APPLICATIONS OF PROJECTIONS TO A STRUCTURED MODEL OF COMMUNICATION PROTOCOLS[1]

A. Udaya Shankar and Simon S. Lam

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

TR-214 October 1982

[1]This work was supported by National Science Foundation Grant No. ECS78-01803.

Table of Contents

Abstract

The method of projection is an approach to reduce the complexity of analyzing nontrivial communication protocols with several distinguishable functions. In this method, a faithful image protocol is constructed for each function of interest in the original protocol. The method has been described earlier using an abstract model in which protocol entities are specified by states and state transitions, and the construction of faithful image protocols involves operations on sets of states and transitions. These set operations are inconvenient when the protocol being projected is large. In this paper we describe a model in which the entities are specified by programs, and the construction of faithful image protocols can be done easily without the use of set operations (in most cases). Thus, the model is useful for complex multi-function protocols and has been applied to specify and verify a version of the high-level data link control (HDLC) protocol. The model includes several realistic protocol features such as multi-field messages and timers.

1. INTRODUCTION

The method of projections, described in [LAM 82a, LAM 82b], is an approach to reduce the complexity of analyzing non-trivial communication protocols with several distinguishable functions. The application of the method to the protocol model used in [LAM 82b] involves set operations that can become difficult when large protocols are considered. In this paper, we describe a protocol model in which the method of projections can be applied without the need for set operations (in most cases).

A protocol system consists of a network of protocol entities and channels. The network configuration is static but may be arbitrary. At any time, the global state of the system is described by a joint description of the states of the entities and channels. Thus, the set G of global states of the protocol system is the cartesian product of the entity and channel state spaces. The states of entities and channels (and hence the global state) can change due to the occurrence of the following events: entities sending messages into channels, entities receiving messages from channels, errors occurring to messages in transit within channels, and entities receiving signals from their timers and their external users.

An event can occur in a particular global state only if it is enabled in that state. We make the usual assumption that the occurrence of each event causes an indivisible transition from one global state to another. Furthermore, if there is any possibility of events occurring simultaneously, such a simultaneous occurrence can be represented as a sequence of the same event occurrences in some arbitrary order. This last assumption is reasonable because events in communication protocols can usually be defined in such a way that their occurrences are instantaneous.

The set τ of global transitions is determined by the set of entity and channel events (their enabling conditions and actions). τ is a binary relation on G and the pair (G,τ) defines a directed graph whose nodes are elements of G and whose arcs are elements of τ. 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 and is denoted by R. R contains all available information on the logical correctness properties of the protocol system. Let R_s denote the set of reachable states in G. R_s, which may be obtained from R, determines the safety (partial correctness) properties of the protocol system. Liveness properties, however, require knowledge of the set of paths in R.

1.1 The Projection Idea

We observe that 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 transfer in two directions [ISO 79].

To analyze 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. Bochmann and Chung [BOCH 77] used such a decomposition approach to specify a version of the HDLC protocol. 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 packets).

Most communication protocols that have been rigorously analyzed and presented in the literature are protocols with a single function: either a connection management function or a one-way data transfer function [STEN 76, HAIL 80, BRAN 82]. The following question arises: Are the safety and liveness properties that are proved for a single function protocol still valid when it is implemented as part of a multi-function protocol with the two types of dependencies mentioned above? 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 protocol analysis. The projection approach avoids the characterization of R. Instead, from the given protocol a faithful image protocol is constructed 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 treating groups of states, messages and events in the original protocol as equivalent and aggregating them [LAM 82b]. As a result, image protocols are smaller than the original protocol. For example, an image protocol of HDLC for the function of one-way data transfer would be of the same order of complexity as other one-way data transfer protocols that have been successfully analyzed [STEN 76, DIVI 82]. Informally, an image protocol is faithful if the following is true: Any logical property, safety or liveness, concerning the projected function holds in the image protocol system if and only if it also holds in the original protocol system. The precise meaning of faithfulness is defined in [LAM 82b] and is repeated in Section 3.2. It has been shown in [LAM 82b] that if an image protocol is constructed with sufficient resolution so that its entities satisfy a well-formed property then the image protocol is faithful (the well-formed property involves an examination of the protocol entities individually and is the weakest condition that one can have without any knowledge of R).

1.2 A Structured Protocol Model

The protocol entities of the model used in [LAM 82b] have very little structure. Each entity is specified by a set of states, a set of messages, and a set of entity events. Each entity event is specified as a state transition that either sends a message, receives a message, or involves no message at all. This lack of structure in the model makes the results obtained in [LAM 82b] applicable to a very general class of protocols. However, this same lack of structure necessitates the use of set operations when constructing faithful image protocols. When large protocols are considered, these set operations can become difficult.

To overcome this difficulty, we must add more structure to the above protocol model, and use this structure to restrict the types of aggregations allowed, thereby facilitating the construction of well-formed image protocols. At the same time, the resulting structured model should be general enough to model a wide class of realistic protocols.

In this paper, we describe such a structured model for a protocol system. Each component (entity or channel) of the protocol system is modeled with a set of variables local to the component, and a list of events that manipulate these variables and pass messages to adjacent components. The events of the entities are specified in a programming language notation. Of the variables used in modeling an entity, some may correspond to variables that are implemented in the protocol system being modeled, while others may be auxiliary variables used in the specification and verification of the protocol system.

The messages in our new protocol model can have multiple fields. We include this feature since many communication protocols use multi-field messages where each field carries information relevant to particular functions of the protocol. The messages are specified in terms of message types. Each message type is similar to a record definition in programming languages. It specifies a list of fields and the values that each field can have, thereby specifying a set of multi-field messages.

Finally, our model can be applied to time-dependent protocols. A time-dependent protocol is a protocol where measures of time (obtained by timers and clocks in entities, and age fields in packets) play a crucial role in the correct logical behavior of the protocol [SHAN 82a, FLET 78, SLOA 79]. Such protocols can be modeled using time variables to measure elapsed times, and time events to age time variables [SHAN 82a]. We allow time variables and time events in the components of our protocol model. Some of these time variables may correspond to timers and clocks that are implemented in the protocol system, while others are auxiliary variables used in stating assertions about the time-dependent behavior of the protocol system.

for this protocol model, we will consider image protocols that are formed by retaining a subset of the variables at each entity, and a subset of the fields in each message type; i.e., all entity states

(messages) that have equal values for retained variables (fields) are treated as equivalent and have the same image in the image protocol. There are three reasons for choosing these restrictions.

First, assertions about the image protocol can be immediately related to the original protocol, since each variable (message field) in the image protocol corresponds exactly to a variable (message field) in the original protocol.

Second, it greatly simplifies the construction of faithful image protocols in most cases. Consider Figure 3. In the old model, the construction of faithful image protocols involved set operations (right side in Figure 3). In the new model, if the program description of the entity events display certain nice structures (described in Section 4), then well-formed image protocols can be found almost by inspection (corresponding to the path involving simple operations in Figure 3). For those entity events whose program descriptions do not display any nice structure, it is necessary to use the set operations of the previous model (corresponding to the path involving transformations and set operations in Figure 3). We have found, however, that in many practical situations, the program descriptions display the nice structures required for easily obtaining well-formed image protocols. For example, well-formed image protocols for a version of HDLC have been constructed without much difficulty [SHAN 82b, SHAN 82c]. In general, the more well-structured the program description, the easier it will be to obtain image protocols by inspection.

Third, the restriction provides an iterative procedure that identifies the variables needed to be retained in order for the image protocol to be well-formed. The procedure is as follows. First, identify the entity variables needed to state the desired logical behavior of the function being projected; typically, these are the variables found in assertions that need to be verified. Obtain the image protocol for this choice of retained variables, and check if it is well-formed. The check for well-formedness, if it fails, will identify variables from the original protocol that caused the image protocol not to be well-formed. Retain these variables and obtain a new image protocol. Keep repeating until a well-formed image protocol is found. The process will terminate in the worst case with the image protocol being the original protocol itself. It has been our experience that for protocols with several distinguishable functions, well-formed image protocols can be constructed for individual functions, that are substantially smaller than the original protocol.

The objective of the method of projections is illustrated in Figure 4. Suppose that we are given a protocol system and one or more assertions that specify the correct behavior of some protocol function. Suppose also that a verifier is available for checking the validity of assertions for a given protocol system. Instead of feeding the assertions and the original protocol system into the verifier, our objective is to first construct a faithful image protocol (which should be much simpler than the original protocol). The image

protocol system and assertions are then fed into the verifier for evaluation. If we are interested in several functions of the protocol, a different image protocol is generated for each function. Since the construction of faithful image protocols involves an examination of protocol entities individually, the complexity of characterizing the reachability graph R for the entire system is avoided.

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 [BOCH 80]. 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 described in [LAM 81, LAM 82a, LAM 82b].

1.3 Summary of the Rest of the paper

In Section 2, we describe our protocol system model without time variables and time events. Consider Figure 3. The right side of Figure 3, involving set operations, describes the steps in the construction of faithful image protocols using the old model. In the new model, we would like to construct faithful image protocols using only simple operations, as shown in the left side of Figure 3. In Section 3, we describe the general procedure for constructing faithful image protocol systems from a given protocol system. The new model is described as a more structured version of the old model. The construction procedure and results obtained for the old model in [LAM 82b] and stated in terms of the new model. This construction procedure corresponds to the path involving transformations and set operation in Figure 3.

In Section 4, we describe entity events whose program descriptions display certain nice structures so that faithful image protocols can be obtained without the need for set operations. The steps in this construction procedure correspond to the left side of Figure 3.

In Section 5, we add time variables and time events to our protocol model, and explain the construction of image protocols for time-dependent protocol systems.

