A Compositional Approach for Designing Multifunction Time-Dependent Protocols *

Jun-Cheol Park Raymond E. Miller
Department of Computer Science
University of Maryland, College Park, MD 20742
{jcpark, miller}@cs.umd.edu

Abstract

We propose a framework based on the model of timed extended finite state machines for building communication protocols which perform several functions, where each function corresponds to a component protocol. For parallel composition, we specify a conjunctive relation which requires that the execution of events in two component protocols be synchronized. We also propose a predicate strengthening technique to refine the composite protocol in a stepwise manner while preserving the invariants of the component protocols. For sequential composition, we present a set of constraints, alternating, ordering and disabling, on the actions of the component protocols, and give sufficient conditions for the composite protocol to retain the safety properties such as freedom from unspecified receptions and freedom from deadlocks. Our sufficient conditions are weaker than those given in previous works.

1 Introduction

Communication protocols performing multiple functions consist of a large number of modules interacting with one another in a complex manner. The complexity of these protocols has led to the compositional design approach in which the protocols performing the various subfunctions are designed and verified separately, and then combined into a composite protocol in a disciplined manner. The major advantage of the approach is that some properties of the composite protocol can be inferred from those of the subfunction protocols which are usually smaller in size and thus much easier to analyze.

Several methods for protocol composition have been proposed, which can be divided into two categories: parallel composition [3, 5] and sequential composition [1, 4, 9, 10].

*This research was supported by NASA Grant No. NAG 5-2648 and NSF Grant No. NCR9506039.

A parallel composition technique allows an interleaved execution of the component protocols. A technique to obtain protocols which perform multiple functions concurrently was presented in [3]. This technique specifies operational constraints on the execution of the component protocols which are based on the extended finite state machine model. This technique, however, assumes that the component protocols do not share any variable or message. In [5], a technique for parallel composition of the component protocols was presented which allows the component protocols to interact with one another by synchronizing events which update shared variables or handle shared messages. It provides a "conjunction" technique to synchronize a pair of matching events in the component protocols and proves that all invariants of the component protocols are preserved in the composite protocol. However, each process in a protocol consists of a set of guarded actions at the same level, which makes it difficult to represent any complex or nested protocol structures.

In a sequential composition of protocols, the component protocols may execute in a sequential manner, such as ordering, alternating, disabling, at each site, but may not execute concurrently. In [1], a method was proposed for constructing multiphase protocols, which go through their phases one at a time with a distinct function performed in each phase. Each phase to perform a function is designed and verified separately, and then connected together to form the required protocol. A similar work presented in [2] applied the multiphase concept to decompose a protocol for simplifying analysis. An improvement of the technique in [1] was proposed in [7] which eliminates the need for final states used for connecting consecutive phases. The method proposed in [4] restricts the execution of the component protocols to an alternating order in which at most one of the component protocols can be active at any given time and the next function to be performed is chosen arbitrarily. In [9, 10],

0-8186-8061-X/97 $10.00 © 1997 IEEE

a framework for sequential composition of protocols was developed which allows component protocols to interact in a more complex and refined fashion than the ways discussed above. The constraints include enabling, inhibition and blocking. The disciplined use of the constraints as well as the specification of the constraints generalizes the existing sequential composition techniques.

For parallel composition, we specify the interactions using a semantic relation(correspondence) between the events of the component protocols. A conjunctive relation requires that the execution of events in two component protocols be delayed until both protocols are ready to execute the respective events. We also propose a predicate strengthening technique to refine the composite protocol in a stepwise manner while preserving the invariants of the component protocols.

For sequential composition, we present a set of constraints, alternating, ordering and disabling, to represent the execution structures between the component protocols. We discuss some sufficient conditions required for the composite protocol to retain the safety properties, such as freedom from unspecified receptions and freedom from deadlocks. The sufficient conditions are weaker than those given in [4, 7, 9, 10] in the sense that the conditions do not include any requirement that if one machine reaches a "transit" state, then the other machine will eventually reach the matching transit state while consuming the current input messages.

