PROTOCOL PROJECTIONS: A METHOD FOR ANALYZING COMMUNICATION PROTOCOLS

Simon S. Lam and A. Udaya Shankar
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

We propose the method of protocol projections to facilitate the analysis of protocols with several distinguishable functions. Image protocols are constructed separately for each protocol function. Image protocols are obtained by aggregating protocol entity states, messages and transitions using equivalence relations. As a result, image protocols are typically much simpler than the original protocol and can be analyzed more easily using available techniques. We have shown that if an image protocol satisfies a well-formed property, then it is faithful, in the sense that its logical correctness properties are the same as the logical correctness properties of the projected function in the original protocol. In this respect, the well-formed property can be regarded as a criterion for well-structured protocols.

1. INTRODUCTION

A communication protocol defines the set of rules that govern the exchange of messages between protocol entities. Such protocol entities are connected by communication channels (real or virtual). Keller [1] has described an abstract model for representing parallel computations. The model is a transition system specified by the pair (G,T) where G is the set of global states and T is a binary relation on G called the set of transitions. Models of communication protocols may be represented by such a transition system. The global state of a model of interacting protocol entities is specified by a joint description of the states of the protocol entities and communication channels. State transitions correspond to the occurrence of various events: an entity sending or receiving messages, errors in channels, an entity receiving signals from its user and timers, etc.

Given an initial state $e_0 \in T$, $T$ determines the reachability tree (space) $R$. $R$ is a directed graph with nodes being elements of $G$ and arcs being elements of $T$. $R$ contains all available information on logical correctness properties of the protocol. Let $R_S$ denote the set of reachable states in $G$. $R_S$, which may be obtained from $R$, determines the safety (partial correctness) properties of the protocol. Liveness properties, however, require knowledge of the set of finite paths in $R$.

Protocol Analysis Approaches

There are three basic approaches to protocol analysis [2]. They differ mainly in the kind of questions that the analyst poses in the characterization of $R$.

1) Reachability analysis

Starting from the initial state $e_0 \in G$, the reachability tree $R$ is traversed using $T$. Many logical errors (e.g., state deadlocks, unspecified receptions, etc.) that require only knowledge of individual reachable states in $R$ can be detected by automated procedures [3]. This approach has two difficulties. First, for any non-trivial protocol, $R$ is extremely large (possibly infinite). Second, many meaningful relationships among "state variables", expressing desirable logical correctness properties of the protocol, require an overall view of $R$ and these are usually not obvious from simply traversing $R$.

2) Proofs of invariant assertions

In this approach [4,5], one attempts to formulate invariant assertions describing desired logical correctness properties (corresponding, hopefully, to the service specifications of the protocol). Note that assertions are predicates on $G$. Hence an assertion can be viewed as a set $J$ of states in $G$ for which the predicate is true. An assertion of $A$ is invariant if the set $J$ is a superset of $R_S$. Invariant assertions of liveness properties can be thought of as specifying target sets in $G$ for trajectories of paths in $R$ to hit, eventually, recurrently, etc. In any case, the method of formulating invariant assertions, in effect, attempts to find "bounds" for the reachability tree $R$ instead of traversing it. These bounds are in the form of predicates which are statements about the state variables and their relationships.

3) Hybrid approach

Both of the above analysis approaches may be used together [1,6,7]. For many models, the state space $G$ can be written as a cross product of a set $G_c$ of control states and a set $G_d$ of data states, $G = G_c \times G_d$. A reachability analysis in $G_c$ yields

E3.2.1

CH1679-0/81/0000-0196 $00.75 © 1981 IEEE
the reachable set of control states. For each reachable control state, assertions may be formulated as predicates on $C_s$.

The difficulty with both the second and third approaches is that the formulation of assertions stating some desired logical correctness properties requires considerable human ingenuity, and cannot be easily automated. However, given some assertions (perhaps derived from protocol service specifications), their verification may be facilitated by various semi-automatic systems using theorem proving and symbolic execution techniques [7, 8].

Analysis of Nontrivial Protocols

The alternating bit protocol has been analyzed by many authors using a variety of techniques [6, 7, 9-12]. This is a relatively simple protocol whose reachability tree $R$ can be easily determined. The X.21 protocol has also been analyzed using automated systems for reachability analysis [13, 14].

