VERIFICATION OF COMMUNICATION PROTOCOLS VIA PROTOCOL PROJECTIONS*

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 communication protocols. In our model, protocol entities are connected by communication channels; messages sent by the protocol entities through the channels may be lost, duplicated and/or reordered. Protocols with several distinguishable functions are considered. Image protocols are constructed separately for each protocol function. Image protocols are obtained by aggregating protocol entity states, messages and events using equivalence relations. As a result, image protocols are typically much simpler than the original protocol and can be analyzed more easily using available techniques. We show that if an image protocol is constructed to satisfy 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.

1. INTRODUCTION

A communication protocol defines the set of rules that govern the exchange of messages between protocol entities connected by communication channels (real or virtual). A communication protocol can be viewed abstractly as a transition system specified by the pair (G,τ) where G is the set of global states and τ is a binary relation in G called the set of transitions [1]. The global state of such a model of interacting protocol entities is specified by a joint description of the states of the protocol entities and communication channels. State transitions correspond to the occurrence of various events: an entity sending or receiving messages, errors in channels, an entity receiving signals from its user and timers, etc.

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

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 g_0 in G, the reachability tree R is traversed using τ. 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 assertions

In this approach [4,5], one attempts to formulate assertions describing desired logical correctness properties (corresponding, hopefully, to the service specifications of the protocol). Note that assertions regarding safety properties are predicates on G. Hence a safety assertion can be viewed as a set J of states in G for which the predicate is true. An assertion of a safety property is invariant if the set J is a superset of R_s. Assertions of liveness properties are predicates on the paths in G. Liveness assertions in temporal logic [5], for example, specify target sets in G for trajectories of paths in R to hit, eventually, recurrently, etc. In any case, the method of formulating 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 the reachable set of control states. For

* This work was supported by National Science Foundation Grant No. ECS78-01803.

each reachable control state, assertions may be formulated as predicates on G_d.

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. Given some assertions (representing rigorous protocol service specifications), their verification may be facilitated by various semi-automatic systems using theorem proving and symbolic execution techniques [7,8].

A relatively large protocol such as HDLC [9] cannot be easily analyzed using a technique based upon any of the three basic approaches mentioned above. Reachability analysis typically cannot be employed due to the many state variables, some of which such as sequence numbers can take on a large number of values. Brand and Joyner [10] analyzed one version of HDLC using a system for symbolic execution. Altogether 1347 theorems had to be proved, 80% of which were proved automatically. The rest was done manually.

The protocol projection idea

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

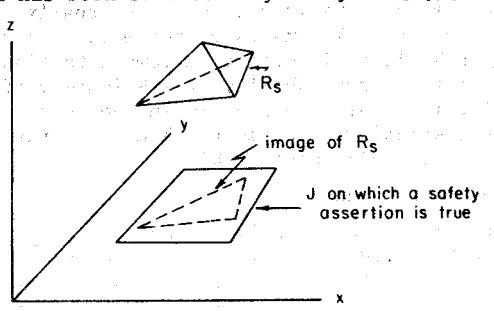

Fig. 1. An illustration of the protocol projection idea.

The protocol projection idea can be illustrated by the picture in Fig. 1. Consider a protocol model with the state description (x,y,z) and the set R_s of reachable states. Suppose that we are only interested in a safety assertion that involves only the variables x and y. To determine whether the assertion is true, it is sufficient for us to know the image of R_s on the (x,y) plane. Obviously, if R_s is known, its image on the (x,y) plane is readily available. However, the complexity of R (and thus R_s) is the basic source of difficulty in protocol analysis. Thus, we would like to take the original protocol and construct from it an image protocol whose set of reachable states duplicates the image of R_s on the (x,y) plane. In fact, we want to construct image protocols that are faithful with respect to both safety and liveness properties. In general, an image protocol is said to be faithful if its logical correctness properties are the same as the logical correctness properties of the projected function in the original protocol.

We show in this paper that if an image protocol is constructed to satisfy a well-formed property then it is faithful. Although the well-formed property is a sufficient condition, we have found that it is the weakest condition that one can have without any knowledge of the reachability tree R of the protocol. (Note again that we cannot assume any knowledge of R since its complexity is the basic source of our difficulties.) 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 [11]. 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.

