On the Construction of Multiphase Communication Protocols*

Gurdip Singh and Madhavi Sammeta
Department of Computing and Information Sciences
Kansas State University
Manhattan, KS 66506

Abstract

The complexity of communication protocols has led to compositional techniques to design and verify protocols. We propose a framework for sequential composition of protocols. We identify two types of interactions, ordering and inhibition, required to specify such compositions. By describing these interactions separately, our framework enhances the applicability to the technique to a larger class of protocols. The technique facilitates modular design and verification. We illustrate the use of our technique by deriving several protocols.

1 Introduction

Communication protocols are complex as they typically involve several activities which may happen concurrently at different places. As a result, several compositional techniques, which follow a "separation of concerns" paradigm, have been proposed for designing and analyzing protocols. These techniques typically involve the following three steps:

(1) Divide the functionality of a protocol into subfunctions,

(2) Develop protocols for the subfunctions, and

(3) Combine the protocols to obtain a protocol for the original problem.

A major advantage of such a technique is that properties of the composite protocol can be inferred from the properties of the protocols for the subfunctions (which are smaller in size and therefore easier to analyze).

The third step of combining the component protocols is the most difficult one and involves specifying interactions between the component protocols. Different techniques have used different types of interactions. Chow, Gouda and Lam [4] proposed a method for constructing multiphase protocols, one which go through multiple phases of behavior performing a distinct function in each phase. Various phases are designed and verified separately and then composed sequentially. For example, the IBM's BSC protocol for data link control, which performs the following three functions in sequence: connection setup, data transfer, and disconnect, was designed in [4]. A similar decomposition of a protocol for analyzing it was proposed in [3]. Lin and Tarng [8] proposed an improved technique which removes some of the limitations of the technique in [4] and expands the applicability of the technique. [7] presented a method to obtain protocols which performed multiple functions concurrently. This technique specifies operational constraints which impose constraints on the message sending steps of the component protocols. This technique may be viewed as parallel composition of protocols which allows interleaved execution of the component protocols (which is not allowed by sequential composition). A technique for parallel composition was presented in [10] which requires synchronization of actions which update shared variables or send/receive shared messages.

In this paper, we present a framework for sequential composition of protocols. We model a protocol as a pair of communicating finite state machines [1] [2]. To construct a multiphase protocol, we may have to order the phases and may have to specify phase transitions (*i.e.*, when to exit a phase). For example, in a data link control protocol, the data transfer phase must start after the connection setup phase completes, and

*This work was supported by NSF grant CCR-9211621.

0-8186-6685-4/94 $4.00 © 1994 IEEE

therefore, we have to specify an ordering constraint between these two phases. However, the disconnect phase may be initiated at any time (even before the data transfer phase has completed). After its initiation, all actions of the data transfer phase are inhibited. In the case, we have to specify that when the disconnect phase is initiated, a phase transition from the data transfer phase to the disconnect phase must occurs (no ordering relationship indicating that the data transfer phase should have completed exists).

We model these interactions required for constructing multiphase protocols using two types of constraints, *ordering* and *inhibition*, where the first constraint is used to enforce ordering between the phases while the inhibition constraint is used to specify phase transitions. We show that multiphase protocols can be obtained by either the ordering constraint or the inhibition constraint or by a combination of both. This specification of constraints allows us to generalize the existing sequential composition techniques. For example, the compositions in [8] correspond to the case which requires ordering between the first and the second phase and inhibition of the second phase when actions of the first phase occur. We find that certain protocols, for example those obtained by ordering alone, do not have to satisfy the conditions required by the techniques in [4] and [8]. This further increases the applicability of the technique. We illustrate the use of our technique by deriving several protocols such as the alternating bit protocol and a data link control protocol.

This framework is also a step towards unifying sequential and parallel composition. As discussed above, most techniques for parallel composition specify "synchronization" constraints on the actions of the component protocols. The specification of sequential composition in our framework has a similar flavor. We can obtain a framework in which the following constraints can be specified between actions of the component protocols: ordering, inhibition and "synchronization". This provides us with a methodology for series-parallel composition of protocols.