The analysis of a relatively large protocol such as HDLC [15] is much more difficult. Reachability analysis typically cannot be employed due to the many state variables, some of which can take on a large number of values. Brand and Joyner [16] analyzed one version of HDLC using a system for symbolic execution. Altogether 1347 theorems had to be proved, 303 of which were proved automatically. The rest was done manually.

A protocol, such as HDLC, can be thought of as having many distinct functions. For instance, we can think of an HDLC protocol entity as consisting of five functional components such as shown in Fig. 1. Each component communicates with a corresponding component in the other protocol entity to accomplish a specific function (e.g., connection management, one-way data transfer, etc.).

**Fig. 1.** Functional components of HDLC.

For such complex protocols, an approach that appears attractive is to decompose each protocol entity into modules for handling the different functions of the protocol. Bothmann and Chung [17] used this approach to specify a version of HDLC. The main difficulty with protocol analysis using a decomposition approach is that the interaction among modules within each protocol entity is not structured. Modules interact through shared variables. They also interact because a message frame (packet) typically encodes data and control messages sent by different modules in one entity to the respective modules in the other entity. To model such dependencies, bothmann and Chung [17] proposed the use of coupled transitions between modules. This permits them to specify the HDLC protocol in a compact fashion, but does not seem to facilitate analysis of the protocol.

The data transfer protocol of Stemming has been successfully analyzed [4, 5]. Invariant assertions concerning both its safety and liveness properties have been proved. However, Stemming's protocol is a one-way data transfer protocol. It corresponds to the interaction of a Data Send module and a Data Receive module in isolation (see Fig. 1.). As such, it constitutes just one function of a real-life protocol such as HDLC. The following question arises: are the safety and liveness properties that are proved for the one-way data transfer protocol still valid when it is implemented as part of an overall protocol with the two types of dependencies mentioned above?

Protocol Analysis via Protocol Projections

Consider a protocol with several distinguishable functions. We would like to ask questions regarding the logical correctness behavior of the protocol concerning these functions. Instead of asking such questions all at the same time, we may ask them with respect to one function of the protocol at a time. The analysis approach is to construct from the given protocol an image protocol for each of the functions that are of interest to us. These will be referred to as the projected functions. The image entity states, messages, and transitions of an image protocol are obtained from those of the original protocol using certain equivalence relations. Entity states, messages, and transitions that are equivalent with respect to a projected function are aggregated in the image protocol. As a result, image protocols are simpler than the original protocol. Each image protocol can thus be more easily analyzed using available means. (An image protocol for the function of one-way data transfer of HDLC will be of the same complexity as Stemming's data transfer protocol and can thus be analyzed.) Thus the method of protocol projections is useful if image protocols are faithful, i.e., logical correctness properties of an image protocol are the same as logical correctness properties of the projected function in the original protocol.

A sufficient condition for an image protocol to be faithful is that the image protocol possesses a well-formed property. Although the well-formed property is a sufficient condition, we have found that it is the weakest condition that one can have without any knowledge of the reachability tree $R$ of the protocol. (Note that we cannot assume any knowledge of $R$ since its complexity is the basic source of our difficulties.) We have found that image protocols for a version of the HDLC protocol with respect to the functions of connection management and one-way data transfers
possess the well-formed property [18]. Thus the well-formed property is not a very stringent requirement. In fact, one can think of the well-formed property as a criterion of well-structured protocols.

The term "protocol projection" has been previously used by Bochmann and Merlin [19] to describe an operation in their method for protocol construction. Their basic idea of "projection onto the relevant actions" is similar to ours herein, but the development and application of the idea in their work and ours are different.

In Section 2 we shall first present our protocol model. In Section 3 we shall present definitions of image entity states, image messages and image transitions for a given projected function. In Section 4 the image protocol for a given function is defined. The well-formed property is introduced. Our main results are stated in two theorems. We shall illustrate the protocol model with a small protocol example. A small example is chosen for the sake of clarity, although the method of protocol projections is intended for the analysis of real-life protocols with multiple functions.

2. THE PROTOCOL MODEL

Our model is based upon the model of Brand and Zafirovolo [20]. Let \( P_1 \) and \( P_2 \) be two protocol entities that communicate with each other. \( P_1 \) sends messages to \( P_2 \) through channel \( C_1 \) and \( P_2 \) sends messages to \( P_1 \) through channel \( C_2 \). (See Fig. 2.) Messages transmitted through a channel may be corrupted by noise. The channels are modeled as FIFO queues with infinite storage capacities.