2. THE PROTOCOL SYSTEM MODEL

Throughout the rest of this paper, we shall consider the protocol system shown in Figure 1. P_1 and P_2 are protocol entities that communicate with each other. P_1 sends messages to P_2 through channel C_1, and P_2 sends messages to P_1 through channel C_2. (This configuration, rather than an arbitrary network of entities and channels, is used strictly for notational simplicity. The results on paper can be immediately generalized to networks of arbitrary interconnection with point-to-point and broadcast channels [LAM

82b].) The entities and channels will be modeled as event-driven processes that interact by passing messages to adjacent components.

An event-driven process consists of events. An event can occur only if variables of the protocol system satisfy certain conditions, denoted as the enabling condition of the event. When an enabled event occurs, variables of the protocol system are affected. Whenever an event-driven process has enabled events, any one of them can occur (we assume fairness in the choice of the event).

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 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. The rule used to select the message to be bumped is referred to as the bumping rule. It must satisfy certain restrictions which are described in Section 2.1.

2.1 Messages of the Protocol Model

The messages of the protocol system have multiple fields, and are specified in terms of message types. A message type is a tuple of the form $(q;f_1:F_1,f_2:F_2,...,f_n:F_n)$ where $n \geq 0$. The first component, q, is the name of the message type and is a constant. The other components (if any) specify a list of parameters and their ranges in a Pascal-type notation. For $i \leq i \leq n$, F_i is a set of values and f_i is a parameter that can range over the values in F_i. f_i is referred to as a field and F_i as a field space. The set of messages specified by message type q is $\{(q,a_1,...,a_n) : a_i \in F_i \text{ for } 1 \leq i \leq n\}$. A message $(q,a_1,...,a_n)$ is referred to as a message of type q. For notational simplicity, we resort to vector notation: $(q;f_1:F_1,...,f_n:F_n)$ as $(q;\underline{f}:\underline{F})$, where $\underline{f} = (f_1,...,f_n)$ and $\underline{F} = F_1 \text{ x...x } F_n$; $(q,a_1,...,a_n)$ as $(q,\underline{a})$. Also, message type $(q;\underline{f}:\underline{F})$ is often referred to by message type q.

Let Q_i denote the (finite) set of message types sent by P_i for i=1 and 2. Let M_i denote the union of message sets specified by the message types in Q_i. M_i is the set of messages sent by P_i.

Bumping rule restriction: Recall that when a message is sent into a finite-buffer loss channel that is already full, a message is deleted according to a bumping rule. The bumping rule can depend upon the state of the channel, but only to the extent of knowing the type of each message.

2.2 Variables of the Entities and Channels

Each component of the protocol model has a set of variables that define its state space.

For i=1 and 2, let P_i have the variables $v_{i,1}, v_{i,2}, \ldots v_{i,k_i}$, and let $S_{i,j}$ denote the domain (set of values) of $v_{i,j}$, for each $v_{i,j}$. The state vector of P_i is defined to be $\underline{v}_i = (v_{i,1}, \ldots, v_{i,k_i})$. The domain of $\underline{v}_i$ is $S_i = S_{i,1} \times \ldots \times S_{i,k_i}$. At any time, the value of $\underline{v}_i$ represents the state of P_i. S_i is the state space of entity P_i. (Some of the variables in $\underline{v}_i$ can be auxiliary variables useful in specification/verification of the protocol system. The $v_{i,j}$'s can be structured variables themselves.)

For channel C_i, define $Channel_i$ as the variable that represents, at any time, the sequence of messages in C_i. Recall that M_i is the set of messages sent by P_i. The domain of $Channel_i$, which is also referred to as the state space of channel C_i, is

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

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

The state of the protocol system at any time is given by the value of ($\underline{v}_1$, $\underline{v}_2$, Channel1, Channel2). The global state space is

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

2.3 Entity Events

We first develop notation that will be used in describing entity events. Let $\underline{x}$ denote a vector of variables. A predicate $\underline{F}(\underline{x})$ denotes a boolean function involving variables in $\underline{x}$. The truth set of $\underline{F}(\underline{x})$, denoted by $\underline{F}$, is the set of values of $\underline{x}$ for which $\underline{F}(\underline{x})$ has the value True. Let $\underline{x}$ and $\underline{y}$ be two vectors of variables. An algorithm $U(\underline{x};\underline{y})$ denotes an algorithm (written in a programming language such as Pascal) that accesses the values of variables in $\underline{x}$ and assigns values to variables in $\underline{y}$. $\underline{x}$ and $\underline{y}$ are referred to respectively as the input and output variables of $U(\underline{x};\underline{y})$. The input space of $U(\underline{x};\underline{y})$ refers to the set of values of $\underline{x}$ for which the behavior of $U(\underline{x};\underline{y})$ will be considered. For each value of $\underline{x}$ in the input space, the execution of $U(\underline{x};\underline{y})$ is well-defined (i.e. terminates after assigning a legal value to $\underline{y}$). $U(\underline{x};\underline{y})$ can be non-deterministic. A transition of U is a pair of values (a;b) such that the execution of $U(\underline{x};\underline{y})$ with $\underline{x}$ having the value a can result in $\underline{y}$ having the value b. (a;b) is said to be a transition from a.

There are three types of entity events: send events, receive events, and internal events. The enabling condition of each event is specified by a predicate in variables of the protocol system. When the predicate is true the event is enabled. Its occurrence results in the execution of an algorithm, denoted as the action of the event. The events of entity P_i are specified as follows.

(1) Send_q, for message type $(q;\underline{f}:\underline{F}) \in Q_i$, denotes the event of P_i sending a message of type q into channel C_i. This send event can be specified by the following program description:

```
Enabling Condition : F(v_i)
Action             : U(v_i ; v_i , f);
                     put(channel_i , (q,f))
```

$\underline{F}(\underline{v}_i)$ denotes a predicate involving variables of $\underline{v}_i$; its truth set $\underline{F}$ is a subset of S_i. (If C_i is a finite-buffer blocking channel, then it is also required that C_i is not full for the event to be enabled.) $U(\underline{v}_i;\underline{v}_i;\underline{f})$ is an algorithm with input variables in $\underline{v}_i$, output variables in $\underline{v}_i$ and $\underline{f}$, and input space $\underline{F}$. The variable $\underline{f}$ in Action has domain $\underline{F}$ and can be thought of as a temporary send buffer that exists only during the event occurrence. put($Channel_i$,(q,$\underline{f}$)) appends to the tail of $Channel_i$ a message of type q with field values equal to the current value of $\underline{f}$. (If C_i is a finite-buffer loss channel and is already full, then a message is bumped instantaneously according to the bumping rule.)

For later use, we express Send_q in the notation of [LAM 82b]. Define

```
[Send_q] = {(s,r,-(q,a)) : s ∈ F and (s;r,a) is a transition of U}
```

Each element (s,r,-(q,a)) in [Send_q] is referred to as a send state event and is to be interpreted as follows: the event is enabled if P_i is in state s; its occurrence causes P_i to enter state r and send message (q,a) into channel C_i. [Send_q] is the set of send state events due to Send_q, and is referred to as the set description of Send_q.

(2) Rec_q, for message type $(q;\underline{f}:\underline{F}) \in Q_j$ $(j \neq i)$, denotes the event of P_i receiving a message of type q from channel C_j. This receive event can be specified by the following program description:

```
Enabling Condition : F(v_i) and first(Channel_j) = q
Action             : get(Channel_j , (q,f));
                     U(v_i , f ; v_i)
```

$\underline{F}(\underline{v}_i)$ denotes a predicate involving variables of $\underline{v}_i$; its truth set $\underline{F}$ is a subset of S_i. first($Channel_j$) is a function that indicates the type of the first message (at the head of) $Channel_j$, provided that $Channel_j$ is not empty. When the first message in $Channel_j$ is a message of type q, get($Channel_j$,(q,$\underline{f}$)) removes that message from $Channel_j$ and assigns its field values to the corresponding variables in $\underline{f}$. The variable $\underline{f}$ is Action has domain $\underline{F}$ and can be thought of as a temporary receive buffer that exists only during the

event occurrence. $U(\underline{v}_i,\underline{f};\underline{v}_i)$ is an algorithm with input variables in $\underline{v}_i$ and $\underline{f}$, output variables in $\underline{v}_i$, and input space $\underline{F}$ x $\underline{F}$.

For later use, we express Rec_q in the notation of [LAM 82b]. Define

```
[Rec_q] = {(s,r,+(q,a)) : s ∈ F, a ∈ F and
                          (s,a;r) is a transition of U}
```

Each element (s,r,+(q,a)) in [Rec_q] is referred to as a receive state event and is to be interpreted as follows: the event is enabled if P_i is in state s and (q,a) is the first message in channel C_j; its occurrence removes the message from the channel and causes P_i to enter state r. [Rec_q] is the set of all receive state events due to Rec_q, and is referred to as the set description of Rec_q.

(3) An internal event a of P_i can be specified by the following program description:

```
Enabling Condition : F(v_i)

Action             : U(v_i ; v_i)
```

$\underline{Fv}_i$ denotes a predicate involving variables of $\underline{v}_i$; its truth set $\underline{F}$ is a subset of S_i. $U(\underline{v}_i;\underline{v}_i)$ is an algorithm with both input and output variables in $\underline{v}_i$, and input space $\underline{F}$. Internal events model such events as timeouts and interactions between the entity and its local user.