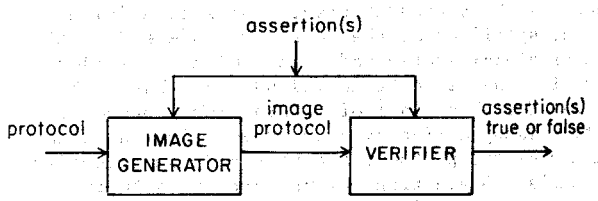

Fig. 2. Application of protocol projections.

The application of protocol projections to the analysis of protocols is illustrated in Fig. 2. Suppose that we are given a protocol and one or more assertions that specify the correct behavior of some protocol function (service specification of the function). Suppose also that a verifier is available for checking the validity of assertions for a given protocol. Instead of feeding the assertion(s) and the original protocol into the verifier, we first construct a well-formed image protocol (which should be much simpler than the original protocol). The image protocol and assertions are then fed into the verifier for evaluation. If we are interested in

several functions of the protocol, a different image protocol is generated for each function.

The protocol model and results in this paper extend the model and results previously presented by us in [12]. We consider herein communication channels that may lose, duplicate and reorder messages. Duplication and reordering of messages are not considered in [12].

The term "protocol projection" has been previously used by Bochmann and Merlin [13] 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 events 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 an extension of the model of Brand and Zafiropulo [14]. Let P_1 and P_2 be two protocol entities that communicate with each other. P_1 sands messages to P_2 through channel C_1, and P_2 sends messages to P_1 through channel C_2. (See Fig. 3.) Messages transmitted through a channel may be reordered, duplicated, and/or lost (due to corruption by noise).

A channel from one entity to another consists of all buffers and communication media between the entities. The channels are modeled as queues of infinite storage capacities. (Most communication protocols have some measure of flow control. As a result, their buffer requirements for messages in transit between entities are bounded. Hence, the assumption of infinite storage capacity is equivalent to satisfying those buffer requirements.)

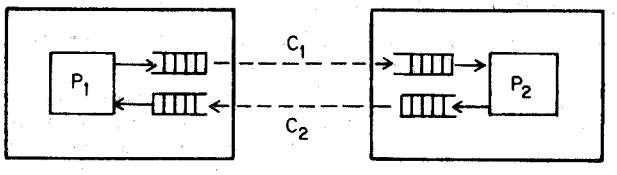

Fig. 3. Components of the protocol model.

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

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

$$\underline{M}_i = \left(\bigcup_{k=1}^{\infty} M_i^k \right) \cup \{\emptyset\}$$

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

The state space of the global model of protocol interaction is

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

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

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

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

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

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

(3) (s, r, α), where α is a special symbol indicating the absence of a message, denotes an internal event of P_1 that does not involve a message. This internal event is enabled when P_1 is in state s; after the event occurrence P_1 is in state r. Internal events model

events such as timeouts and interactions between the entity and its local user. The set of internal events is a subset of $S_1 \times S_1 \times \{\alpha\}$.

The three types of events for entity P_2 are similarly defined, but with the send events in $S_2 \times S_2 \times M_2$, receive events in $S_2 \times S_2 \times M_1$, and internal events in $S_2 \times S_2 \times \{\alpha\}$. Note that both send and receive events affect the states of channels as well as the entities involved. Internal entity events do not affect the states of channels. Let T_i denote the set of events of P_i for i = 1 and 2.

Events occurring in the communication channels will also cause transitions in the global state space (in addition to the entity events described above). These events are described below for channel C_1.

(1) $\xi_{1;k}$ denotes a loss event (e.g., due to noise) that deletes the k^{th} message in the channel. This is enabled if C_1 has at least k messages.

(2) $\delta_{1;k}$ denotes a duplication event that inserts a duplicate of the k^{th} message immediately after it in C_1. This event is enabled if C_1 has at least k messages.

(3) $\chi_{1;k,n}$ denotes a reordering event that takes the k^{th} message in C_1 and inserts it immediately after the n^{th} message. This event is enabled if C_1 has at least max(k,n) messages.