![Diagram](image)

Fig. 2. Components of the protocol model.

(Extension of results in this paper to models of channels that may duplicate and reorder messages and channels that have finite storage capacities will be presented in a forthcoming technical report and in [18].) For \( i = 1 \) and 2, we define the following:

- \( S_i \) is the set of states of entity \( P_i \),
- \( o_i \) is the initial state of \( P_i \) in \( S_i \), and
- \( M_i \) is the set of messages sent by \( P_i \).

To simplify our notation, we assume that

\[ S_1 \cap S_2 = \emptyset \quad \text{and} \quad M_1 \cap M_2 = \emptyset \]

where \( \emptyset \) denotes the null set.

Entity \( P_1 \) is initially in state \( o_1 \). A state transition of \( P_1 \) from state \( s \) to state \( r \) due to the occurrence of event \( e \) is denoted by the 3-tuple \( \langle s, r, e \rangle \). There are 3 types of events that cause state transitions to take place in \( S_i \):

1. \( < s, r, m \rangle \), where \( m \in M_i \), denotes a transition in \( S_i \) due to the sending of a message \( m \) by \( P_i \) into channel \( C_i \). The set of all such "send transitions" is a subset of \( S_i \times S_i \times M_i \).
2. \( < s, r, m \rangle \), where \( m \in M_i \), denotes a transition in \( S_i \) due to the reception of message \( m \) by \( P_i \) from channel \( C_i \). The set of such "receive transitions" is a subset of \( S_i \times S_i \times M_i \).
3. \( < s, r, a \rangle \), where \( a \) is a special symbol indicating the absence of a message, denotes a transition in \( S_i \) that does not involve a message. These transitions are called interface transitions because the state change is caused by events generated at the entity's local interfaces by its user and timers. The set of interface transitions is a subset of \( S_i \times S_i \times \{a\} \).

The three types of events for entity \( P_2 \) are similarly defined, but with the send transitions in \( S_2 \times S_i \times M_2 \), receive transitions in \( S_2 \times S_i \times M_i \), and interface transitions in \( S_2 \times S_i \times \{a\} \). Note that both send and receive transitions affect the states of channels as well as the entities involved. Interface transitions do not affect the states of channels. Let \( T_i \) denote the set of state transitions of \( P_i \) for \( i = 1 \) and 2.

The protocol between \( P_1 \) and \( P_2 \) is defined by specifying:

\( (S_1, S_2, o_1, o_2, M_1, M_2, T_1, T_2) \)

Let \( M_i \) denote a FIFO sequence of messages representing the state of channel \( C_i \) for \( i = 1 \) and 2. There are additional events that cause transitions in the states of the channels. First, each message in a channel may be corrupted by noise (an "error event"). We use the symbol \( n \) to represent the corrupted message in \( M_i \) following the occurrence of an error event. Second, if the first element of \( M_i \) is a corrupted message, it is received by the destination entity and discarded (an "error detection event"). We assume that both types of events do not affect the states of \( P_1 \) and \( P_2 \). Define

\[ M_i^1 = M_i \cup \{n\} \]

and

\[ M_i^k = M_i^{k-1} \times M_i^1 \quad \text{for} \quad k = 2, 3, 4, \ldots \]

The set of all possible message sequences in \( C_i \) is a subset of
\[ M_1 = \bigcup_{k=1}^{\infty} M_k \cup \emptyset. \]

where \( \emptyset \) denotes the empty set.

The state space of the global model of protocol interaction is

\[ G = S_1 M_1 \times M_2 \times S_2. \]

Each global state is a 4-tuple \( < s_1, m_1, m_2, s_2 > \)

where \( s_i \in S_i \) and \( m_i \in M_i \) for \( i = 1 \) and \( 2 \). The initial global state \( s_0 \) will be assumed to be \( c_0, 0, 0, 0 > \)

in the rest of this paper without any loss of generality.

We shall make the assumption that if multiple events in \( P_1, P_2, C_1 \) and \( C_2 \) occur simultaneously, then such an occurrence can be represented as a sequence of occurrences of events in some arbitrary order. This is called the arbitration condition [1]. In the present model, the arbitration condition is acceptable if mutual exclusion is enforced among event occurrences in the same protocol entities i.e. the state transitions of protocol entities are implemented as indivisible (atomic) operations.