This paper is organized as follows. In the next section, we discuss our model of computation. Section 3 discusses the ordering and inhibition constraints. Section 3.1 discusses construction of a multiphase protocol. In Section 4, we give conditions for safety of the multiphase protocol and we conclude in Section 5.

2 Model

A distributed protocol is a set of processes. For simplicity, we will restrict ourselves to protocols involving two processes only. Each process is a tuple $< S, A, T, s_0 >$, where S is the set of states, A is the set of actions, T is the transition function $S * A \longrightarrow S$, and s_0 is the initial state. A process may be viewed as a directed labeled graph where S forms the set of nodes, T defines the edges and A defines the labels of the edges. Node s_0 is distinguished as the initial node. Figure 1 gives an example of a protocol (M, N).

An action is either (1) a sending action, represented by $-m$ (action of sending a message m), (2) a receiving action, represented by $+m$ (action of receiving a message m), or (3) an internal action. The corresponding edges in the graph are called *sending*, *receiving* and *internal* edges respectively. If there exists a transition from state u to v labeled a, we say that this transition is *incident from* u and *incident on* v. For example, in Figure 1, there exists an edge labeled $-g1$ incident from 0 and incident on 1. A *receiving path* is a sequence of receiving transitions $t_1, t_2, \ldots, t_n$, which form a directed path in the graph. A sending path is defined similarly. For example, $-g4, -g2$ forms a sending path in process N of Figure 1.

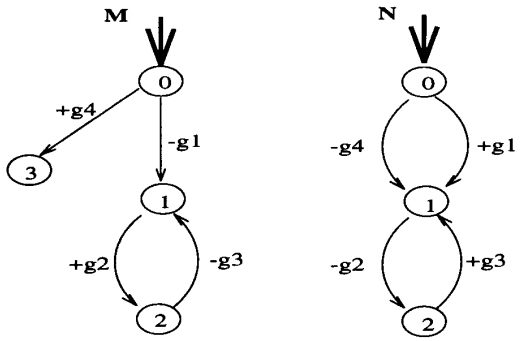

Figure 1: Example of a protocol.

The global state of the protocol (M, N) is presented as a tuple $< u, v, x, y >$, where u (v) is the state of M (N) and x (y) represents the strings of messages in transit to M (N). The initial global state is $< s_{m0}, s_{n0}, \lambda, \lambda >$, where s_{m0} and s_{n0} are the initial states of M and N respectively and λ denotes an empty sequence. The global state may change due to a transition in either M or N. If the global state

changes from S to S' due to a transition in either M or N, then we say that S' is a *successor* state of S. State S is reachable from S' if there exists a sequence of global states, $S_1 = S'$, S_2, ..., $S_n = S$, such that S_{i+1} is a successor of S_i. S is a *reachable* state if it is reachable from the initial state.

A state with no transition incident from it is a *final* state. In Figure 1, M has 3 as the final state while N has no final state. Following [8], a state pair (u, v) of (M, N) is *stable* if the following hold: (1) (u, v, λ, λ) is a reachable state and (2) if (u, v, x, y) is a reachable state then $x = y = \lambda$. For example, in Figure 1, $(1, 1)$ and $(3, 1)$ are stable.

Let $first(x)$ denote the first element in the sequence x. A reachable state $< u, v, x, y >$ is an *unspecified reception* state if one of the following holds: (1) $x \neq \lambda$ and there does not exist a transition labeled $+first(x)$ incident from u or (2) $y \neq \lambda$ and there does not exist a transition labeled $+first(y)$ incident from v. For example, in the protocol of Figure 1, $< 3, 2, g_2, \lambda >$ is an unspecified reception state since it is reachable via the path $< 0, 0, \lambda, \lambda > \longrightarrow < 0, 1, g_4, \lambda > \longrightarrow < 3, 1, \lambda, \lambda > \longrightarrow < 3, 2, g_2, \lambda >$ and there is no transition from 3 in M labeled $+g_2$. A protocol is *free from unspecified receptions* if there does not exist a reachable unspecified reception state.

