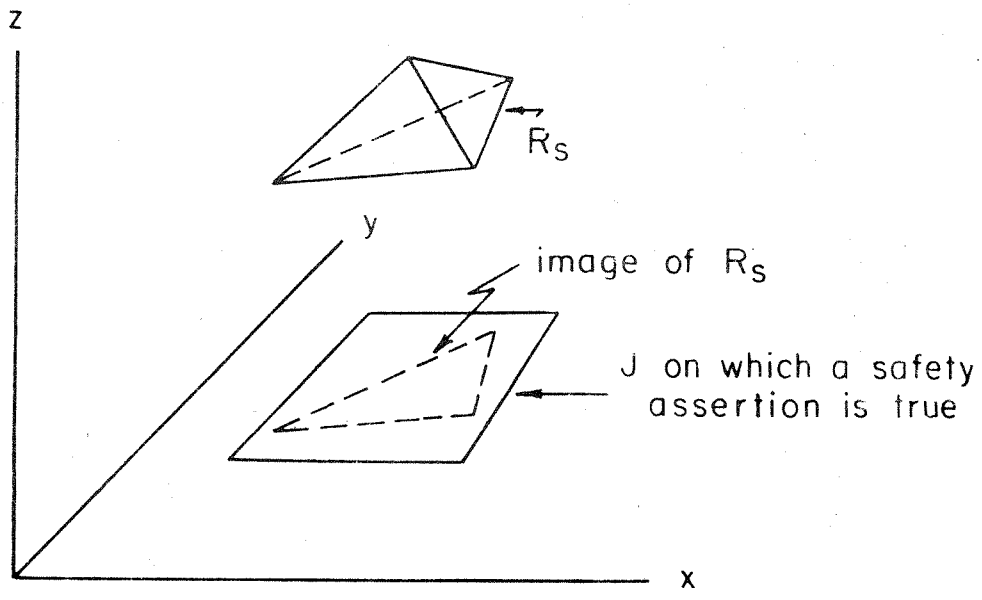

Figure 2. An illustration of the projection idea.

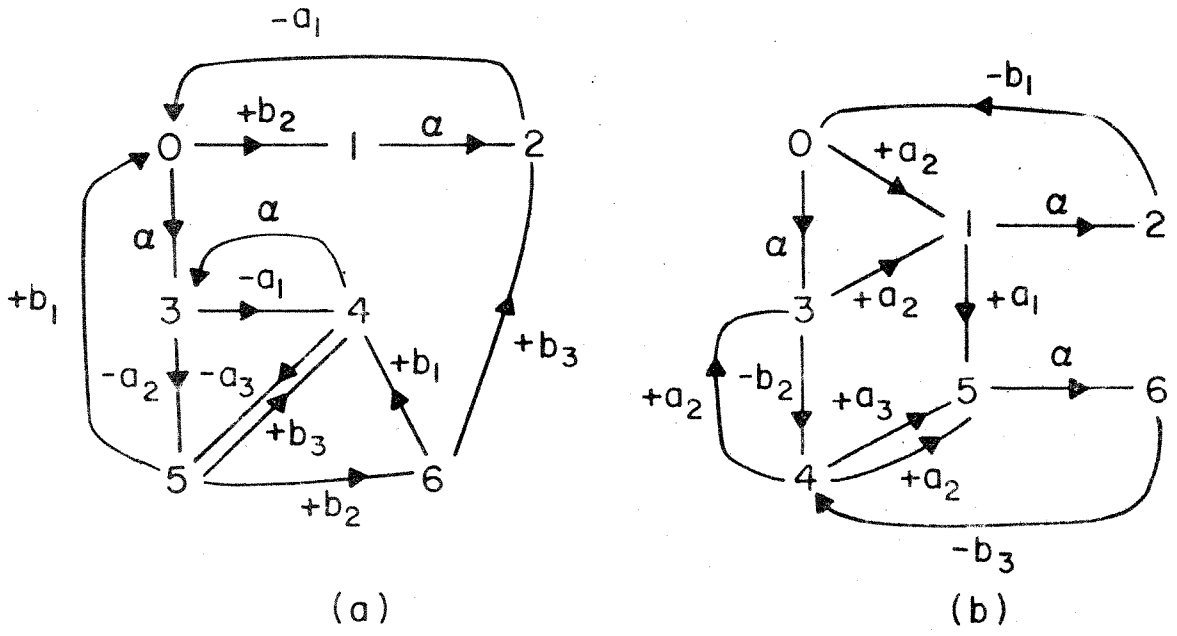

Figure 3. Two communicating finite state machines.