Let \( \tau_1 \) denote the set of state transitions in \( G \) due to the 3 types of events in \( P_1 \). Let \( \tau_c \) denote the set of transitions in \( G \) due to the occurrence of error events and error detection events in the channels. Define \( T \) to be \( \tau_1 \cup \tau_c \).

The transition system \( (G, T) \) defines the global model of the interaction between \( P_1 \) and \( P_2 \). Recall that \( R \) denotes the reachability tree from the initial state \( s_0 = < c_0, 0, 0, 0 > \) in \( G \) using the binary relation \( T \). A path in \( R \) is defined to be an ordered sequence of states in \( G \) starting from \( s_0 \).

An example

We illustrate the above protocol model with an example of a symmetric full-duplex data transfer protocol to be described next. Since the protocol is full-duplex, it has two basic distinguishable functions: one-way data transfer from \( P_1 \) to \( P_2 \) and one-way data transfer from \( P_2 \) to \( P_1 \). For each direction, the protocol uses a very simple acknowledgement scheme. For simplicity, we have assumed that the channels used in this example are error-free. (In this case the protocol does not need timers).

Consider protocol entity \( P_1 \). \( P_1 \) has an infinite array of data blocks, SOURCE[i] for \( i = 0,1, 2, \ldots \), destined for \( P_2 \), and an infinite empty array, SINK[i] for \( i = 0,1,2, \ldots \), to store data blocks received from \( P_2 \). The state of \( P_1 \) is specified by the values of the 5-tuples

\[ <VS, D_{OUT}, VR, ACK\_DUE, BUSY> \]

where VS and VR are nonnegative integers while the others are boolean variables. Entity \( P_2 \) has a similar set of variables. For convenience, we have omitted qualifiers (1 or 2) for these variables and we shall omit them as long as it is clear whether we are referring to \( P_1 \) or \( P_2 \). The initial state of both entities is \( <0, false, 0, false, false> \).

The set \( M_1 \) of messages sent by \( P_1 \) is \{DATA1, ACK1, DATA1&ACK1\} where DATA1 is a data block, ACK1 is an acknowledgement and DATA1&ACK1 is a message encoding both DATA1 and ACK1. The set \( M_2 \) is \{DATA2, ACK2, DATA2&ACK2\}.

The set of events that cause state transitions in \( P_1 \) is presented in Table 1. Events 1-3 correspond to the sending of a message by \( P_1 \). Events 6-8 correspond to the reception of a message by \( P_1 \). Events 4 and 5 correspond to interface transitions caused by an external agent locally connected to \( P_1 \) (e.g., user, channel controller). The enabling condition of an event defines the set of entity states at which the event may take place. The action of each event causes the entity to enter a new state. Thus, each entry in Table 1 actually specifies a set of transitions. For example, the event SENDDATA defines the set of transitions \( <s, r, DATA1> : DATA1 \in M_1 \) s and r are 5-tuples in \( s_1 \) such that s satisfies \( D_{OUT} = BUSY = false \) and \( r \) is the same as \( s \) except that \( D_{OUT} = true \) and \( VS \) is incremented by 1.

In Table 1, the operation put(CHANNEL, DATA1&ACK) encodes DATA and ACK into a single message and appends it to the end of the sequence of messages in CHANNEL. The operation get(CHANNEL, DATA1&ACK) removes the message DATA1&ACK at the head of CHANNEL and decodes it into two messages DATA and ACK.

The transitions of \( P_2 \) are identical to those shown in Table 1 for \( P_1 \), except that the variables CHANNEL1, DATA1 and ACK1 and the variables CHANNEL2, DATA2 and ACK2 are interchanged.

The example protocol has two functions corresponding to data transfers in the two directions. The protocol is extremely simple but it embodies the two types of dependencies that one encounters when one attempts to decompose protocols into functional components. First, the variable BUSY is shared by both functions of the protocol. Second, the messages DATA1&ACK1 and DATA2&ACK2 are also shared. Such dependencies present major obstacles for protocol analysis using a decomposition approach. However, the method of protocol projections will be used to obtain a faithful image protocol for each function.