For later use, we express internal event a in the notation of [LAM 82b]. Define

```
[a] = {(s,r,α) : a ∈ F and (s;r) is a transition of U}
```

where α is a special symbol denoting the absence of a message. Each element (s,r,α) in [a] is referred to as an internal state event and is to be interpreted as follows: the event is enabled if P_i is in state s; its occurrence causes P_i to enter state r. [a] is the set of all internal state events due to a, and is referred to as the set description of a.

(In the description of each event above, in order to avoid cumbersome notation we have omitted qualifiers for the predicate F and the algorithm U that would identify the event to which they belong. We will use qualifiers for $\underline{F}$ and U only when the need arises. Similarly, we will omit the subscript i in $\underline{v}_i$ if there is no ambiguity.)

The set of send events of P_i is $\{send\_q : q \in Q_i\}$. The set of receive events of P_i is $\{Rec\_q : q \in Q_j$ $(j \neq i)\}$. Let A_i denote the set of internal events of P_i. We shall use T_i to denote the set of all events specified for P_i.

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.

For the sake of clarity, we have made the simplifying assumption that a send event appends a message to the end of the message sequence in a channel, and only the first message in a channel is eligible for reception. Except for a special case involving infinite-buffer channels, which we will point out later, our results in this paper are also valid for the case where a send event can insert anywhere into the message sequence in C_i, and any message in C_i is eligible for reception.

2.4 Channel Events

Let E_i denote the set of channel events specified for channel C_i for i=1 and 2. The occurrence of a channel event in E_i depends on and changes only the state of C_i; no other channel or protocol entity is involved. We use such channel events to model channel errors as well as message deletions by a channel "controller."

The following three types of error events are considered: loss events, duplication events, and reordering events. Each event is defined for a specified position of C_i. A loss event for the j-th position of C_i is enabled if $Channel_i$ has a message in that position; its occurrence deletes that message from $Channel_i$. A duplication event for the j-th position of C_i is enabled if $Channel_i$ has a message in that position; its occurrence inserts a duplicate of that message immediately behind it in $Channel_i$. A reordering event from the j-th to the k-th position of C_i is enabled if $Channel_i$ has messages in both these positions; its occurrence moves the message in the j-th position to immediately behind the message in the k-th position.

E_i can be specified to contain any collection of events of each type. The following special case is sometimes considered.

Uniform error model. If E_i contains a loss (duplication) event, then E_i contains loss (duplication) events for every position in C_i. If E_i contains a reordering event, then E_i contains reordering events for every pair of positions in C_i.

If E_i does not satisfy the above condition, then C_i is said to be specified by a nonuniform error model.

We also allow the set of message types Q_i to be specified such that some message types in the set are invulnerable to one or more types of error events. In this case, we include in the enabling condition of each channel event the additional requirement that the message affected (i.e., lost, duplicated, or reordered) must be vulnerable to that type of error.

For consistency, we require that all duplicated events are inhibited if C_i is a finite-buffer blocking channel and it is full. If C_i 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.

Finally, for each channel C_i, we include in E_i a message deletion event that is specified by the following assumption:

Finite life-time assumption. We assume that the message flow in a channel cannot be blocked due to an entity refusing to accept messages indefinitely. The first message in a channel will be deleted by a channel controller, if after a finite time duration, it has not been removed by the receiving entity.

2.5 The System

The protocol system is completely specified by the following for i=1 and 2:

$\underline{v}_i$: S_i, the state vector and state space of P_i;

Q_i, the set of message types sent by P_i;

T_i, the set of entity events of P_i

(T_i = {Send_q : q ϵ Q_i} U {Rec_q : q ϵ Q_j (j $\neq$ i)} U A_i);

E_i, the set of channel events of C_i;

g_0, the initial global state of the protocol system.

The set of events affecting the protocol system is

$$T_1 \cup T_2 \cup E_i \cup E_2$$

We make the assumption that if multiple events in the entities and channels occur simultaneously then such an occurrence can be represented as a sequence of occurrences of the same events in some arbitrary order [KELL 76]. In the present model, this assumption is acceptable if the events of each protocol entity are implemented as indivisible operations. Note that channel events can be considered to occur instantaneously.

Each event in E is enabled over a set of global states and gives rise to a set of global transitions. Recall that τ denotes the set of transitions in the global state space G. Given event e ϵ E and global states g and h (not necessarily distinct) we say that e takes the protocol system from g to h, if e is enabled when the protocol system is at state g and its occurrence can result in the protocol system entering state h.

Recall that the transition system (G, τ) can be represented as a directed graph. 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,...,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.

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 $f_0,f_1,\ldots,f_n,g_0,g_1,\ldots,g_m$. A path x is extendible by sequence y to z = x,y if z is also a path.

3. CONSTRUCTING A FAITHFUL IMAGE PROTOCOL SYSTEM

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

S_i, M_i, T_i, E_i, for i = 1 and 2

in the original protocol system specification. Protocol quantities (entity states, messages or events) in each partition subset are treated as equivalent and are projected onto (aggregated) to form) a single quantity, called their image, in the image protocol system. The resulting image entity states, image messages, and image events specify an image protocol. (We refer the reader to [LAM 82b] for a more detailed exposition.)

For a protocol quantity x, we shall use the notation x' to denote its image. (The same notation x' will sometimes also be used to refer to the set of protocol quantities in the original protocol system that have the x' image .) For a set x of protocol quantities in the original system specification, we use x' to denote the set of image quantities derived from x, i.e.,

$x' = \{x' : x \in x\}$.

In Section 3.1, we present the procedure for constructing an image protocol, given a projection of S_i specified by a subvector $\underline{v}_i'$ of $\underline{v}_i$. (We will explain in Section 3.2 how these subvectors are obtained.)

All states in S_i that have the same values for variables in $\underline{v}_i'$ are treated as equivalent and have the same image. S_i' is the set of images of S_i. Following this, the projection of M_i is done by partitioning Q_i. All message types in each partition subset are projected onto an image message type. Q_i' is the set of image message types obtained from Q_i. In the projection of T_i, all Send_q (Rec_q) events involving message types that have the same image q' are projected only a Send_q' (Rec_q') event in T_i'. The internal events in A_i are projected onto image internal events in A_i'.

The resulting image protocol system is just like any real protocol system, and is specified by:

For i = 1 and 2

$\underline{v}_i'$: S_i', the state vector and state space of P_i

Q_i', the set of message types sent by P_i;

T_i', the set of image entity events of P_i

($T_i' = \{Send\_q' : q' \in Q_i'\} \cup \{Rec\_q' : q' \in Q_j' \ (j \neq i)\} \cup A_i'$);

E_i', the set of image channel events of C_i;

g_0', the image initial state of the protocol system.

In general, this image protocol need not be faithful. In Section 3.2, we state our definition of faith-

fulness, and a well-formed property which, if satisfied by the image protocol entities, is a sufficient condition for faithfulness [LAM 82b]. Using the well-formed property, we describe an iterative procedure to obtain the image vectors $\underline{v}_1'$ and $\underline{v}_2'$ needed for the construction procedure of Section 3.1.

Briefly, an initial choice of $\underline{v}_i'$ is made by considering the desired logical behavior (assertion) that has to be verified. Then, more variables are added to $\underline{v}_i'$ so that the image entity events satisfy the well-formed property.

The construction procedure described in this section is obtained from [LAM 82b] and involves set operations (right side of Figure 3). That is, we have to transform the protocol system description from program notation to set notation, use set operations to obtain faithful image protocols, and then retransform the set notation into program notation (see Figure 3).

In Section 4 we will show that if the program descriptions of entity events display certain nice structures, then faithful image protocol systems can be obtained directly (by inspection) from the original protocol system (this corresponds to path involving simple operations in Figure 3).

In practice, we have found that the results of Section 4 can be used to construct most of the image protocol. Those portions of the image protocol that cannot be constructed in this way, can be constructed by the general construction procedure in this Section 3.

3.1 Constructing an Image Protocol System

We start by considering the state vector $\underline{v}_i$ in each entity P_i. Its image $\underline{v}_i'$ is the subvector of $\underline{v}_i$ consisting of the variables that we choose to retain in the image protocol (we will explain in Section 3.2 how this subvector is chosen). $\underline{v}_i'$ is referred to as the image state vector of P_i.

3.1.1 Projection of entity states

For each entity state $\underline{s} \in S_i$, its image $\underline{s}'$ is obtained by deleting all components in $\underline{s}$ that are not in $\underline{v}_i'$. S_i' is the set of images of states in S_i. Note that all entity states in S_i that do not differ in the values of $\underline{v}_i'$ are projected onto a single image in S_i'. S_i' is the domain of $\underline{v}_i'$, and is referred to as the image state space of P_i. Elements of S_i' are referred to as image entity states.

3.1.2 Projection of message types

The projection of the message types in Q_i depends upon the projections of the entity state spaces.

A message type $(q,\underline{f}:\underline{F}) \in Q_i$ can be projected onto a null image, denoted by (β), if no message of type q causes state changes in the image entity state space S_j' $(j \neq i)$; i.e., every tuple $(s,r,(q,a)) \in [Rec\_q]$ has $s'=r'$. (β) is treated as a message type with no fields; hence, (β) is also a message (of type β).

Message types $(q_1;\underline{f}_1:\underline{F}_1)$; $(q_2;\underline{f}_2:\underline{F}_2),\ldots,(q_n;\underline{f}_n:\underline{F}_n)$ with non-null images in Q_i can be projected onto the image $(q';\underline{f}':\underline{F}')$ if

a) $\underline{F}'$ is the product of field spaces common to $\underline{F}_1,\underline{F}_2,\ldots,\underline{F}_n$, and

b) for any q_i and q_j, a message of type q_i and a message of type q_j cause identical state transitions in S_j' provided they have the same values in the fields corresponding to $\underline{F}'$.