A protocol is *deadlock free* if there does not exist a reachable state $< u, v, \lambda, \lambda >$ such that all transitions incident from u and v are receiving transitions, where u and v are not both final states. A protocol is safe if it is free from unspecified receptions and deadlock-free. The safety of a protocol can be verified using techniques such as reachability analysis [6] [9] [5].

3 A Framework for constructing multiphase protocols

In this section, we will present the composition principle. We will consider composition of two component protocols only (the principle can be extended to any number of component protocols). Let $p_1 = (M_1, N_1)$ and $p_2 = (M_2, N_2)$ be two protocols.

The main idea in sequential composition is to order actions of p_1 before those of p_2. We find that in several multiphase protocols, we need to order actions of p_2 only after a subset of actions of p_1 (as

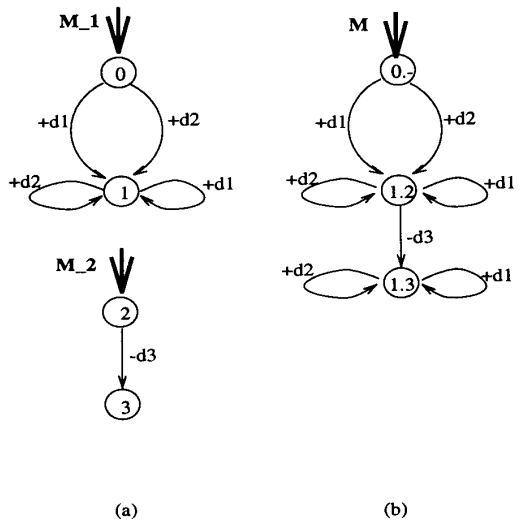

Figure 2: A Data Transfer Protocol.

pointed out in [8], if p_1 does not possess final nodes then it may not be possible to order all actions of p_1 before those of p_2). For example, consider the example shown in Figure 2. Let M_1 be a process in p_1 in which it may receive two data messages from N_1 containing the same information (maybe sent along different logical channels) *i.e.*, both $d1$ and $d2$ contain the same information. After receiving one of them, M_1 ignores the other message (N_1 is not shown). Let M_2 be a process in p_2, in which M_2 is required to propagate the information to another process N_2. In this case, we want to specify that M_2 can be initiated after M_1 has received the information, *i.e.*, when it reaches state 1. However, at this point, there may still be messages in transit to M_1 (either $d1$ or $d2$ could be in transit depending on which one is received first). Since we only want to order the phases, we require that after M_2 has been initiated, both M_1 and M_2 execute concurrently. In this case, as shown in Figure 2(b), the composite process M enters the combined state in which the first component indicates the state of M_1 and the second component indicates the state of M_2. All transitions incident on u and v are now incident on $u.v$. Thus, after M_2 is initiated, M_1 can execute the rest of its actions.

The ordering constraint is particularly useful when there exist state pairs (u, v) in p_1 which represent a logical connection point to initiate p_2 (*i.e.*, when it

reaches this state pair, it has performed the required actions to initiate M_2) but p_1 may still have messages in transit to be processed (*i.e.*, (u, v) is not stable but eventually the state $< u, v, \lambda, \lambda >$ is reached). In this case, after p_1 initiates p_2, it remains active and then becomes quiescent after taking care of its left-over messages.

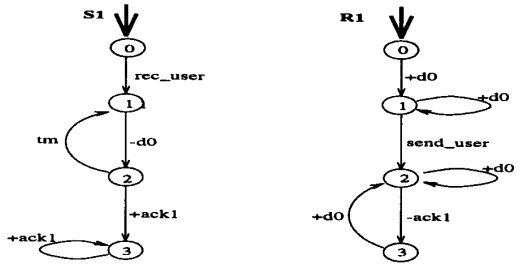

Figure 3: Data Transfer with bit 0.

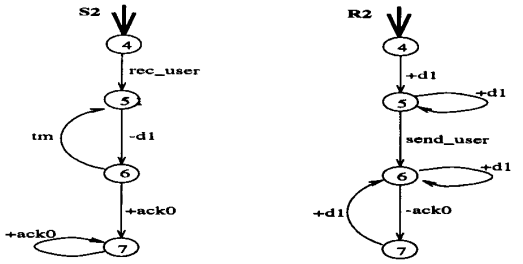

Figure 4: Data Transfer with bit 1.