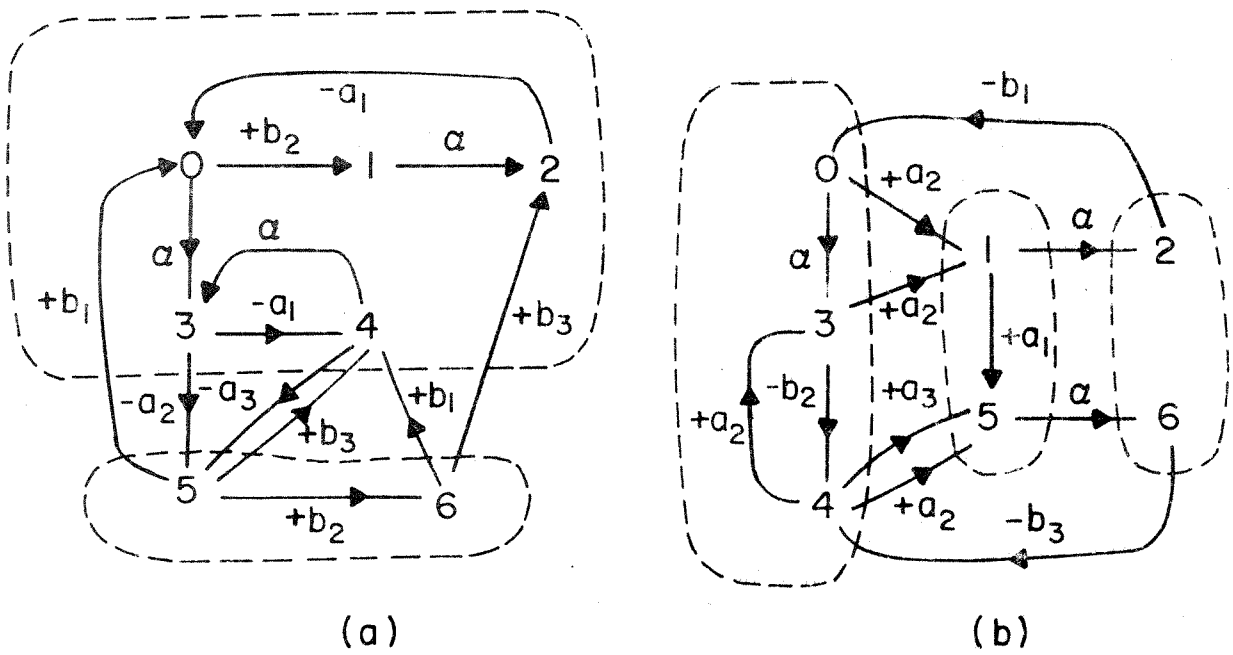

Figure 4. A partitioning of the entity state spaces.