The organization of the paper is as follows. The next section is a review of the timed extended finite state machine model for protocol specification. In section 3, we present the parallel composition techniques for the construction of composite protocols and demonstrate its applicability by giving examples. In section 4, we give the sequential construction techniques for alternating, ordering and disabling functions, and discuss the conditions for the protocols to retain the safety properties. In section 5, we summarize our work and discuss some areas for future research. The proofs of lemmas and theorems are omitted due to the space limit and can be found in the full paper.

2 The model

2.1 Timed extended finite state machine

In the timed extended finite state machine(TEFSM) model, a distributed protocol is a set of processes, where each process is a timed extended finite state machine that can communicate with other processes via FIFO channels.

Definition 1 A TEFSM is specified as a tuple $< S, M, V, s_0, \delta >$, where S is a finite set of states, M is a finite set of messages that can be sent or received, V is a finite set of context variables, $s_0 \in S$ is the initial state, and $\delta : S \times E \rightarrow S$ is a partial state transition function, where E is a finite set of events, each of which can be classified as: (1) a receive transition of the form $P(V) \times RCV(M) \rightarrow A(V)$, where $P(V)$ is an enabling predicate on V which can be satisfied by the values of V at the state before the action, $RCV(M)$ is a set of blocking receptions of messages in M from the other machines, and $A(V)$ is the set of actions on V which can read the values contained in the local variables, write local variables, and/or send messages in M to the other machines, and (2) a non-receive transition of the form $P(V) \times T \rightarrow A(V)$, where T is a time interval $[l, u]$, $0 \leq l \leq u < \infty$, which specifies an upper and lower bound on the time that a transition may be enabled before occurring. If time limits are not specified, then default values of $[0, \infty)$ are presumed.

We assume that the rate of ticking between clocks in different machines is the same, which enables us to stipulate an imaginary global clock in the protocol system such that all of the relative times of the protocol processes appear to refer to it.

Definition 2 A channel C_{ij} is a FIFO queue connecting P_i to P_j. The contents of C_{ij}, denoted by c_{ij}, represents the FIFO queue of messages being sent from P_i to P_j, but not yet consumed by P_j. A bound D_{ij} is associated with each channel C_{ij} such that any message sent from P_i to P_j is delivered within the bound.

A non-receive transition is enabled if the predicate associated with it is true. A non-receive transition tr is *executable* when the following conditions are true: (1) the machine is in the head state of tr; (2) the predicate is true; and (3) the timing requirement is satisfied. The time interval is measured from the point at which the machine is in the head state of tr and predicate is true.

On the other hand, a receive transition is *executable* if it is *enabled*. A send action "send x from P_i to P_j" inserts $< x, at(x) >$ to the tail of C_{ij}, i.e., $c_{ij} \leftarrow c_{ij} \cdot < x, at(x) >$, where $at(x)$ is the time when the send action has occurred. A receive transition tr "receive x by P_j from P_i" is *potentially enabled* if $c_{ij} =< x, at(x) > \cdot c_{ij}$ and $at(x) < tick(head(tr)) < at(x) + D_{ij}$, where $tick(head(tr))$ is the time of the machine which has been waiting for the execution of tr at the head state of it. A potentially enabled receive transition may be delayed for at most $at(x) + D_{ij} - tick(head(tr))$ time units before becoming an *enabled* receive transition.

The execution of any transition is spontaneous in the sense that both the state change and the action associated with the transition occur simultaneously and take no time to complete.

2.2 Scheduling

In our protocol model, there may be more than one enabled transition, in which case choices can be made arbitrarily as long as there exists no enabled transition that has been enabled for longer than its timeout value without being executed. For this purpose, each protocol entity executing its process keeps the elapsed time, called *elapsedtime*, since the protocol entity visited the current state of the process.

In a composite protocol from more than one component protocol, furthermore, we allow a protocol entity to execute more than one transition at a time. This proposition follows from our interpretation of the time interval associated with each transition, i.e., every transition, say tr, that has been enabled for its timeout value must occur immediately unless an enabled transition other than tr from the same component protocol occurs when the timeout of tr expires.