As another example, consider the data transfer protocol, $AB0 = (S1, R1)$, of Figure 3, which sends a message with bit 0 from $S1$ to $R1$. In this protocol, we assume that timeouts may be premature and channels may lose messages. Therefore, $S1$ may send several $d0$ messages before any one of them is received. The action *tm* represents the timeout action, *rec_user* represents the action of fetching the next data item to be sent and *send_user* represents the action of depositing the next data item received. The protocol $AB1$ of Figure 4 is similar to $AB0$ (except that bit 1 is used instead of 0). In the alternating bit protocol, a message with bit 1 is sent only after an acknowledgement for the message with bit 0 has been received. This constraint translates into specifying that $S2$ can be initiated after $S1$ reaches state 3. We obtain a protocol, $AB01$, shown in Figure 5, by combining $AB0$ and $AB1$ using this ordering constraint. After $S1$ receives

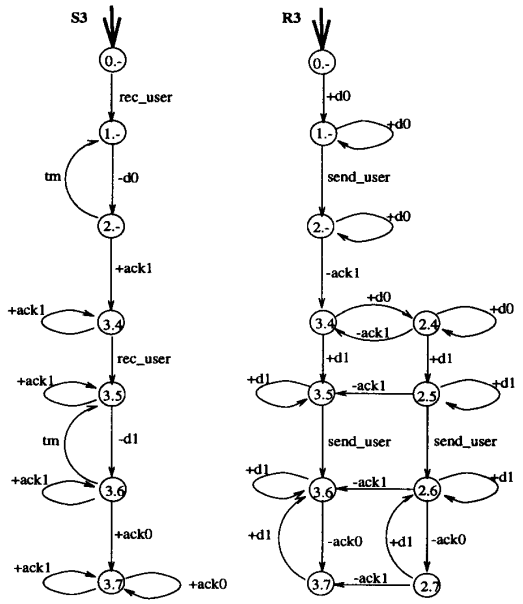

Figure 5: A Composite Data Transfer Protocol.

$+ack1$, it enters a combined state 3.4 in which both $S2$ and $S1$ can perform actions. This protocol performs both functions, transfers a message with bit 0 and a message with bit 1. However, it does not satisfy the conditions (given in Section 4) for combining a protocol with itself to obtain the alternating bit protocol. As discussed later, we have to specify some inhibition constraints to obtain the alternating bit protocol.

If only ordering is specified then both phases remain active. In some multiphase protocols, we may have to specify phase transitions points (*i.e.*, points at which one of the phases must be terminated and complete switch over to the other phase must be made). A second type of constraint, called *inhibition*, is needed for this purpose. For example, consider again the protocols of Figure 3 and Figure 4. After $R1$ reaches state 3, it repeatedly receives any $d0$ messages in transit and responds by sending $ack1$. In the alternating bit protocol, after $R2$ has received $d1$, no more $ack1$ messages are sent. We indicate this by specifying that the execution of transition $+d1$ inhibits $R1$. The protocol, $(S4, R4)$, obtained using this additional constraint is shown in Figure 6. In this case, on receiving $d1$, the first component of the state becomes undefined (indicating that no sending or internal actions of $R1$ can

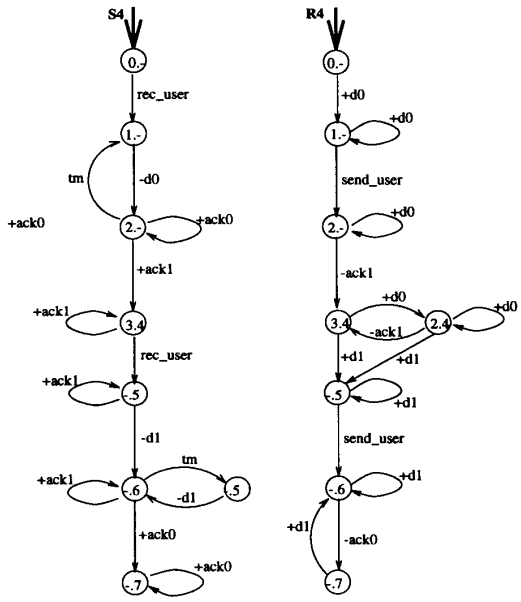

Figure 6: Data Transfer Protocol with inhibition.