(Note that the names of the fields in $\underline{f}'$ can differ from the names of the corresponding fields in $\underline{f}_i$; they are only parameters.)

Let Q_i' be the set of images of message types of Q_i. We refer to the image of message type $(q;\underline{f}:\underline{F}) \in Q_i$ by the notation $(q';\underline{f}':\underline{F}')$. For any tuple $\underline{a} \in \underline{F}$, its image $\underline{a}'$ is defined to be the subvector of $\underline{a}$ corresponding to field spaces in $\underline{F}'$. The image of message $(q;\underline{a})$ is then defined to be $(q',\underline{a}')$.

The image message set $M_i' = \{m' : m \in M_i\}$ sent by P_i can be obtained as the union of message sets specified by the image message types in Q_i'.

Sometimes it is desirable to have a finer partitioning of a message set. A higher resolution may be needed to make certain logical statements about the behavior of the protocol system, or to satisfy the well-formed property for some receive events (see below). A stronger criterion for treating message types as equivalent is to require also that their send events cause identical state changes in the image state space of their sender.

Also if we permit some message types in Q_i to be invulnerable to the occurrence of one or more types of channel events, or the bumping rule to depend upon the message types, then two message types may be treated as equivalent only if they have the same characterization of invulnerability and are treated as identical by the bumping rule. One consequence of this generality is that multiple null image message types are needed in Q_i', i.e., we have to define different β's, one for each combination of priority class and invulnerability.

We shall deal with a null image message type (β) in two different ways depending upon whether C_i is a finite-buffer channel or an infinite-buffer channel. With a finite-buffer channel, (β) is included in Q_i' as described above. With an infinite-buffer channel, (β) may be excluded from Q_i'. Note that the reception of a null image message does not affect the image state of the receiving entity. The fact that a null image message does occupy a buffer in C_i can be ignored for an infinite-buffer channel. Thus, (β) can be eliminated from Q_i' if the following assumptions are made. First, the insertion and deletion of messages associated with send and receive events respectively must take place at the two end points of the message sequence in C_i. Second, C_i must be characterized by the uniform error model. From now on, an infinite-buffer channel C_i will be considered with the implicit assumption that (β) has been deleted from Q_i' and the uniform error model applies.

3.1.3 Images of channel states, global states, and sequence of global states

The preceding equivalence relation defined for messages is now extended to channel states, global states, and sequences of global states.

The image $\underline{m}_i'$ of channel state $\underline{m}_i$ is obtained by taking the image of each message in $\underline{m}_i$. In addition, for infinite-buffer channels, messages with a null image are deleted from $\underline{m}_i'$. $\underline{M}_k'$, the state space of channel C_i in the protocol is the set of images of channel states in $\underline{M}_i$.

We will continue to use the name $Channel_i$ to denote the variable representing the message sequence at any time in C_i in the image protocol.

The image of global state $g = (s_1,s_2; \underline{m}_1,\underline{m}_2)$ is defined to be $g' = (s_1',s_2'; \underline{m}_1',\underline{m}_2')$. G', the global state space of the image protocol is the set of images of global states in G. Thus, we have $G' = (S_1' \times S_2' \times M_1' \times M_2')$.

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 in the sequence are replaced by a single occurrence of the image. 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.

3.1.4 Projection of events

The image of an event denotes the effect of the event that can be observed in the image global state space. The image of an event itself becomes an event in the image protocol [LAM 82b]. It has been shown in [LAM 82b] that the set of images of channel events in E_i is E_i itself. Hence, the set E_i' of channel events of C_i in the image protocol is E_i itself.

We now describe the images of the entity events. The images of the three types of entity state events (s,r,+m), (s,r,-m) and (s,r,α) are (s',r',+m'), (s',r',-m') and (s',r',α) respectively. The image of an entity state event is also referred to as an image entity state event. The image of an entity state event describes the effect of the event that can be observed in the image global state space. A state event whose occurrence does not affect the image global state is said to be a null image state event. An image internal state event (s',r',α) is a null image state event if s' = r'.

For each image message type $(q',\underline{f}':\underline{F}') \in Q_i'$, the image event Send_q' is defined in set notation by

```
[Send_q'] = {(s',r',-(q',a')) : for some (q,f:F) whose image
                                is (q',f':F')}
                                (s,r,-(q,a)) ∈ [Send_q]
```

Send_q' is the image of all Send_q events where the image of q is q'. Recall that the set notation is

transformable to a program notation. To be precise, we must find (a) a predicate $E(\underline{v}_i')$ with truth set {s' : (s',r',-(q',a')) ϵ [Send_q']}, and (b) an algorithm $U(\underline{v}_i';\underline{v}_i';\underline{f}')$ with input variables $\underline{v}_i'$, output variables $\underline{v}_i'$ and $\underline{f}'$, and input space $\underline{F}$ such that [Send_q'] = {(s',r',-(q',a')) : s' ϵ $\underline{F}$ and (s',r',a') is a transition of U}. Then, the event Send_q' can be specified in program notation by

```
Enabling Condition : F(v_i'
Action             : U(v_i'; v_i', f'));
                     put(Channel_i, (q',f'))
```

For each image message type $(q';\underline{f}':\underline{F}') \epsilon Q_i'$, the image event Rec_q' is specified in state notation as

```
[Rec_q'] = {(s',r',+(q',a')) : (s,r,+(q,a)) ε [Rec_q], q is any
                               one message type with image q'}
```

To specify Rec_q' in program notation, it is necessary to find (a) a predicate $E(\underline{v}_j)$ with truth set E = {s' : (s',r',(q',a')) ϵ [Rec_q']}, and 8b) an algorithm $U(\underline{v}_j';\underline{f}';\underline{v}_j')$ with input variables $\underline{v}_j'$ and $\underline{f}'$, output variables $\underline{v}_j'$, and input space $\underline{F}$ x $\underline{F}'$ such that [Rec_q'] = {(s',r',+(q',a')) : s' ϵ $\underline{F}$, a' ϵ $\underline{F}'$, and (s';a',r') is a transition of U}. Then, Rec_q' can be specified in program notation as:

```
Enabling Condition : E(v_j) and first(Channel_i) = q'
Action             : get(Channel_i, (q',f'));
                     U(v_j', f'; v_j')
```

For each internal event a ϵ A_i, the image of [a] is defined to be [a]' = {(s',r',α) : (s,r,α) ϵ [a], (s',r',α) is not a null image event}. In particular, if for internal event a ϵ A_i, [a]' has only null image state events, then the image of a is said to be the null image event.

Internal events $a_1,a_2,\ldots,a_n \epsilon A_i$ can be projected onto the same image internal event a' if for all a_i, a_j, $[a_i]'$ differs from $[a_j]'$ only in null image state events. The image internal event a' is specified in state notation as

```
[a'] = {(s',r',α) : (s,r,α) ε [a] for some a whose image
                    is a', and (s',r',α) is not a null image event}
```

Let A_i' be the set of images of events in A_i, where null image events are not included.

To specify image internal event a' in program notation it is necessary to find (a) a predicate $\underline{F}(\underline{v}_i')$ with truth set $\underline{F}$ = {s' : (s',r',α) ϵ [a']}, and (b) an algorithm $U(\underline{v}_i';\underline{v}_i')$ with both input and output variables in $\underline{v}_i'$, and input space $\underline{F}$ such that [a'] = {(s',r',α) : s' ϵ E and (s',r') is a transition of U}. Then internal event a' can be specified in program notation as:

```
Enabling Condition : F(v_i')
```

Action : $U(\underline{v}_i';\underline{v}_i')$

(Null image events can be added [a'] if that would make it more convenient to specify $\underline{F}(\underline{v}_i')$ and $U(\underline{v}_i';\underline{v}_i')$.)

If channel C_i is an infinite-buffer channel then send and receive events involving null image message (β) can be treated as internal events. The image send state event (s',r',-**b**) is represented as (s',r(,α), and is a null image if s'=r'. Then the event Send_β in T_i' is instead placed in A_i' (in the program description of Send_**b**, $\underline{f}$' and put(Channel$_i$,(β) are removed). The image receive state event (s',r',+β) can be treated as a null image event since s'=r' by definition. Hence Rec_β in T_i' can be entirely removed.

3.2 Faithful Image Protocols

Given a protocol system and a choice of image state vectors $\underline{v}_1'$ and $\underline{v}_2'$, we have shown how to construct an image protocol specified by:

$\underline{v}_i'$: S_i'; Q_i'; T_i'; E_i; g_0' for i = 1 and 2.

In general, this image protocol system need not be faithful. Let R' denote the reachability graph from g_0' of the image protocol system. For image protocols constructed as described in Section 3.1, if g is a reachable state in the original protocol system, then its image g' is reachable in the image protocol system [LAM 82b].

Informally, an image protocol is said to be faithful if the following is true: any logical property, safety or liveness, that can be stated using the resolution of the image protocol, holds for the original protocol if and only if it holds for the image protocol.

Definition (Faithful Image Protocol). For every reachable global state g' of the image protocol system, the following two conditions hold for every reachable global state f of the original protocol system, whose image is g':

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

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

The image protocols constructed as described in Section 3.1 always satisfy condition (F1) [LAM 82b]. The well-formed conditions are next described.

Definition. For a and b in S_i, b is <u>internally reachable</u> 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.

Note that a Send_q event involving P_i and C_i, can be regarded as an internal event for the above definition if C_i is an infinite-buffer channel and the message being sent has a null image. Under these conditions, a send state event (a,b,-m) ϵ [Send_q] can be regarded as the internal state event (a,b,α).