2.3 Definitions and notations

For simplicity, we will discuss the composition techniques for two process protocols only throughout this paper. The techniques can be extended to more than the two process case in a natural way for the parallel composition and with some minor modification in the recovery scheme for the sequential composition.

In the rest of the paper, we use a pair (P_1, P_2) to denote a protocol consisting of TEFSMs P_1 and P_2 that can communicate with each other via FIFO channels.

The *system state* of (P_1, P_2) is a tuple $< v, w, x, y >$, where v and w are the current states of P_1 and P_2, respectively, and x and y are the FIFO message sequences being transmitted to P_1 and P_2, respectively. The *global state* of (P_1, P_2) consists of the system state and the values of the context variables at the system state. A global state s of (P_1, P_2) is a *deadlock state* iff (1) $x = y = \epsilon$, and (2) no transition in either P_1 or P_2 is executable in s, where $< v, w, x, y >$ is the system state of s. A global state s of (P_1, P_2) is an *unspecified reception state* iff either (1) $x \neq \epsilon$ and no receive transition at v is executable, or (2) $y \neq \epsilon$ and no receive transition at w is executable, where $< v, w, x, y >$ is the system state of s. A protocol (P_1, P_2) is *safe* iff any reachable state in the protocol is neither a deadlock state nor an unspecified reception state. The safety of a protocol can be verified using timed reachability analysis [8, 6].

A *timed sequence* σ in a TEFSM M is a finite or infinite sequence of pairs $< e_i, t_i >$, $i = 1, 2, \ldots, n(\text{or } \infty)$, where $t_i \leq t_{i+1}$ and each pair $< e_i, t_i >$ denotes that a transition e_i of M has occurred when the time is equal to t_i. A timed sequence σ in a TEFSM M is *valid* if each transition e_i has been enabled at $head(e_i)$ for w time units before occurring, where $w \in [l, u](e_i)$.

Notation: (1) Given a finite sequence σ, $first(\sigma)$ and $last(\sigma)$ denote the first and last element of σ, respectively. ϵ denotes an empty sequence, $|\epsilon| = 0$. (2) Given a sequence of transitions σ, we denote $\sigma \downarrow_P$ for the projection of σ onto the transitions of P. (3) Given a transition tr, $head(tr)$ and $tail(tr)$ are respectively the head and tail state of tr. (4) Given a transition tr, $cond(tr)$, $action(tr)$, and $[l, u](tr)$ are respectively the condition, action and time interval associated with tr.

3 Parallel composition

In this section, we combine the component protocols by specifying a set of constraints on their execution to achieve a composite protocol whose execution is an interleaved execution of the component protocols subject to the constraints. The constraints are local in that they are applied to the processes at the same site.

We identify a conjunction constraint which is also studied in [5] for synchronizing a pair of events from the component protocols. As a non-synchronizing parallel composition, we also present a predicate strengthening technique which refines the execution of the composite protocol while retaining the safety properties of the component protocols.

We start with an algorithm to generate the cross product of two component protocols, whose execution would be an unconstrained interleaving of those of the component protocols provided that the component protocols do not share any variable or message.

Definition 3 Let X be a composite protocol of the component protocols P and Q. A transition tr in X is $P(Q$, resp.$)$-*inherited* if tr is included in X because it is in $P(Q$, resp.$)$.

Algorithm $P_1|||Q_1$
Input: $P =< S_{p1}, M_p, V_{p1}, s_{p1}, \delta_{p1} >$ and $Q =< S_{q1}, M_q, V_{q1}, s_{q1}, \delta_{q1} >$
Output: $P_1|||Q_1$
Auxiliary Variables: $t_p = t_q = 0$, initially. At each execution of a $Q(P$, resp.$)$-inherited transition, $t_p(t_q$, resp.$)$ is updated to incorporate the time taken for the execution so that, from the $P(Q$, resp.$)$'s point of view, the time $t_p(t_q$, resp.$)$ for the execution of a series of $Q(P$, resp.$)$-inherited transitions, if any, can be regarded as the delay taken for $P(Q$, resp.$)$-inherited transitions.