Note that these events do not affect the states of the entities.

Note that the enabling condition of each channel event does not depend upon the specific message occupying the k^{th} position. This implies that if channel C_i loses (duplicates, repositions) messages, then any message in M_i may be lost (duplicated, repositioned).

The three types of events for channel C_2 are similarly defined. Let E_i denote the set of events of C_i for i = 1 and 2. Define

$$E_i = \{\xi_{i;k}, \delta_{i;k}, \chi_{i;k,n} : k,n=1,2,\ldots\}.$$

Note that k and n range from 1 to ∞ in the definition of E_i, which implies that no position of the message sequence in C_i is exempt from an error event.

The above assumptions about the behavior of channels are summarized in the following axiom.

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

The protocol between P_1 is defined by specifying,

$$(S_1, S_2, o_1, o_2, M_1, M_2, T_1, T_2, E_1, E_2).$$

Recall that τ is the set of transitions in the global state space G. Define $T = T_1 \cup T_2 \cup E_1 \cup E_2$. Each event in T is enabled over a set of global states, and gives rise to a set of global state transitions. If the protocol is at global state g, and event t (in T) is enabled at g, then t can take the protocol to another global state, which we denote by t(g).

The transition system (G,τ) defines the global model of the interaction between P_1 and P_2. A path in G is defined to be a sequence of states $f_0 \rightarrow f_1 \rightarrow \ldots \rightarrow f_n$ in G, such that there exist events $t_1, t_2, \ldots, t_n$ in T with $t_i(f_{i-1}) = f_i$ for $1 \leq i \leq n$. Recall that R denotes the reachability tree from the initial state g_0. A path in R is a path in G that starts from the initial global state (i.e. $f_0 = g_0$).

We shall make the assumption that if multiple events in P_1, P_2, C_1 and C_2 occur concurrently, 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 events of protocol entities are implemented as indivisible (atomic) operations.

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, we have assumed that the channels used in this example are FIFO queues (i.e. the event sets E_1 and E_2 are null).

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

The state of P_1, at any time, is the value of the 6-tuple

<VS, D_OUT, VR, ACK_DUE, SINK, BUSY>.

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, Empty, False>.

The messages used in the protocol are of the following types: DATA, ACK and DATA&ACK. DATA ranges over the set of data blocks that can be sent in the protocol. ACK has a single value corresponding to a positive acknowledgement. A DATA&ACK type message contains an encoding of a DATA type message and a positive acknowledgement.

The set of events that cause state changes in P_1 is presented in Table 1. Event types 1-3 correspond to the sending of a message by P_1. Event types 6-8 correspond to the reception of a message by P_1. Event types 4 and 5 correspond to internal events caused by an agent locally connected to P_1 (e.g., user, channel controller). The enabling condition of an event type defines the entity states and channel states at which the events may take place. The action of each event causes the entity to enter a new state. DATA1 and DATA2 are messages of type DATA. DATA1&ACK1 and DATA2&ACK2 are messages of type DATA&ACK. ACK1 and ACK2 are messages of type ACK.

Note that each event type in Table 1 actually specifies a set of events in T as defined earlier. For example, the event type SEND_DATA defines the set of events {<s, r, DATA1> : DATA1 is of type DATA, s and r are 6-tuples in S_1 such that s satisfies D_OUT = BUSY = false and r is the same as s except that D_OUT is true and VS is incremented by 1}.

In Table 1, the operation put(CHANNEL1, DATA1&ACK1) encodes DATA1 and ACK1 into a single message and appends it to the end of the sequence of messages in CHANNEL1. The operation get(CHANNEL2, DATA2&ACK2) removes the message DATA2&ACK2 at the head of CHANNEL2 and decodes it into two messages DATA2 and ACK2.

The events of P_2 are identical to those shown in Table 1 for P_1, except that CHANNEL1 and the messages DATA1 and ACK1 are interchanged with CHANNEL2 and the messages DATA2 and ACK2.

The example protocol has two functions corresponding to data transfers in the two directions. The protocol is extremely simple but it embodies two types of dependencies that one encounters when one attempts to decompose a protocol into functional components. First, the variable BUSY is shared by both functions of the protocol. Second, the messages 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. The image of a protocol quantity is obtained by aggregating all those protocol quantities that are equivalent. Images of protocol entity states, messages, global states, events and paths are next defined and should be obtained in the order shown below.