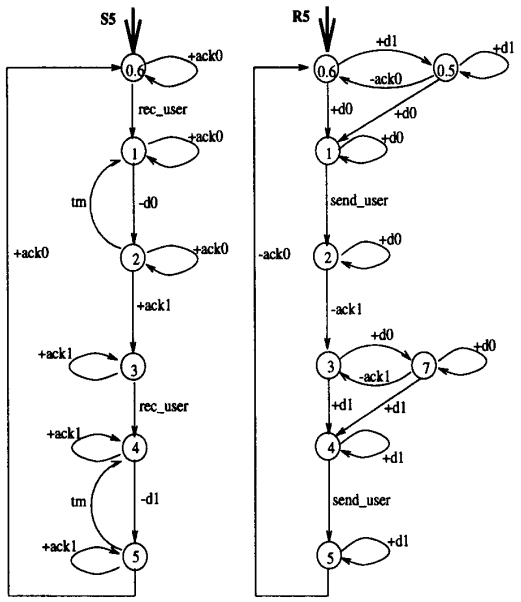

Figure 7: Alternating Bit Protocol.

be performed). For example, in state $-.5$, only transition incident on 5 in $S2$ may be performed. We may have to add additional receiving loops to ignore messages of the inhibited phase. Similarly, we specify that *rec_user* inhibits all actions of $S1$. By combining $(S4, R4)$ with itself, we obtain the alternating bit protocol, which is shown in Figure 7 (in the Figure, we have renumbered some of the states). In this case, ordering is specified between each iteration of the protocol and $+d0$ (*rec_user*) inhibits all actions of the previous iteration of $R4$ ($S4$). Some extra states and transition are present due to our construction procedure. These can be removed by adopting some step-wise refinement procedure. In these compositions, explicit phase transition points are specified using the inhibition constraint. This is typically required in protocols with no final states.

[8] derived the HDLC/NRM protocol which consists of two phases. The first phase establishes a connection and subsequently disconnects it. The second phase consists of a data transfer protocol. Figure 8 gives a simplified version of these protocols (no message loss is assumed). The message *req* is sent to request a connection while *disc* is sent to disconnect it. In this case, P_d can be initiated when P_c reaches state 2. However, we specify that action $-disc$ inhibits all actions of P_d while $+disc$ inhibits all actions of S_d.

We have given examples of compositions in which there exist ordering between p_1 and p_2 and actions of p_1 may inhibit p_2 or actions of p_2 may inhibit p_1. Inhibition may also be specified between phases which are not ordered. For example, if the disconnect protocol is modeled as a separate phase, then we may specify that $-disc$ inhibits the data transfer phase (no ordering is specified between the data transfer phase and the disconnect phase). As another example, in the two-phase commit problem, protocols for committing a transaction and aborting a transaction can be modeled as separate phases. In this case, an abort action will inhibit all actions of the commit phase.

Multiphase protocols may be obtained by ordering, inhibition or a combination of both. We find that both are independent constraints and should be specified as such. The ordering constraint specifies the connection points while the inhibition constraint specifies points at which phase transition must take place. The compositions defined in [8] correspond a particular combination of ordering and inhibition (the phases are ordered and the first phase inhibits the second phase).