3. PROJECTIONS

Projections are achieved using equivalence relations. We shall denote the image of a protocol quantity \( x \) by \( x' \) which is obtained by aggregating all the protocol quantities that are equivalent. Images of protocol entity states, messages and global states are next defined and should be

E3.2.4
4. OUR IMAGE PROTOCOL

Our objective is to define an image protocol that is faithful, i.e., such that its logical protocol corrects the same errors as the logical protocol for which an image protocol is defined. The state space for the logical protocol is S = {S1, S2, ..., Sk}, where S1 is the initial state and Sk is the goal state. The image protocol operates on a subset of states, the image states, which are projected from the logical state space.

The image protocol starts by partitioning the state space S into subsets S1 and S2, where S1 contains states that can be reached from the initial state S1 and S2 contains states that cannot be reached from S1. The image states are then defined as the projections of the states in S1 and S2 to the image space.

The image protocol is defined as follows:

1. **Image Transition Function**: For any state S ∈ S, the image transition function T(S) is defined as the set of image states that can be reached from S in the logical protocol.

2. **Message Passing**: Messages are passed between image states according to the transition function, but only between states that are connected in the logical protocol.

3. **Message Storage**: Messages are stored in the image states as they are received, and can be retrieved when required.

4. **Message Reordering**: When messages are received in an order different from the order in which they were sent, they are reordered according to their delay in the logical protocol.

5. **Message Confirmation**: Each message is confirmed as received when the recipient state receives it.

The image protocol is simulated by the original protocol, but only on a subset of the states that are reachable from the initial state. This subset of states is determined by the image protocol, and it is the same subset for all protocols that are faithful to the original protocol.

Note that parallel image protocols are also defined, where multiple image protocols are executed in parallel. Parallel image protocols are used when the original protocol is parallelizable, i.e., when the protocol can be divided into independent subprotocols. The image protocol for parallel subprotocols can be defined as the union of the image protocols for each subprotocol.
Channel error axiom. If channel $C_t$ corrupts messages, then any message in $M_t$ may be corrupted.

Next consider $t' = (s', r', m') \in T'_1$ and some entity state $a \in a$. $r'$ is immediately $m'$-reachable from $a$ if either: (i) for $t'$ being an interface or send transition, there exists $(a, b, n) \in T_1$ for some $b \in r$ and some $n$ where $m'M_n$, or (ii) for $t'$ being a receive transition, there exists $(a, b, n) \in T_1$ for all $n$ such that $m'M_n'$. (In the protocol model of interest in this paper, send transitions in $T_1$ involving messages with null images can be regarded as internal interface transitions for the above definition.)

Definition. Partitions $S'_1$ and $S'_2$ of an image protocol are said to be well-formed if for any $(s', r', m') \in T'_1 \cup T'_2$ and for any $a \in a$, $r'$ is $m'$-reachable from $a$.

Definition. Partitions $S'_1$ and $S'_2$ of an image protocol are said to be strongly well-formed if for any $(s', r', m') \in T'_1 \cup T'_2$ and for any $a \in a$, $r'$ is immediately $m'$-reachable from $a$.

Obviously, an image protocol that is strongly well-formed must also be well-formed. In the above definitions, the sets $T_1$ contain both interface transitions and transitions involving messages. Also, $s'$ may be the same as $r'$ if $m'$ is the nonnull image of a message.

Given a global state $g$, we let $t(g)$ denote the global state that results upon executing an enabled transition $t$, where $t \in T$.

A path $w = g_0 \Rightarrow g_1 \Rightarrow \ldots \Rightarrow g_n$ in $R$ is extendable to a path $x = g_0 \Rightarrow g_1 \Rightarrow \ldots \Rightarrow g_n \Rightarrow g_{n+1} \Rightarrow \ldots \Rightarrow g_m$ in $R$ if there exist transitions $t_1, t_2, \ldots, t_m$ in $T$ such that $t_1 (g_{n+1}) = g_{n+1}$.

The extendability of paths is defined similarly in $R'$. Theorem 2. Given a well-formed image protocol, (i) for any path $w$ in $R$ and $u'$ in $R'$, if $w = uu'$ is extendable to $v'$ then $w$ is extendable to $x$ such that $x = xv'$; (ii) $R' = R'$.