Projection of entity states

We start by partitioning the state space S_i of protocol entity P_i for i = 1 and 2. Let S_i' denote a partition of S_i. S_i' is a collection of mutually exclusive and collectively exhaustive subsets of S_i. These subsets will be referred to as partition cells. The partition cells are defined so that entity states within a partition cell are equivalent with respect to the protocol function for which an image protocol is desired. The equivalence relation is determined by the semantics of the projected function. The image of a state $s \in S_i$ is defined as the partition cell in S_i' that contains s. We shall use the notation s' to denote both the image and the partition cell ($s \in s' \subseteq S_i$). We say that states s and r have the same image if and only if both s and r are in the same partition cell (they are equivalent with respect to the projected function). Furthermore, each partition cell in S_i' will also be referred to as an image state, and S_i' constitutes the image state space of P_i.

Projection of messages

The above equivalence relation is next extended to messages in M_1 and M_2. Two messages m and n in M_i are said to be equivalent if and only if they cause identical changes in both image state spaces S_1' and S_2'; in other words, m and n are equivalent if and only if for any pair of states (s',r') in S_1' and in S_2', m causes a change from some state in s' to some state in r' whenever n causes a change from some state in s' to some state in r'. This equivalence relation induces partitions of M_1 and M_2. Each set of equivalent messages in the partitions is said to be the image of individual messages in the set, and will also be referred to as an image message. Messages that cause only changes internal to image entity states in both S_1' and S_2' are said to have the null image. The image message sets are defined by

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

Images of channel states

The image $\underline{m}_i'$ of channel state $\underline{m}_i$ is obtained by taking the image of each element in $\underline{m}_i$ and

deleting the null images. (Implicit in this operation is the assumption that a message with a null image has no effect on the projected function. Note that this statement is not true when channels with finite storage capacities are considered.)

Images of global states

The above equivalence relations can now be used to define image global states. Two global states $g = \langle s_1, \underline{m}_1, \underline{m}_2, s_2 \rangle$ and $h = \langle r_1, \underline{n}_1, n_2, r_2 \rangle$ in G are equivalent ($g' = h'$) if and only if $s'_1 = r'_1$ and $\underline{m}'_1 = \underline{n}'_1$ for i = 1 and 2. The image global state space, denoted by G', is $\{g' : g \in G\}$.

Projection of entity events

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

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

Note that events in T_i associated with state changes from the set s' to the set r' and messages with the same non-null image are aggregated into a single image event in T'_i; internal events associated with state changes from s' to r' are aggregated into a single image internal event (s', r', α).

Note that internal events in T_i causing state changes within the same partition cell do not contribute to T'_1. Similarly, events in T_i involving messages with the null image also do not contribute to T'_i.

Image of a sequence of global states

Given w, a sequence of global states in G, its image w' is defined to be the sequence of image global states in w, where consecutive elements with the same image are aggregated into a single element (in w'). (Recall that a path in G is a sequence of global states.)

4. IMAGE PROTOCOL

Our objective is to define a new protocol that is faithful to a given projected function, i.e., such that its logical correctness properties are the same as the logical correctness properties of the projected function in the original protocol. We call this protocol an image protocol. A natural candidate is a protocol between entities P'_1 and P'_2, using channels C_1 and C_2. For i = 1 and 2: let S'_1 be the entity state space of P'_1; let o'_i be the initial state of P'_i, and let M'_i be the set of messages sent by P'_i. The global state space of the image protocol is G'. Let T'_i be the set of entity events for P'_i. Given the channel error axiom, the events of channel C_i remain the same as before.

The image protocol is completely specified by

$$(S'_1, S'_2, o'_1, o'_2, M'_1, M'_2, T'_1, T'_2, E_1, E_2)$$

Define $T' = T'_1 \cup T'_2 \cup E_1 \cup E_2$. As before, T' gives rise to a set of transitions τ' and G'. The transition system (G', τ') describes the interaction between the entities P'_1 and P'_2. Let R' be the reachability tree of the image protocol. Let R'_s be the set of reachable states ($R'_s \subseteq G'$)

Theorem 1. (i) the image of every path in R is a path in R'

(ii) $\{g' : g \in R_s\} \subseteq R'_s$.