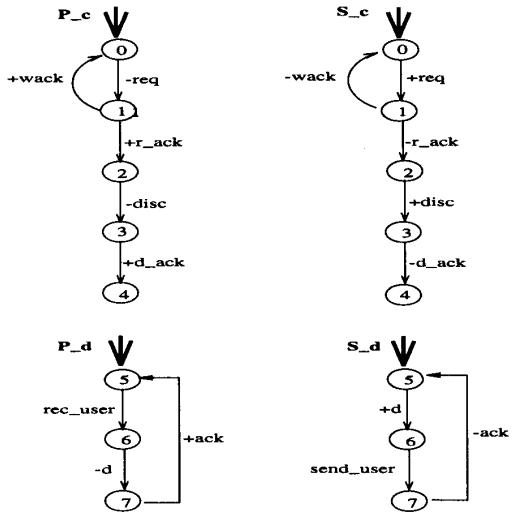

Figure 8: Connection Establishment and Data Transfer

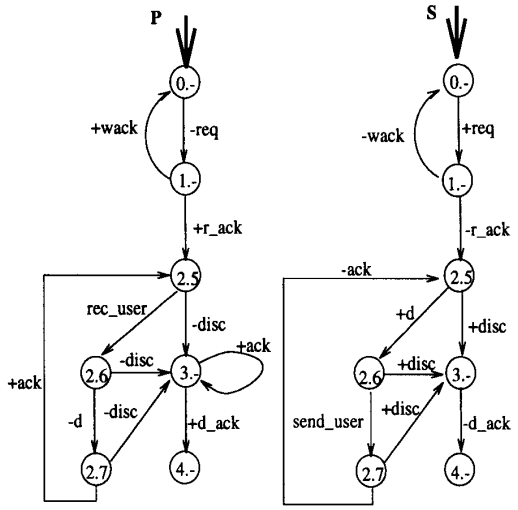

Figure 9: The Composite Protocol.

3.1 Constructing the multiphase protocol

Let $p_1 = (M_1, N_1)$ and $p_2 = (M_2, N_2)$. The ordering constraint is defined using a set $order(p_1, p_2)$ of tuples of the form (u, w), where u and w are states of M_1 and N_1 respectively. Each element of $order(p_1, p_2)$ is referred to as an exit pair. We say that u is a connection state of M_1 if there exists a pair (u, w) in $order(p_1, p_2)$. Let s_{m2} and s_{n2} denote the initial node of M_2 and N_2 respectively. Informally, if $(u, v) \in order(M, N)$ then M (N) enters the combined state $u.s_{m2}$ $(v.s_{n2})$ when M_1 (N_1) reaches state u (v). We specify the inhibition constraint using a set $inhibit(p_1, p_2)$ of actions. We require that it contain actions of either p_1 or p_2 and if $+m$ $(-m)$ is present in $inhibit(p_1, p_2)$ then $-m$ $(+m)$ is also present in $inhibit(p_1, p_2)$.

We denote a composite protocol $p = (M, N)$ as follows $< p_1, p_2,\ order(p_1, p_2),\ inhibit(p_1, p_2) >$. We will now gives the rules for obtaining M (rules of N are similar). Let s_{m1} and s_{m2} denote the initial states of M_1 and M_2 respectively. The initial state of M is $(s_{m1}.-)$ if $order(p_1, p_2)$ is not empty or s_{m1} is not a connection state. Otherwise, the initial state is $(s_{m1}.s_{m2})$. In the following, when we say add a node, we will mean the addition of a node if it is not already present. We assume that p_1 and p_2 do not have common action labels. The set of nodes and transitions of M are obtained as follows:

1. Let $(w.-)$ be a node in M, and a transition $w \longrightarrow u$ labeled a in M_1. If u is not a connection state, then add the node $(u.-)$, and a transition from $(w.-)$ to $(u.-)$ labeled a. This case corresponds to the case when M_1 is executing alone. If u is a connection state, then add the node $(u.s_{m2})$, and a transition from $(w.-)$ to $(u.s_{m2})$ labeled a.
2. Let $(-.v)$ be a node in M, and a transition $v \longrightarrow v'$ labeled a in M_2. Then, add the node $(-.v')$, and a transition from $(-.v)$ to $(-.v')$ labeled a. This corresponds to a transition in which M_2 is executing alone (after inhibiting M_1).
3. Let $(w.v)$ be a node in M, and a transition $w \longrightarrow w'$ labeled a in M_1. If a is present in $inhibit(p_1, p_2)$ then add the node $(w'.-)$ with transition labeled a from $(w.v)$ to $(w'.-)$. Thus, the execution of action a makes the state of M_2 undefined and therefore, no actions of M_2 are enabled from the resulting state. Otherwise, add node $(w'.v)$ with transition from $(w.v)$ to $(w'.v)$ labeled a.
4. Let $(u.v)$ be a node in M, and a transition $v \longrightarrow v'$ labeled a in M_2. If a is present in $inhibit(p_1, p_2)$ then add the node $(-.v')$, and a transition from (u, v) to $(-.v')$ labeled a. Otherwise, add node $(u.v')$, and a transition from $(u.v)$ to $(u.v')$ labeled a.
5. For each receiving transition labeled $+m$ in M_2, add a loop with label $+m$ to $(u'.-)$, where u' is any state in M_1.