Definition. An image internal event a' ϵ A_i' is well-formed if for each (s',r',α) ϵ [a'] where s' $\neq$ r' the following holds: for every b whose image is s', there is some c ϵ S_i that is internally reachable from b, and (c,d,α) ϵ T_i for some d whose image is r'.

Definition. An image send event Send_q' of P_i is well-formed if for each (s',r',+(q',a')) ϵ [Send_q'] the following holds: for every b whose image is s', there is some c ϵ S_i that is internally reachable from b and (c,d,(p,e)) ϵ [Send_p] for some d whose image is r' and for some (p,e) whose image is (q',a').

Definition. An image receive event Rec_q' of P_i is well-formed if for each (s',r',-(q',a')) ϵ [Rec_q'] the following holds: for every b whose image is a' and for every (p,e) whose image is (q',a'), there is some c ϵ S_i that is internally reachable from b, and (c,d,-(p,e)) ϵ [Rec_p] for some d whose image is r'.

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

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

The image protocol is said to be well-formed (strongly well-formed), if every image entity event in T_i' for i = 1 and 2 is well-formed (strongly well-formed). An image protocol with well-formed protocol entities is faithful [LAM 82b].

The well-formed property is used to determine the state vectors $\underline{v}_1'$ and $\underline{v}_2'$ in the image protocol as follows.

First, obtain the variables $\underline{v}_1$ and $\underline{v}_2$ needed to state the desired logical behavior of the function being projected. Let $\underline{v}_1'$ and $\underline{v}_2'$ consist of these variables. Obtain the image protocol for this choice of $\underline{v}_1'$ and $\underline{v}_2'$, and check if it is well-formed. If it is not well-formed, then it means that more variables from $\underline{v}_i$ are needed in $\underline{v}_i'$ (i.e., a higher resolution is called for). These variables can be detected in the course of checking for well-formedness. Add these variables to $\underline{v}_i'$ and construct an image protocol again. Keep repeating until a well-formed image protocol is obtained. The procedure will terminate at the worst case with the image protocol being the original protocol itself. It has been our experience that for protocols with several distinguishable functions, well-formed image protocols can be constructed for individual functions, that are substantially smaller than the original protocol [SHAN 82b].

4. IMAGE PROTOCOLS EASILY CONSTRUCTED

In Section 3, we described a general procedure to construct faithful image protocols. The construction procedure involved set operations, which we would like to avoid. In this section we consider protocol entities that have events whose program descriptions have certain simple structures. Given these simple structures, we show that image message types and the program descriptions of well-formed image events may be constructed by inspection. The cases considered here have been used to substantially simplify the construction of well-formed image protocols for a version of HDLC [SHAN 82b].

The structures of the entity events considered here are not intended to cover all situations where the program description of entity events displays structure that facilitates the obtaining of well-formed image protocols. In general, we note that the more well-structured the program description, the easier it will be to find the program description of the image events.

We first introduce some notation. Let $\underline{x}$ and $\underline{y}$ be two vectors of variables. Let $\underline{x}$' and $\underline{y}$' be subvectors of $\underline{x}$ and $\underline{y}$ respectively. Given two predicates $\underline{F}_1(\underline{x})$ and $\underline{F}_2(\underline{x})$, $\underline{F}_1(\underline{x}) = \underline{F}_2(x)$ means that their truth sets are equal. If predicate $\underline{F}(\underline{x})$ involves only variables in $\underline{x}$', we say $\underline{F}(\underline{x}) = \underline{F}(\underline{x}')$.

Given two algorithms $U(\underline{x};\underline{y})$ and $U_2(\underline{x};\underline{y})$, with the same input space E, $U_1(\underline{x};\underline{y}) = U_2(\underline{x};\underline{y})$ means that both U_1 and U_2 have the same set of transitions originating from E. For a given input space, if $U(\underline{x};\underline{y})$ accesses variables only in $\underline{x}$' and assigns values to variables only in $\underline{y}$', then we say $U(\underline{x};\underline{y})$ (using the set of transitions of U, we can formally define when a variable in $\underline{x}$ is accessed and when a variable in $\underline{y}$ is assigned a value by U).

In what follows, $\underline{v}$" denotes the vector of variables in $\underline{v}_i$ that are not in the image vector $\underline{v}_i'$, for i = 1 and 2. (We will as usual omit the subscript i for $\underline{v}_i$ unless confusion arises. Proofs of these observations are in the appendix.)

Observation 1. The image a' of internal event $a \in A_i$ is a null image event if and only if $U(\underline{v};\underline{v}) = U"(\underline{v};\underline{v}")$.

Observation 2. If internal event $a \in A_i$ has its enabling condition $E(\underline{v}) = E'(\underline{v}') \wedge E"(\underline{v}")$, and its action $U(\underline{v};\underline{v}) = U"(\underline{v};\underline{v}"); U'(\underline{v}';\underline{v}')$, then

a) the image of $a' \in A_i$ can be specified by:

Enabling Condition : $E'(\underline{v}')$

Action : $U'(\underline{v}';\underline{v}')$

b) if $E(\underline{v} = E'(\underline{v}')$ above, then a' is strongly well-formed.

Observation 3. A message type $(q;\underline{f}:\underline{F}) \in Q_i$ has a null image (β) if and only if Rec_q has its action algorithm $U(\underline{v},\underline{f};\underline{v}) = U''(\underline{v},\underline{f};\underline{v}'')$.

Recall that if channel C_i is an infinite-buffer channel, then for any message type $(q,\underline{f}:\underline{F}) \in Q_i$ with null image (β), the image of Rec_q is not included in the image protocol.

Observation 4. Let channel C_i be a finite-buffer channel and let $(q_1;\underline{f}_1:\underline{F}_1),\ldots,(q_n:\underline{f}_n:\underline{F}_n)$ be the message types in Q_i with null image (β). If Rec_q_k, for $i \leq k \leq n$ has its enabling condition predicate $E_k(\underline{v}) = E'_k(\underline{v}') \wedge E''_k(\underline{v}'')$, and its action algorithm $U_k(\underline{v},\underline{f}_k;\underline{v}) = U''_k(\underline{v},\underline{f}_k;\underline{v}'')$, then

a) Rec_(β) in the image protocol can be specified by

Enabling Condition : $(E'_1(\underline{v}') \vee E'_2(\underline{v}') \vee \ldots \vee E'_n(\underline{v}'))$
and first($Channel_i$) = β

Action : get($Channel_i$, (β))

b) If $E_k(\underline{v}) = E'_k(\underline{v}')$ above, then Rec_β is strongly well-formed.

Observation 5. For message types $(q_1:\underline{f}_1:\underline{F}_1)$, $(q_2:\underline{f}_2:\underline{F}_2),\ldots,(q_n:\underline{f}_n:\underline{F}_n)$ in Q_i, if Rec_q_k for $i \leq k \leq n$ has

a) its enabling condition predicate $E_k(\underline{v}) = E'(\underline{v}') \wedge E''_k(\underline{v}'')$, and

b) its action algorithm $U_k(\underline{v},\underline{f}_k;\underline{v}) = U''_k(\underline{v},\underline{f}_k;\underline{v}'')$; $U'(\underline{v}';\underline{f}'_k;\underline{v}')$,

where $(q_k;\underline{f}_k:\underline{F}_k) = (q_k;\underline{f}'_k:\underline{F}',\underline{f}''_k:\underline{F}''_k)$, then

c) $(q_k;\underline{f}_k:\underline{F}_k)$ can be projected onto the image message type $(q';\underline{f}':\underline{F}')$,

d) Rec_q' can be specified by:

Enabling Condition : E'($\underline{v}'$) and first($Channel_i$) = q'

Action : get($Channel_i$, (q',$\underline{f}'$));
U'($\underline{v}'$,$\underline{f}'$;$\underline{v}'$)

e) If in (a) above, $E_k(\underline{v}) = E'(\underline{v}')$ for $1 \leq k \leq n$, then
Rec_q' is strongly well-formed.

Observation 6. If message types $(q_1;\underline{f}_1:\underline{F}_1),\ldots,(q_n;\underline{f}_n:\underline{F}_n)$ in Q_i are projected onto image message $(q';\underline{f}':\underline{F}')$, and Send_$q_k$ for $1 \leq k \leq n$ has

a) its enabling condition predicate $E_k(\underline{v}) = E'_k(\underline{v}') \wedge E''_k(\underline{v}'')$, and

b) its action algorithm $U_k(\underline{v};\underline{f}_k,\underline{v}) = U''_k(\underline{v};\underline{f}''_k,\underline{v}'')$; $U'_k(\underline{v}';\underline{f}'_k,\underline{v}')$

where $(q_k;\underline{f}_k:\underline{F}_k = (q_k;\underline{f}_k':\underline{F}_1")$, then

c) Send_q' is specified by

Enabling Condition : E_k') v $E_2'(\underline{v}')$ v...v $E_n'(\underline{v}')$

Action : {Choose any k such that $E_k(\underline{v}') = True$}

$U_k'(\underline{v}';\underline{f}',\underline{v}')$;
put(Channel$_i$, $(q',\underline{f}')$)

d) Send_q' is strongly well-formed if the set $\{q_1,q_2,\ldots,q_n\}$ can be partitioned so that each partition subset $\{q_{k1},q_{k2},\ldots,q_{km}\}$ has

$E_{k_1}'(\underline{v}') = E_{k_2}'(\underline{v}') = \ldots = E_{k_m}'(\underline{v}')$,

$E_{k_1}"(\underline{v}")$ v $E_{k_2}"(\underline{v}")$ v...v $E_{k_m}"(\underline{v}") = True$, and

$U_{k_1}'(\underline{v}';\underline{f}',\underline{v}') = \ldots = U_{k_m}'(\underline{v}';\underline{f}',\underline{v}')$

In Observation 6, part (c), we note that if q' = β and channel C_i is an infinite-buffer channel, then "put(Channel$_i$,(q',f'))" is not included. Further, if $U_k(\underline{v};\underline{f}_k,\underline{v}) = U_k"(\underline{v};\underline{f}_k,\underline{v}")$ for $1 \leq k \leq n$, then Send_β is a null event and is not included in the image protocol.