A proof of the above theorem is given in [15] and is omitted here. We note that since invariant assertions of the safety properties of the image protocol correspond to supersets in G' that cover R'_s, the second result in Theorem 1 implies that such assertions are also valid in the original protocol. However, it is possible that assertions concerning the projected function are true in the original protocol but are not true in the image protocol. Also, we cannot draw any conclusions about the liveness properties of the projected function in the original protocol based upon the image protocol.

We present next sufficient conditions for the image protocol to be faithful to the projected function.

Definition. For i = 1 and 2, for $r' \in S'_i$, $a \in S_i$, and image message m', r' is immediately m'-reachable from a if:

(i) given that $m' \in M'_i \cup \{\alpha\}$, for some n whose image is m', there exists $(a,b,n) \in T_i$ for some $b \in r'$; or

(ii) given that $m' \in M'_j$ ($j \neq i$), for each n whose image is m', there exists $(a,b,n) \in T_i$ for some $b \in r'$.

Definition. For i = 1 and 2, for $r' \in S'_i$, $a \in S_i$, and image message m', r' is m'-reachable from a if:

(i) given that $m' \in M'_i \cup \{\alpha\}$, there is a sequence of internal events with state changes inside a' that will take P_i from a to another entity state c from which r' is immediately m'-reachable; or

(ii) given that $m' \in M'_j$ ($j \neq i$), for each n whose image is m', there is a sequence of internal events with state changes inside a' that will take P_i from a to some c such that there exists $(c,b,n) \in T_i$ for some $b \in r'$.

(In the protocol model of interest in this paper, send events in T_i involving messages with null images can be regarded as internal events for the above definition.) Note from the construction of T_1' and T_2', that for every $(s',r',m') \in T_i'$, there is a $(a,b,n) \in T_i$ such that $a' = s'$, $b' = r'$, and $n' = m'$.

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 $a \in s'$, 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 $a \in s'$, 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_i' contain both interface events and events involving messages. Also, s' may be the same as r' if m' is the nonnull image of a message.

A path $w = f_0 \rightarrow f_1 \rightarrow \ldots \rightarrow f_n$ in G is extendable to $x = f_0 \rightarrow \ldots \rightarrow f_n \rightarrow f_{n+1} \rightarrow \ldots \rightarrow f_{n+m}$, if x is also a path in G. The extendability of paths is defined similarly in G'.

Theorem 2. Given a well-formed image protocol,

(i) for any path w in R and u' in R', if w'=u' and u' is extendable to v' then w is extendable to x such that x'=v';

(ii) $\{g' : g \in R_s\} = R_s'$

A proof of the above theorem is given in [15]. Part (i) of Theorem 2 together with part (i) 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. Part (ii) of Theorem 2 implies that 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, ACK_DUE and SINK 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 4-tuples of the form <VR, ACK_DUE, SINK, BUSY>. In other words, the image of <VS, D_OUT, VR, ACK_DUE, SINK, 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 as follows: M_1' is the set of messages of type DATA, and M_2' consists of a single acknowledgement message of type ACK. In tables 2 and 3, 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'. ACK2' is the image of ACK2 and DATA2&ACK2 in M_2. DATA2 in M_2 has the null image and is not included in M_2'.

Using the definition for image events of protocol entities, the set of image events of P_1' is found and shown in Table 2, and the set of image events 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. SINK[i] = SOURCE[i] for $0 \leq i < VR$.
2. $VS \geq VR \geq VS-1$.
3. DATA1' in CHANNEL1 => (D_OUT)
   $\wedge$(DATA1' = SOURCE[VS-1])
   $\wedge$(exactly one DATA1' message in CHANNEL1)
   $\wedge$(not ACK_DUE) $\wedge$ (VS = VR + 1)
   $\wedge$(no ACK2' message in CHANNEL2).
4. ACK_DUE => (D_OUT)
   $\wedge$(no DATA1' message in CHANNEL1)
   $\wedge$(VS=VR) $\wedge$ (no ACK2' message in CHANNEL2)
5. ACK2' in CHANNEL2 => (D_OUT)
   $\wedge$(no DATA1' message in CHANNEL1) $\wedge$(VS=VR)
   $\wedge$(not ACK_DUE)
   $\wedge$(exactly one ACK2' message in CHANNEL2)
6. not D_OUT => 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 [15] 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 [11]. 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 S_1 and S_2, to form S_1' and S_2', does not result in any loss of information for the projected function. If S_1' and S_2' were not well-formed, then we have made the error of considering two entity states s and r in S_i 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 S_1', S_2' to obtain image protocols.