6. For each receiving transition labeled $+m$ in M_1, add a loop with label $+m$ to $(-.v')$, where v' is any state in M_2.

The above procedure may add some additional receiving transitions which may never get executed. Several special cases can be identified to eliminate these transitions. For example, in rule 3, if a is a receiving transition, all edges incident on v are receiving edges, and for all exit pairs (u,t), if u has no reachable sending transition (this implies that all messages to N_1 are sent before any message to N_2) then the transition with label a may be omitted. We have used this rule to eliminate certain transitions in Figure 5 (*e.g.*, the transition labeled $+d0$ from 3.5 to 2.5). Similarly, some transitions added by rule 5 and rule 6 can be eliminated by making the rules more complicated (this elimination may require reachability analysis of p_1; [8] gives rules which eliminate extra transitions added by rule 5). For simplicity, we have presented the simpler versions here.

A protocol may also be combined with itself. If $p_1 = (M_1, N_1)$ is combined with itself, then the combined protocol is denoted by $(p_1^*, order(p_1), inhibit(p_1))$. The initial state in this case is obtained by fusing each connection state in $order(p_1)$ with the initial state. The same rules given above are followed for constructing the protocol.

We will now illustrate the technique by obtaining some protocols. We obtain $AB01$ of Figure 5 from $AB0$ and $AB1$ of Figure 3 and Figure 4 respectively, using $order(AB0, AB1) = \{(3,3)\}$, and $inhibit(AB0, AB1) = \{\}$. The protocol, $ABP01$, of Figure 6 is obtained by combining $AB0$ and $AB1$ using $order(AB0, AB1) = \{(3,3)\}$ and $inhibit(AB0, AB1) = \{-d1, +d1, rec\_user\}$. The alternating bit protocol is obtained by combining $ABP01$ with itself using $order(ABP01) = \{6,6\}$ (note that states have been renumbered) and $inhibit(ABP01) = \{rec\_user, -d0, +d0\}$. The protocol of Figure 9 is obtained using $\{(2,2)\}$ for ordering and $\{-disc, +disc\}$ for inhibition.

4 Safety of the multiphase protocol

In the following, we will discuss the sufficient conditions for safety of the multiphase protocol. We have kept the construction procedure simple. As a result, the safety conditions are more stringent. As pointed at some places, the construction procedure may be made more complex to weaken the safety conditions. We first define two safety conditions $C1$ and $C2$.

Condition $C1$: If $inhibit(p_1, p_2)$ consists of actions of p_1 then the following are true:

1. All actions incident from any exit state belong to $inhibit(p_1, p_2)$,
2. All exit pairs are stable, and
3. If M_2 initiates p_2 then if (u,v) is an exit pair and (w,v) is a stable state pair then (w,v) must also be an exit pair (this is the definition of closure in [8]). A similar condition must hold if N_2 initiates p_2.

These conditions are similar to the one described in [8]. Condition (1) can be relaxed by adding some additional restrictions on the exit pairs.

Condition $C2$: If $inhibit(p_1, p_2)$ does not consist of actions of p_1 then we do not require the exit pairs to be stable. For example, if only ordering is specified, and an exit pair (u,v) is not stable then $< u, v, x, y >$ may be a reachable state, where $x \neq \lambda$ or $y \neq \lambda$. In this case, after p_2 is initiated, p_1 remains active and can process messages in x and y. However, we do require the following: If M_2 initiates p_2 and (w, u, x, y) is a reachable state in p_1 such that w is a connection state, u is not a connection state and N_1 has not yet reached a connection state, then there exists an exit pair (w,v) such that v is reachable from u by consuming y', where y' is a subsequence of y. A similar condition must hold if N_2 initiates the protocol.