In applying these observations to obtain well-formed image protocols, the following guidelines can be used.

First, obtain the image vectors $\underline{v}_1'$ and $\underline{v}_2'$ by retaining all variables in $\underline{v}_1$ and $\underline{v}_2$ needed to state the desired logical behavior of the protocol system. (These would be the variables present in the assertion that has to be verified.) Next, include more variables in $\underline{v}_i'$ so that the program descriptions of a substantial number of entity events display the structure shown in the observations. (This amounts to expanding $\underline{v}_i'$ so that only variables in $\underline{v}_i'$ are used in assigning values to variables in $\underline{v}_i'$.) Construct the image protocol system, using the set operations in Section 3 for those entity events that display no suitable structure. Check if it is well-formed. If not, include the required variables (indicated when the well-formed check fails) in $\underline{v}_i'$ and repeat the procedure. In practice this turns out to be quite easy to do for many communication protocols. It has been applied successfully to a version of HDLC [SHAN 82c].

5. TIME-DEPENDENT COMMUNICATION PROTOCOLS

In many communication protocol systems, a component is obliged to satisfy some constraints on the time intervals between occurrences of system events involving that component. (e.g., an entity must send an acknowledgement to a message within a specified response time of receiving that message; the time duration that a message resides in a channel is less than a specified maximum channel delay.) Timers, clocks, and age fields in packets are used to implement these time constraints in the protocol system.

Because (physical) time elapses at the same rate everywhere, these time constraints satisfied locally by components give rise to precedence relations between remote events in different components. Note that such precedence relations are not established through a chain of system events. If these precedence relations are vital to the proper functioning of the protocol system, then many logical correctness properties of importance cannot be verified without including measures of time in the modeling. We refer to such systems as time-dependent systems [SHAN 82a, FLET 78, SLOA 79].

In our current model of a protocol system (as defined in Section 2) there are entity send events, entity receive events, entity internal events and channel events. Note that a send or a receive event involves only two components (an entity and a channel) of the system, and an entity internal event or a channel event involves only one component of the system. We shall refer to these events of our model as system events, inasmuch as they correspond to events that occur in the protocol system being modeled. This current protocol model with only system events is inadequate to model time-dependent systems [SHAN 82a]. It is necessary to relate the elapsed times measured at different components. To do this we use time variables and time events, to be explained in Section 5.1. In Section 5.2 we incorporate these time variables and time events into our protocol model. In Section 5.3, we extend the method of projections to this time-dependence protocol model.

5.1 Time Variables and Time Events

Time variables are variables that indicate elapsed time in integer ticks. Time events are events that increase the elapsed time indicated by time variables.

Each time variable takes its values from N_t = {Off,0,1,2,...}. A time variable is termed inactive if its value is Off, else it is termed active. In modeling a protocol system, whenever it becomes necessary to measure an elapsed time in a component, we include a time variable in the component. The value of a time variable can be changed in only two ways. First, it can be aged by a time event. When an active time variable is aged, its value is incremented by 1; when an inactive time variable is aged, its value is not affected. Second, a time variable in a component can be reset to any value in N_t by a system event involving that component. Thus, for an active time variable, the difference between its current value and the value it was last reset to, indicates the time elapsed since the last reset.

We will use two types of time variables in our model: global time variables and local time variables. All global time variables in a system model are aged by the same time event, referred to as the global time event. Thus, all active global time variables are coupled. The global time event models the elapse of physical time in the protocol system model.

In general, a global time variable does not correspond to an actual timer implemented in the

protocol system. Global time variables are typically used as auxiliary variables that are needed for stating assertions about the system's behavior, or to model time constraints satisfied by components.

Local time variables are used to model the timers and clocks that are implemented in system components. Every local time variable t is associated with a unique time event, referred to as the local time event of t. t can be aged only by its local time event, and the local time event of t can age only t. This decouple the aging of t from the aging of any other time variable.

With each local time variable t, we associate a global time variable t^* and a reset value t_0. Whenever t is reset, both t^* and t_0 are reset to the same value. t^* is affected by the global time event just like any other global time variable. Therefore, at any time, t^* indicates the value that t should have if t were aged by the global time event (note that t is active if and only if t^* is active). The accuracy of local time variable t is specified by its accuracy axiom which bounds $t-t^*$ at any time (Off-Off is treated as 0). For example, the condition $|t-t^*| \leq 1$ can specify a timer with no accumulating error; the condition $|t-t^*| \leq 1 + a(t^*-t_0)$ can specify a timer with maximum relative error a in the clock frequency.

In this model, neither the local time event of t nor the global time event can occur, if such an occurrence would violate the accuracy axiom. (The meaning of this will be made clear in the discussion on time constraints below.)

The accuracy axiom for each local time variable t cannot be chosen arbitrarily. For example, the accuracy axiom $|t-t^*| \leq 0$ would deadlock the local time event of t and the global time event. In this paper, we insist that the accuracy axiom for any local time variable t is an upper bound on $|t-t^*|$ which is monotonically non-decreasing both in t and in t^*, and whose minimum value is 1. This ensures that the time events do not get deadlocked due to unreasonable accuracy axioms. (There are weaker sufficient conditions.)

Time constraints

Time variables and time events can be used to model time constraints in components. Let e_1 and e_2 denote two system events of a component. Let t be a time variable (local or global) that is reset to 0 by e_1, reset to Off by e_2, and not reset by any other system event. Let D denote a specified time period.

The time constraint that e_2 will not occur until D time units have elapsed since the occurrence of e_1, can be modeled by including $(t \geq D)$ as part of the enabling condition of e_2.

Another example of a time constraint is that e_2 will occur not later than D time units after the occurrence of e_1. This cannot be modeled by including $(t \leq D)$ in the enabling condition of e_2. (This is

because in our model an enabled event does not have to occur.) However, it can be modeled by including ($t < D$) in the enabling condition of the time event for t. This particular interpretation does not mean that in the distributed system, time comes to a halt whenever t equals D. Rather, it models the guarantee that the component will execute the event e_2 before (or as soon as) t reaches D. Such guarantees by a component are modeled by enabling conditions for time events, and will be referred to as the components local time axioms.

Note that in the above situation, the component cannot guarantee the local time axiom if e_2 is not enabled some time before t reaches D, or if e_2 is a receive event, or if e_2 is a send event which may be blocked. In short, a local time axiom, like the accuracy axioms, cannot be any arbitrary time constraint. It must be a constraint that the component can guarantee without any cooperation from its environment. If that is the case, then in the protocol system model, the time events will never be deadlocked.

Note that the accuracy axiom for a local time variable t, considered earlier, is a special case of a local time axiom; the module guarantees that the variable t is incremented at (more or less) the same rate as the global time event occurs. We shall only consider systems whose modules satisfy their local time axioms.

5.2 A Time-Dependent Protocol System Model

We now augment the original protocol system model with time variables and time events. In each entity P_i, some of the variables in $\underline{v}_i$ can be global or local time variables. For each local time variable in $\underline{v}_i$, the associated global time variable and associated reset value are also considered to be in $\underline{v}_i$. The accuracy axiom for each local time variable v in $\underline{v}_i$ is specified by upper and lower bounds or the difference between v and its associated global time variable. The bound itself may be a function of the reset value associated with v.

(Henceforth, when we refer to time variables of P_i, we do not mean the global time variables associated with the local time variables of P_i.) We assume that the local time axioms of P_i can be represented by stating the set of values allowed for each time variable of P_i. Hence, the domain $S_{i,j}$ of time variable $v_{i,j}$ is either a proper subset of N_t (if there is a local time axiom restricting $v_{i,j}$), or is equal to N_t (if $v_{i,j}$ is not involved in a local time axiom). Each of the variables associated with local time variables has a domain N_t. The state space S_i of entity P_i is the product of all the $S_{i,j}$'s (as defined earlier in Section 2.2).

In channel C_i, we associate with every message in transit a time value that indicates the time spent by that message in the channel. This time value is referred to as the age of the message. For notational convenience we will assume that this age is a global time value. The state of channel C_i is given by the

sequence of (message, age) pairs in C_i. We will continue to denote the state space of channel C_i by $\underline{M}_i$. (A definition of this $\underline{M}_i$ can be obtained from the former definition of $\underline{M}_i$ (Section 2.2) by treating M_i^j as the cartesian product of (M_i x N_t) with itself j times.) In Section 2.2, $Channel_i$ represented the message sequence in channel C_i. We now redefine $Channel_i$ to represent the sequence of (message, age) pairs in channel C_i, at any time. When a message is sent into a channel, its age is set to 0.

We now allow channel C_i to be a channel with a specified maximum delay denoted by $MaxDelay_i$. A message that stays in C_i for longer than $MaxDelay_i$ is lost. We model this with the following local time axiom for channel C_i: every age value in $Channel_i$ is less than or equal to $MaxDelay_i$.

Entity Events

The events of protocol entity P_i are as defined before (in Section 2.3). The values of time variables can be used in the events to model time constraints. The time variables can be reset in the events. We use the notation Reset(v,a) to represent a program statement that resets time variable v to the value $a \in N_t$. If v is a local time variable, then its associated global time variable and reset value are also set to a by Reset(v,a).