We are investigating issues of time variables, implementation variables, channels with finite storage capacities, and protocol synthesis, as well as the development of efficient algorithms for obtaining protocol images and checking the well-formed property [11]. The extension of the theory herein to protocols involving more than two protocol entities can be done in a straightforward manner.

REFERENCES

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

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

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

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

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

[6] Bochmann, G. V. and J. Gecsei, "A Unified Method for the Specification and Verification of Protocols," IFIP Congress 77, North Holland Publishing Company, 1977.

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

[8] Good, D. I. and R. M. Cohen, "Principles of Proving Concurrent Programs in Gypsy," Proc. 6th Annual ACM Symposium on Principles on Programming Languages, San Antonio, Jan. 1979.

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

[10] Brand, D. and W. H. Joyner, "Verification of HDLC," IBM Research Report RC7779, Yorktown Heights, New York, July 1979.

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

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

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

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

[15] Lam, S. S. and A. U. Shankar, "Protocol Verification via Protocol Projections, Part 1: Theory," Dept. of Computer Sciences, Univ. of Texas at Austin, Technical Report, 1981 (in preparation).

Event Type	Enabling Condition	Action
1. SEND_DATA	not BUSY and not D_OUT	DATA1 := SOURCE[VS]; put(CHANNEL1, DATA1); VS := VS + 1; D_OUT := true
2. SEND_DATA&ACK	not BUSY and not D_OUT and ACK_DUE	DATA1 := SOURCE[VS]; put(CHANNEL1, DATA1&ACK1); VS := VS + 1; D_OUT := true; ACK_DUE := false
3. SEND_ACK	not BUSY and ACK_DUE	put(CHANNEL1, ACK1); ACK_DUE := false
4. START_BUSY	not BUSY	BUSY := true
5. STOP_BUSY	BUSY	BUSY := false
6. REC_DATA	first(CHANNEL2) = DATA2	get(CHANNEL2, DATA2); SINK[VR] := DATA2; VR := VR + 1; ACK_DUE := true
7. REC_DATA&ACK	first(CHANNEL2) = DATA2&ACK2	get(CHANNEL2, DATA2&ACK2); SINK[VR] := DATA2; VR := VR + 1; ACK_DUE := true; D_OUT := false
8. REC_ACK	first(CHANNEL2) = ACK2	get(CHANNEL2, ACK2); D_OUT := false

TABLE 1. Events of Entity P_1 in the Protocol Example.

Event Type	Enabling Condition	Action
1. SEND_DATA'	not BUSY and not D_OUT	DATA1' := SOURCE[VS]; put(CHANNEL1, DATA1'); VS := VS + 1; D_OUT := true
2. START_BUSY	not BUSY	BUSY := true
3. STOP_BUSY	BUSY	BUSY := false
4. REC_ACK'	first(CHANNEL2) = ACK2'	get(CHANNEL2, ACK2'); D_OUT := false

TABLE 2. Events of P_1' in the Image Protocol.

Event Type	Enabling Condition	Action
1. REC_DATA'	first(CHANNEL1) = DATA1'	get(CHANNEL1, DATA1'); SINK[VR] := DATA1'; VR := VR + 1; ACK_DUE := true
2. START_BUSY	not BUSY	BUSY := true
3. STOP_BUSY	BUSY	BUSY := false
4. SEND_ACK'	not BUSY and ACK_DUE	put(CHANNEL1, ACK2'); ACK_DUE := false

TABLE 3. Events of P_2' in the Image Protocol