A proof of the above theorem is given in [21]. Part (1) of Theorem 2 together with part (ii) in Theorem 1 imply that the liveness properties of a well-formed image protocol are the same as the liveness properties of the projected function in the original protocol. Since $R' = R'$ in part (ii) of Theorem 2, the safety properties are also the same. Hence a well-formed image protocol is faithful.

An image protocol of the example

Reconsider the full-duplex data transfer example introduced earlier in Section 2 and illustrated in Table 1. The protocol has two functions, namely, data transfer in the two directions between $P_1$ and $P_2$. We shall next present an image protocol of the function of data transfer from $P_1$ to $P_2$.

Observe that $VR$ and $ACK\_DUE$ in $P_1$, and $VS$ and $D\_OUT$ in $P_2$ are not needed for the $P_1$ to $P_2$ data transfer. We let $S'_1$ consist of all 3-tuples of the form $<VS, D\_OUT, BUSY>$ and $S'_2$ consist of all 3-tuples of the form $<VR, ACK\_DUE, BUSY>$. In other words, the image of $<VS, D\_OUT, VR, ACK\_DUE, BUSY>$ in $S'_1$ is $<VS, D\_OUT, BUSY>$ in $S'_1$.

Using the definition for images of messages in Section 3, we find that the image message sets are $M'_1 = \{DATA1\}'$ and $M'_2 = \{ACK2\}'$.

Consider $M'_1$. DATA1' is the image of DATA1 and DATA1\&ACK1 in $M_1$. ACK1 in $M_1$ has the null image and is not included in $M'_1$.

Using the definition for image transitions of protocol entities, the set of image transitions of $P_1$ is found and shown in Table 2, and the set of image transitions of $P_2$ is found and shown in Table 3.

The image protocol for the function of data transfer from $P_1$ to $P_2$ is relatively simple and the following invariant assertions of its logical behavior have been proved.

Invariant assertions
1. $\text{SINK}[i] = \text{SOURCE}[i]$ for $0 \leq i < VR$.
2. $\text{VS} \geq VR \geq \text{VS}-1$.
3. $\text{DATA1'}$ in CHANNEL1$\Rightarrow$ (D\_OUT)
   $\Rightarrow (\text{DATA1'} = \text{SOURCE}(\text{VS}-1))$
   $\Rightarrow (\text{exactly one DATA1 message in CHANNEL1})$
   $\Rightarrow (\text{not ACK\_DUE} \land (\text{VS} = \text{VR} + 1))$
   $\Rightarrow (\text{no ACK2 message in CHANNEL2})$.
4. $\text{ACK\_DUE} \Rightarrow (\text{D\_OUT})$
   $\Rightarrow (\text{no DATA1 message in CHANNEL1})$
   $\Rightarrow (\text{VS} = \text{VR})$ (no ACK2 message in CHANNEL2).
5. $\text{ACK2'}$ in CHANNEL2 $\Rightarrow$ (D\_OUT)
   $\Rightarrow (\text{DATA1 message in CHANNEL1}) 
   \Rightarrow (\text{not ACK\_DUE})$
   $\Rightarrow (\text{exactly one ACK2 message in CHANNEL2})$.
6. not D\_OUT $\Rightarrow$ VS $= VR$.

It can be shown using the definition of well-formed partitions that the image protocol defined above is strongly well-formed. Therefore, the above invariant assertions also describe the logical behavior of data transfer from $P_1$ to $P_2$ in the original protocol.

5. DISCUSSIONS

The method of protocol projections is intended to facilitate the analysis of protocols with several distinguishable functions. Image protocols are defined separately for each protocol function. We have shown that if an image protocol is well-formed,
then it is faithful, in the sense that its logical correctness properties are the same as the logical correctness properties of the projected function in the original protocol.

Unlike a decomposition approach, the method of protocol projections is not handicapped by dependencies that exist between different functional components of a protocol due to (1) the sharing of variables, and (2) the encoding of messages for the different functions into the same message frames (packets). Such dependencies are naturally accounted for in the definition of image protocols, since all the entity states and messages that are relevant to a function are included in its image protocol. On the other hand, since image protocols are obtained by the aggregation of equivalent states and messages, they are typically much simpler than the original protocol, and can be more easily analyzed by available verification techniques.