Note that checking conditions $C1$ and $C2$ requires reachability analysis of p_1 and p_2 (even though we may know that they are safe). The proofs of the following theorems are delegated to the full paper.

Theorem 4.1
$< p_1, p_2, order(p_1, p_2), inhibit(p_1, p_2) >$ is safe if (1) both p_1 and p_2 are safe and (2) conditions $C1$ and $C2$ are satisfied.

Theorem 4.2 *$< p_1^*, order(p_1), inhibit(p_1) >$ is safe if:*
(1) p_1 is safe,
(2) all actions enabled in the initial state are in $inhibit(p_1)$ (this implies that each iteration inhibits the previous one) and $C2$ is satisfied,
(3) If u is a connection state then it has no sending transitions incident from it, and

(4) if u is a connection state and $< u, v, x, y >$ is a reachable state in p_1 then the following must be true: If $+g$ is a transition reachable from the initial state in M_1 with no intervening receiving transitions then $g \notin x$. A similar condition must hold for N_1.

Condition 4 is required to avoid confusion between the messages of the current iteration with those of the previous iteration. The protocol of Figure 5 does not satisfy this condition and therefore, if we compose this protocol with itself, it will not result in a safe protocol.

5 Discussion and conclusion

We have presented a framework for sequential composition of protocols. We identified two types of constraints, ordering and inhibition, and showed that these are orthogonal to each other. We gave examples of multiphase protocols which require either ordering or inhibition or a combination of both. We gave sufficient conditions to ensure that the resulting composite protocol is safe given that the component protocols are safe.

There are several direction in which this framework can be extended. Firstly, the finite-state machine model is limited in its expressiveness and an extension for extended finite-state machine model is needed. Secondly, the constraints have been used in a restricted form to specify sequential composition. In general, we may specify ordering between arbitrary pair of states (some constraints will be required to ensure safety). For example, consider the case in which the protocol of Figure 2 is extended so that after receiving the message, M_1 sends an acknowledgement. Similarly, M_2 receives an acknowledgement for the message it sends. We may combine these protocols with the constraint that M_2 can send $d3$ after M_1 has received the information and that M_1 can send an acknowledgement only after M_2 has received an acknowledgement. Parallel composition techniques, such as those developed in [10], specify synchronization between the actions of the component protocols (if actions a and b are *synchronized* then a must wait until b is enabled and vice-versa). We can obtain a more general framework which allows the following three types of constraints between the actions of the component protocols: ordering, inhibition, and synchronization. This framework could be used to specify series-parallel compositions.

References

[1] Bochmann, G. Finite state description of communication protocols. *Computer Networks*, (2), 1978.

[2] Brand, D. and Zafiropulo, P. On communicating finite state machines. *Journal of the ACM*, 30(1), 1983.

[3] Choi, T.Y. and Miller, R.E. A decomposition method for the analysis and design of finite state protocols. In *Proceedings of the 8th Data Communication Symposium*, 1983.

[4] Chow, C., Gouda, M., and Lam, S. A discipline for constructing multi-phase communicating protocols. *ACM Transactions of Computer Systems*, 3(4), 1985.

[5] Gouda, M. and Han, J. Protocol validation by fair progress state exploration. *Computer Networks and ISDN Systems*, 9, 1985.

[6] Lin, F., Chu, P., and Liu, M.T. Protocol verification using reachability analysis: the staet space explosion problem and relief strategies. In *Proceedings of the ACM SIGCOMM Workshop*, 1987.

[7] Lin, H. A methodology for constructing communication protocols with multiple concurrent functions. *Distributed Computing*, 3, 1988.

[8] Lin, H. and Tarng, C. An improved method for constructing multiphase communications protocols. *IEEE Transactions on Computers*, 42(1), 1993.

[9] Maxemchuk, N. and Sabnani, K. Probabilistic verification of communication protocols. In *Proceedings of the IFIP International Workshop on Protocol Specification, testing and verification*, 1987.

[10] Singh, G. A compositional approach for designing protocols. In *Proceedings of the IEEE Int'l Conference on Network Protocols*, 1993.