To maintain the meaning of time variables, we insist that any change done to a time variable v in $\underline{v}_i$ is through the Reset(v,a) program statement. We also insist that associated global time variables and reset values cannot be accessed by the entity events. This is because the sole purpose of the associated variables is to model the accuracy of local time variables. (Indeed, if P_i can access the global time variable associated with a local time variable v, then we could just as well have modeled v as a global time variable in the first place.)

Channel Events

The channel events are as described in Section 2.4 except that now the events act on (message, age) pairs. Also, if channel C_i satisfies the local time axiom $Channel_i(Age) < MaxDelay_i$, then for the axiom to be valid we must include in E_i a message deletion event that deletes any message whose age is $MaxDelay_i$. Note that this message deletion event need not be included (again) if either E_i already has the message loss event (Section 2.4), or C_i is non-reordering (recall that the message deletion event of Section 2.4 is still specified in E_i).

Time Events

We now define the local time events and the global time event for the protocol model.

For each local time variable $v_{i,j}$ in $\underline{v}_i$, i=1 and 2, there is a local time event whose occurrence ages v by 1 tick if v is active. This local time event is enabled if its occurrence does not cause $v_{i,j}$ to violate its time axiom $S_{i,j}$, nor cause $v_{i,j}$ (and its associated variables) to violate its accuracy axiom $ACC_{i,j}$.

For all the global time variables of the protocol model, there is one global time event whose occurrence ages all active global time variables by 1 tick. This includes the age values in $Channel_i$ and the associated global time variables in $\underline{v}_i$ for i=1 and 2. This global time event is enabled if its occurrence does not cause

(i) a global time variable $v_{i,j}$ to violate its time axiom $S_{i,j}$; and

(ii) the global time variable associated with a local time variable $v_{i,j}$ to violate its accuracy axiom $ACC_{i,j}$; and

(iii) if channel C_i has bounded delay $MaxDelay_i$, then no packet age exceeds $MaxDelay_i$.

5.3 Faithful Image Protocols

In Sections 3 and 4 we have described how to obtain faithful image protocols from a given non time-dependent protocol system. We now augment those construction procedures to handle the time-dependent protocol system described in Section 5.2.

Image Protocol Entity

When specifying the image state vectors $\underline{v}_i'$ a local time variable is included in $\underline{v}_i'$ if and only if its associated variables are also included in $\underline{v}_i'$. For each time variable included in $\underline{v}_i'$, its local time axiom and accuracy axiom (if any) are identical to those in the original protocol system.

The projection of messages, entity events, and the definition of well-formed is as defined in Section 3. Both the general construction procedure of Section 3 and the special construction procedures of Section 4 are applicable. Because of the restrictions on the usage of time variables in entity events, the associated variables in v_1 and v_2 need not be examined while constructing image protocols. Note that Reset statements in the entity events of the original protocol are either excluded or included in the image protocol.

Image Channels

In projecting a channel state onto its image, a (message, age) pair (m,a) is replaced by (m',a). If m' is β and the channel is an infinite-buffer channel, then the (message, age) pair deleted.

The set of channel events in the image protocol remain the same as in the original protocol.

Image Time Events

The image local time events and the image global time event in the image protocol are defined as in Section 5.2, except that the time variables are now from $\underline{v}_i'$ (instead of from $\underline{v}_i$).

We will next state the conditions for the image time events to be well-formed. Let $\underline{t}_i''$ denote those time variables in $\underline{v}_i$ that are not included in $\underline{v}_i'$. (Associated global time variables are not included in $\underline{t}_i''$) Let S_i'' denote the domain of $\underline{t}_i''$. S_i'' is the product of the domains $s_{i,j}$ of the time variables in $\underline{t}_i''$. For any value $b'' \in s_i''$, let (b'') denote the result of aging each value in b".

Definition The image global time vents are well-formed if for every value $a'' \in S_i''$, there is a sequence of null-image internal events of P_i that will take $\underline{t}_i''$ from a" to a value b" such that $next(b'') \in S_i''$.

Note that the above definition does not distinguish between local time variables and global time variables, nor between local time events and the global time event. Furthermore, the accuracy axioms for local time variables do not enter the definition; only the local time axioms $S_{i,j}$ are used. (In fact, the above definition implicitly treats all time variables as global time variables.)

Theorem. If all image system events and image time events are well-formed, then the image protocol is faithful. (The proof is in the appendix.)

6. CONCLUSION

The method of projections, described in [LAM 82b], is an approach to reduce the complexity of analyzing nontrivial communication protocols with several distinguishable functions. The protocol model used in [LAM 82b] was a very general one, in which the construction of faithful image protocols involved set operations that are difficult when large protocols are considered.

In this paper, we have described a protocol system model in which the method of projections can be applied without the need for set operations in most cases. Each protocol entity is modeled as an event-driven process that sends and receives messages into channels, and manipulates a set of variables local to itself. The events of the entity are specified in a programming language notation. The messages in the protocol model can have multiple fields (a feature found in many communication protocols) and are specified in terms of message types. The model also has time variables and time events so that it can be applied to (time-dependent) protocol systems where measures of time (obtained by timers and clocks in entities, and age fields in packets) play a crucial role in the correct logical operation of the protocol.

We have considered image protocols that are formed by retaining a subset of the variables at each entity, and a subset of the fields in each message type. This restriction allows assertions about the image protocols to be easily related to the original protocols. Further, it supplies an iterative procedure to construct faithful image protocols.

We have shown that if the entity events satisfy certain nice structures, then faithful image protocols can be obtained directly and easily from the original protocol. In general, the more well-structured the program description of entity events, the easier it is to obtain faithful image protocols. We have applied these results to obtain faithful image protocols for a small example. For a larger example see [SHAN 82b].

APPENDIX

In the proofs below, for any element $s \in S_i$, s" denotes the subvector of s that corresponds to $\underline{v}_i$". s can be written as (s',s"). $\underline{S}_i$"denotes $\{s" : s \in S_i'\}$. S_i can be treated as $S_i' \times S_i"$

Proof of Observation 1

If a has $U(\underline{v}; \underline{v}) = U"(\underline{v};\underline{v}")$, then for every state event $(s,r,\alpha) \in [a]$, s'=r' since variables in $\underline{v}_i'$ are not affected. Hence a has a null image. If a has a null image, then in each state event $(s,r,\alpha) \in [a]$, s'=r' and no variables in $\underline{v}_i'$ need be assigned values. Hence $U(\underline{v};\underline{v}) = U"(\underline{v};\underline{v}")$.

End of Proof of Observation 1.

Proof of Observation 2

(i) Proof of (a).

The set of image state events of a are $[a]' = \{(s',r',\alpha) : s \in E$ and (s;r) is a transition of U$\}$. The set of state events of a' as specified in Observation 2 is $[a'] = \{(s',r',\alpha) : s' \in E'$ and (s';r') is a transition of U'$\}$. We need to show that $[a'] = [a]$. From the structure of E, if $s \in E$ then $s' \in E'$. From the structure of U, if (s;r) is a transition of U, then (s',r') is a transition of U'. Hence $[a]'$ $[a']$. for the reverse containment, choose any $t \in E$ (recall that by assumption E is not empty), and any $(s',r',\alpha) \in [a']$. t can be treated as (t',t"). (s',t") is an element of S_i. Further, (s',t") $\in E$ since $s' \in E'$ and $t" \in E"$ (since $t \in E$). From the structure of U, ((s',t"); (r',u")) is a transition of U for some $u" \in S_i"$ Hence $((s',t");(r',u"),\alpha) \in [a]$. Hence $[a']$ $[a]'$ and we have $[a'] = [a]'$.

(ii) Proof of (b).

If $\underline{F}(\underline{v}) = E'(\underline{v}_i')$, for any s whose images $s' \in E'$, we know that $s \in E$. This,, with the argument in (i), shows that any $(s',r',\alpha) \in [a']$ is strongly well-formed.

End of Proof of Observation 2.

Proof of Observation 3

If $(q;\underline{f}:\underline{F})$ has a null image, then each $(s,r,+(q,a)) \in [Rec\_q]$ has s'=r'. Hence the algorithm $U(\underline{v},\underline{f};\underline{v})$ need not assign values to $\underline{v}'_i$. Hence $U(\underline{v},\underline{f};\underline{v}) = U"(\underline{v},\underline{f};\underline{v}")$. If $U(\underline{v},\underline{f};\underline{v}) = U"(\underline{v},\underline{f};\underline{v}")$, then in each $(s,r,+(q,a)) \in [Rec\_q]$, s'=r' since $\underline{v}'_i$ is not affected. Hence $(q;\underline{f}:\underline{F})$ has a null image.

End of Proof of Observation 3.

Proof of Observation 4

(i) Proof of (a)

The set of image state events due to Rec_q_k is

$[Rec\_q_k]' = \{(s',s',+b) : s \in E_k\}$

$= \{(s',s',+\beta) : s' \in E'_k\}$

(because of structure of E_k and assumption that E_k is not empty)

The set of state events specified by Rec_β is

$[Rec\_\beta] = \{(s',s',+\beta) : s' \in E_k$ for any $1 \leq k \leq n\}$

Clearly, $[Rec\_\beta]$ is the union of $[Rec\_q_k]'$ over $1 \leq k \leq n$.

(ii) Proof of (b)

If $\underline{F}_k(\underline{v}) = \underline{F}'_k(\underline{v}'_i)$, for any s whose image $s' \in E'_k$, we know that $s \in E_k$. This and the argument in (i) shows that any $(s',s',+\beta)$ in $[Rec\_\beta]$ is strongly well-formed.