Note that the well-formed property is a sufficient condition for an image protocol to be faithful. However, a careful examination of the proof of Theorem 2 in [21] will show that this is the weakest sufficient condition that one can state without any knowledge of the reachability tree R. We have found that image protocols are well-formed for a version of the HDLC protocol with respect to the functions of connection management and one-way data transfers [18]. Thus, the well-formed property is not as stringent as it may appear.

One interpretation of a well-formed image protocol is that the aggregation of states in S₁ and S₂, to form S₁' and S₂', does not result in any loss of information for the projected function. If S₁' and S₂' were not well-formed, then we have made the error of considering two entity states s and r in S₁ as equivalent with respect to the projected function, though in fact they are not. In this respect, one can think of the well-formed property as a criterion of well-structured protocols. That is, a protocol would be considered well-structured, if to each of the basic protocol functions, we can define "maximal" partitions S₁' and S₂' to obtain image protocols.

We have extended the results in this paper to models of channels that may duplicate and reorder messages. We are also investigating issues of time variables, implementation variables and protocol synthesis, as well as the development of efficient algorithms for obtaining protocol images and checking the well-formed property [18].

REFERENCES


### TABLE 1. Transitions of Entity $P_1$ in the Protocol Example.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Enabling Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND_DATA</td>
<td>not BUSY and not D_OUT</td>
<td>DATA1 := SOURCE[V]; put(CHANNEL1, DATA1); VS := VS + 1; D_OUT := true</td>
</tr>
<tr>
<td>SEND_DATA_ACK</td>
<td>not BUSY and not D_OUT and ACK_DUE</td>
<td>DATA1 := SOURCE[V]; put(CHANNEL1, DATA1&amp;ACK2); VS := VS + 1; D_OUT := true; ACK_DUE := false</td>
</tr>
<tr>
<td>SEND_ACK</td>
<td>not_BUSY and ACK_DUE</td>
<td>put(CHANNEL1, ACK1); ACK_DUE := false</td>
</tr>
<tr>
<td>STOP_BUSY</td>
<td>not BUSY</td>
<td>BUSY := true</td>
</tr>
<tr>
<td>REC_DATA</td>
<td>first(CHANNEL2) = DATA2</td>
<td>get(CHANNEL2, DATA2); SINK[VR] := DATA2; VR := VR + 1; ACK_DUE := true</td>
</tr>
<tr>
<td>REC_DATA_ACK</td>
<td>first(CHANNEL2) = DATA2&amp;ACK2</td>
<td>get(CHannel2, DATA2&amp;ACK2); SINK[VR] := DATA2; VR := VR + 1; ACK_DUE := true; D_OUT := false</td>
</tr>
<tr>
<td>REC_ACK</td>
<td>first(CHANNEL2) = ACK2</td>
<td>get(CHannel2, ACK2); D_OUT := false</td>
</tr>
</tbody>
</table>

### TABLE 2. Transitions of $P_1$ in the Image Protocol.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Enabling Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND_DATA'</td>
<td>not BUSY and not D_OUT</td>
<td>DATA1 := SOURCE[V]; put(CHANNEL1, DATA1'); VS := VS + 1; D_OUT := true</td>
</tr>
<tr>
<td>START_BUSY</td>
<td>not BUSY</td>
<td>BUSY := true</td>
</tr>
<tr>
<td>STOP_BUSY</td>
<td>BUSY</td>
<td>BUSY := false</td>
</tr>
<tr>
<td>REC_ACK'</td>
<td>first(CHANNEL2) = ACK2'</td>
<td>get(CHannel2, ACK2'); D_OUT := false</td>
</tr>
</tbody>
</table>

### TABLE 3. Transitions of $P_2$ in the Image Protocol.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Enabling Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>REC_DATA'</td>
<td>first(CHANNEL1) = DATA1'</td>
<td>get(CHannel1, DATA1'); SINK[VR] := DATA1'; VR := VR + 1; ACK_DUE := true</td>
</tr>
<tr>
<td>START_BUSY</td>
<td>not BUSY</td>
<td>BUSY := true</td>
</tr>
<tr>
<td>STOP_BUSY</td>
<td>BUSY</td>
<td>BUSY := false</td>
</tr>
<tr>
<td>SEND_ACK'</td>
<td>not_BUSY and ACK_DUE</td>
<td>put(CHannel1, ACK2'); ACK_DUE := false</td>
</tr>
</tbody>
</table>