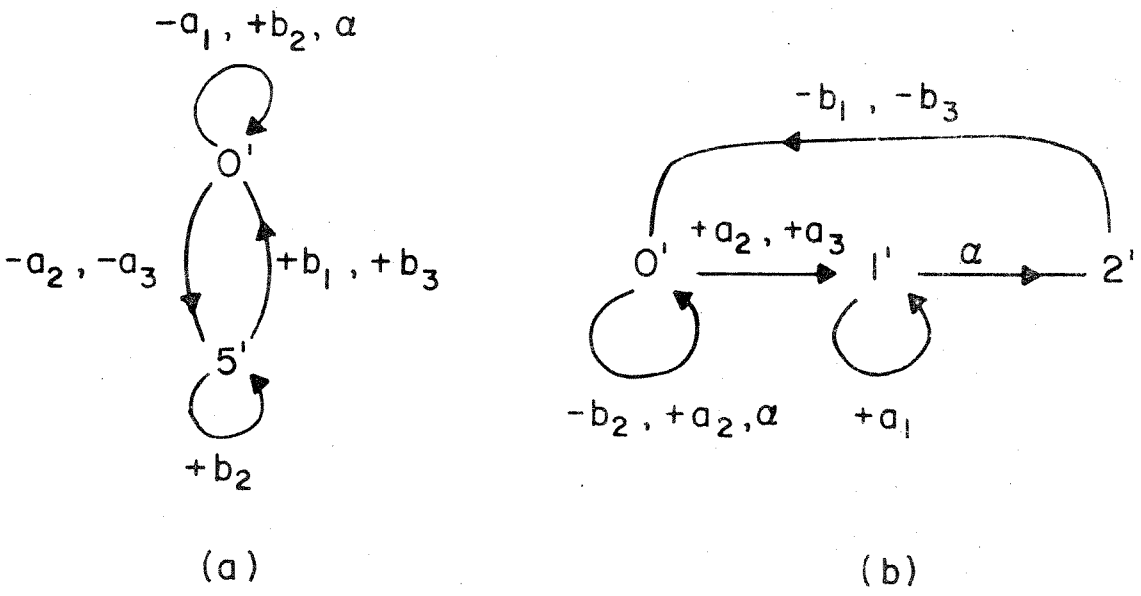

Figure 5. Transitions in image state spaces.

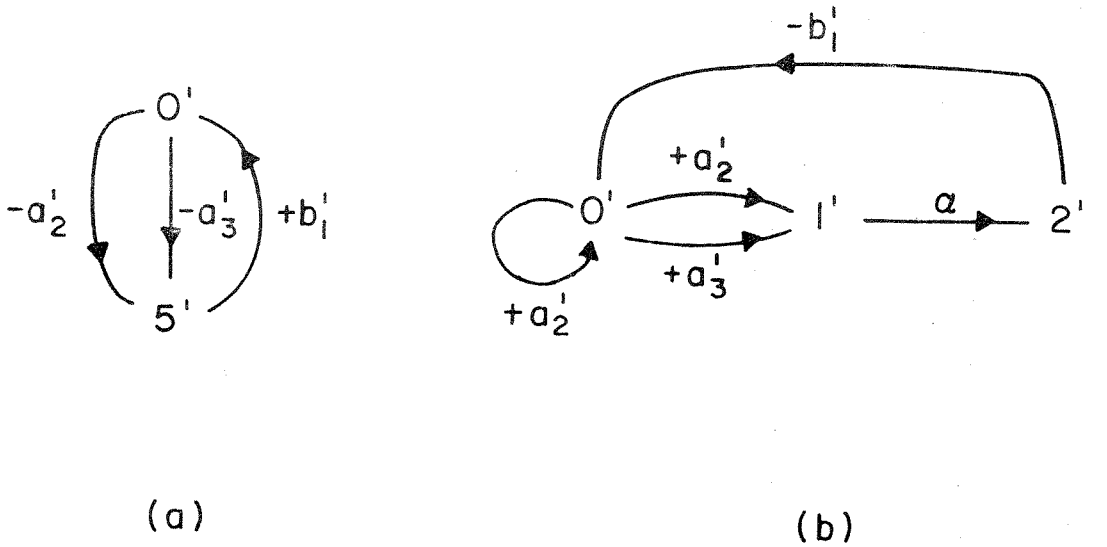

Figure 6. An image protocol of the communicting finite state machines.

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&ACK	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 D_OUT	SDATA := SOURCE[VS]; put(CHANNEL1, (DATA', SDATA)); VS := VS + 1; D_OUT := true
2. 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. SEND_ACK'	ACK_DUE	put(CHANNEL1, (ACK')); ACK_DUE := false

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

Event Name	Enabling Condition	Action
1. SEND_DATA'		SDATA := SOURCE[VS]; put(CHANNEL1, (DATA', SDATA)); VS := VS + 1

TABLE 4. Events of P_1 in the initial image protocol.

Event Name	Enabling Condition	Action
1. REC_DATA'	first(CHANNEL1) = DATA'	get(CHANNEL1, (DATA', RDATA)); SINK[VR] := RDATA; VR := VR + 1

TABLE 5. Events of P_2 in the initial image protocol.