End of Proof of Observation 4.

Proof of Observation 5

(i) Proof of (c) and (d)

For any message $(q_k;a)$ of message type $(q_k;\underline{f}_k:\underline{F}_k)$, the set of state transitions in S'_j caused by its reception is

$[(q_k,a)]' = \{(s',r') : s \in E_k$ and $(s,a;r)$ is a transition of $U_k\}$

For any $a' \in \underline{F}'$, define

$[(q',a')] = \{(s',r') : s' \in E'$ and $(s',a';r')$ is a transition of $U'\}$

We will show that $[(q_k,a)]' = [(q',a')]$. Choose any $(s',r') \in [(q_k,a)]'$. Let (s',r') be obtained from transition (s,a;r) of U_k. From the structure of E_k, $s \in E_k$ means that $s' \in E'$. From the structure of U_k, (s',a';r) is a transition of U'. Hence $[(q_k,a)]'$ $[(q',a')]$. Consider any $(s',r') \in [(q',a')]$. Choose any $t \in E_k$ (E_k is not

empty by assumption). Then (s',t") ϵ E_k because s' ϵ E' and t" ϵ E_k''. From the structure of U_k, ((s',t"), a; (r',u")) is a transition of U_k, for any a ϵ a' and some u" ϵ S_j''. Hence [(q',a')] [$(q_k$,a)]. Thus, all messages (q_k,a) for any $1 \leq k \leq n$, and any a ϵ a' cause the same state transitions in S_j'. Hence they can be projected onto the same image (q',a'). Further, [Rec_q_k]' = [Rec_q'] since

$$[Rec\_q_k]' = \{(s',r';+(q',a')) : a \epsilon \underline{F}_k,\ (s',r') \epsilon [(q_k,a)]'\}$$

$$= \{(s',r';+(q',a')) : a \epsilon \underline{F}_k,\ (s',r') \epsilon [(q',a')]\}$$

$$= \{(s',r';+(q',a')) : a \epsilon \underline{F}',\ (s',r') \epsilon [(q',a')]\}$$

$$= [Rec\_q']$$

(ii) Proof of (e)

If $E_k(\underline{v}) = E_k(\underline{v}_i')$, then for any s whose image s' ϵ E', we know that s ϵ E_k for any k. This and the argument in (i) show that each (s',r',+(q',a')) ϵ [Rec_q'] is strongly well-formed.

End of Proof of Observation 5.

Proof of Observation 6

(i) Proof of (c)

The set of image send state events specified by Send_q_k is

$$[Send\_q_k]' = \{(s',r',-(q',a')) : s \epsilon E_k \text{ and } (s;a,r) \text{ is a transition of } U_k\}$$

$$= \{(s',r',-(q',a')) : s' \epsilon E_k' \text{ and } (s';a',r') \text{ is a transition of } U_k'\}$$

(The last step is due to the structure of E_k and U_k.)

The set of send state events specified by Send_q' in (c) is

$$[Send\_q'] = \{(s',r',-(q',a')) : \text{for } 1 \leq k \leq n,\ s' \epsilon E_k'$$

$$\text{and } (s';a',r') \text{ is a transition of } U_k'\}$$

Clearly, [Send_q'] = [Send_q_1]' U ... U [Send_q_n]'.

(ii) Proof of (d)

Consider $k_1,k_2,...,k_m$ where $E_{k_m}'(\underline{v}_i')$ and $U_{k_1}'(\underline{v}_i';\underline{f}',\underline{v}_i') = ... = U_{k_m}'(\underline{v}_i';\underline{f}',\underline{v}_i')$. Clearly, [Send_$q_{k_1}$]' = ... = [Send_$q_{k_m}$]'. Further, since $E_{k_1}''(\underline{v}")$ v ... v $E_{k_m}''(\underline{v}")$ = True, then for any s whose image s' ϵ E_{k_1}', we know that s ϵ E_{k_j} for some j. From (i) we know that (s,r,-(q_{k_j},a)) ϵ [Send_q_{k_k}] for some r ϵ r' and some a ϵ a'. Thus, each (s',r',-(q',a')) ϵ [Send_q'] is strongly well-formed.

Proof of Theorem

To prove this theorem, it is sufficient to show the following (refer to [LAM 82b]):

(1) For any two global states g and h of the original protocol system, if a time event can take the protocol system from g to h, then in the image protocol system either g'=h' or the same time event can take the image protocol from g' to h'.

(2) Given global states g' and h' of the image protocol system with well-formed time events, if an image time event can take g' to h', then in the original protocol system, then for any global state f in g such that f'=g', f is extendible to a path w such that w'=g',h'.

Proof of (1)

Suppose the time event in the original protocol is the local time event for time variable $v_{i,j}$ in $\underline{v}_i$. If $v_{i,j}$ is not retained in $\underline{v}_i'$ then g'=h'. Suppose $v_{i,j}$ is retained in $\underline{v}_i'$. Then its associated global time variable and reset value are also in $\underline{v}_i'$. Further, the time axiom $S_{i,j}$ for $v_{i,j}$ is also the same in the image protocol. Hence the local time event for $v_{i,j}$ in the image protocol is enabled, and its action takes the image protocol from g' to h'.

Suppose the time event in the original protocol is the global time event. Assume that g' $\neq$h'. The enabling condition of the global time event is a conjunction of 8a) conditions involving variables in $\underline{v}_i'$ and ages of messages in $\text{Channel}_i'$, and (b) conditions involving variables not in $\underline{v}_i'$ and ages of messages not in $\text{Channel}_i'$ (if any). Thus, if the global time event is enabled in the original protocol, then the image time event is enabled in the image protocol. Its occurrence will take the image protocol from g' to h'.

Proof of (2)

Suppose the time event in the image protocol system is the local time event for $v_{i,j}$ in $\underline{v}_i'$. Since the local time event for $v_{i,j}$ in the original protocol system is identically defined (in the same variables, accuracy axiom and time axiom), it is enabled at all states f ϵ g'. Its occurrence will take the original protocol system from f to a state h such that h'=g'.

Suppose the time event in the image protocol system is the global time event. If the global time event in the original protocol is enabled at a state f ϵ g', then its occurrence will take the original protocol system from f to a state h such that g'=g'. Now, suppose that the global time event in the original protocol is not enabled at a state f ϵ g'. This can happen only because some time variables in $\underline{v}_i$ that are not in $\underline{v}_i'$ would violate their time axioms and/or their accuracy axioms. If accuracy axioms are being violated, then the local time events of the concerned local time variables can occur, thereby releasing the global time event from these restrictions. If time axioms are being violated, then from the definition of

well-formedness, there is a sequence of null image internal events that will take the protocol system from f to a state where these time axioms are no longer violated. In either case, f can be extended to a path p such that p'=g' and the global time event can now occur. Its occurrence takes the protocol system from the last state in p to q such that q'=h'. Hence f can be extended to a path w such that w'=g',h'.

REFERENCES

[BOCH 77] Bochmann, G. V. and R. J. Chung.
A Formalized Specification of HDLC Classes of Procedures.
Conf. Rec. Nat. Telecommun. Conf. , December, 1977.
Los Angeles.

[BOCH 80] Bochmann, G. V. and P. Merlin.
On the Construction of Communication Protocols.
Proc. 5th ICCC , October, 1980.
Atlanta.

[BRAN 82] Brand, D. and W. H. Joyner.
Verification of HDLC.
IEEE Trans. on Commun. , May, 1982.

[DIVI 82] DiVito, B. L.
Verification of Communication Protocols and Abstract Process Models.
PhD thesis, Department of Computer Sciences, University of Texas at Austin, August, 1982.

[FLET 78] Fletcher, J. G. and R. W. Watson.
Mechanisms for a Reliable Timer-based Protocol.
Computer Networks, Vol. 2 :271-290, 1978.

[HAIL 80] Hailpern, B. T. and S. S. Owicki.
Verifying Network Protocols using Temporal Logic.
Technical Report 192, Computer Systems Laboratories, Stanford Univ., June, 1980.

[ISO 79] International Standards Organization.
Data Communications--HDLC Procedures--Elements of Procedures.
1979.
Ref. No. ISO 4335.

[KELL 76] Keller, R. M.
Formal Verification of Parallel Programs.
Comm. ACM , July, 1976.

[LAM 81] Lam, S. S. and A. U. Shankar.
Protocol Projections: A Method for Analyzing Communication Protocols.
Conf. Rec. National Telecommunications Conference , November, 1981.
New Orleans.

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

[LAM 82b] Lam, S. S. and A. U. Shankar.
Protocol Verification via Projections.
Technical Report 207, Dept. of Computer Sciences, University of Texas at Austin, August, 1982.

[SHAN 82a] Shankar, A. U. and S. S. Lam.
On Time-Dependent Communication Protocols and their Projections.
Proc. 2nd Int. Workshop on Protocol Specification, Testing and Verification , May, 1982.
Idyllwild, CA.

[SHAN 82b] Shankar, A. U. and S. S. Lam.
An HDLC Protocol and its Verification using Image Protocols.
Technical Report 212, Dept. of Computer Sciences, University of Texas at Austin, September, 1982.

[SHAN 82c] Shankar, A. U.
Analysis of Communication Protocols via Protocol Projections.
PhD thesis, Dept. of Electrical Engineering, Univ. of Texas at Austin, December, 1982.
(In preparation).

[SLOA 79] Sloan, L. J.
Limiting the Lifetime of Packets in Computer Networks.
Computer Networks, Vol. 3 :453-445, 1979.

[STEN 76] Stenning, N. V.
A Data Transfer Protocol.
Computer Networks , September, 1976.