1. (s_{p1}, s_{q1}) is the initial state of $P_1|||Q_1$.
2. **repeat** the following **until** no new state or transition can be added to $P_1|||Q_1$ <**for** each state (u, v)

in $P_1|||Q_1$ **do**: [**if** $u \rightarrow w$ is a transition labeled a in P_1 **then** add the state (w, v) and a transition $(u, v) \rightarrow (w, v)$ labeled a into $P_1|||Q_1$; **if** $v \rightarrow x$ is a transition labeled b in Q_1 **then** add the state (u, x) and a transition $(u, v) \rightarrow (u, x)$ labeled b into $P_1|||Q_1$;]>

3. **for** each non-receive transition tr in $P_1|||Q_1$ **do**: <**if** tr is P-inherited **then** $[[l, u](tr) \leftarrow [max(0, l - t_p), u - t_p]$; $action(tr) \leftarrow [action(tr); t_p \leftarrow 0; t_q \leftarrow t_q + elapsedtime]]$; **if** tr is Q-inherited **then** $[[l, u](tr) \leftarrow [max(0, l - t_q), u - t_q]$; $action(tr) \leftarrow [action; t_q \leftarrow 0; t_p \leftarrow t_p + elapsedtime]]$ >

Lemma 1 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$, where P and Q do not share any variable or message, be the component protocols of a composite protocol $C = (P_1|||Q_1, P_2|||Q_2)$. If σ is a valid sequence in C, then $\sigma \downarrow_P (\sigma \downarrow_Q$, resp.) is a valid sequence in $P(Q$, resp.).

Corollary 1 Let A be a safety property in $P(Q$, resp.) which refers only to variables in $P(Q$, resp.). Then A holds in $P|||Q$.

Lemma 2 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$, where P and Q do not share any variable or message, be the component protocols of a composite protocol $C = (P_1|||Q_1, P_2|||Q_2)$. If P and Q are safe, then C is also safe.

The same problem was studied in [3], but the sufficient condition for deadlock freedom in [3] is more complex than ours to rule out the possibility of a (combined) message transmission/reception being blocked due to the message combining technique adopted in [3] and the wrong-chosen operational constraints of the composite protocol.

3.1 Synchronizing events

This section discusses the construction of a composite protocol when the component protocols may share messages and update common variables. In the following, we discuss a conjunctive constraint with which the composite protocol can retain the safety properties of the component protocols.

Algorithm $P|(tr_p, tr_q)|Q$
Input: $P =< S_p, M_p, V_p, s_p, \delta_p >$, $Q =< S_q, M_q, V_q, s_q, \delta_q >$ and a pair of transitions (tr_p, tr_q) to be synchronized, where $[l, u](tr_p) = [l, u](tr_q) = [0, \infty)$.
Output: $P|(tr_p, tr_q)|Q$
Auxiliary Variables: $t_p = t_q = 0$, initially.

1. $P|(tr_p, tr_q)|Q \leftarrow P|||Q$.

2. Let $action(tr_c)$ be the common part, if any, of $action(tr_p)$ and $action(tr_q)$. $action(tr_{p'})$ and $action(tr_{q'})$ are the remaining action parts, respectively, such that $action(tr_p) = [action(tr_c); action(tr_{p'})]$ and $action(tr_q) = [action(tr_c); action(tr_{q'})]$.[1]

3. Add a transition tr_{pq} from $(head(tr_p), head(tr_q))$ to $(tail(tr_p), tail(tr_q))$, where tr_{pq} has the condition $[cond(tr_p) \wedge cond(tr_q)]$, and action $[action(tr_c); action(tr_{p'}); action(tr_{q'}); t_p \leftarrow t_q \leftarrow 0;]$, respectively.

4. Remove the transitions labeled tr_p or tr_q, and isolated states, if generated, from $P|(tr_p, tr_q)|Q$.

It is natural that for a synchronizing pair (tr_p, tr_q) of events, any transition incident from either $head(tr_p)$ or $head(tr_q)$ other than tr_p or tr_q, if any, should be subject to the synchronization, too. Hence, we impose the following restriction R_1: Let $sync(P, Q)$ be the set of transition pairs which need to be synchronized with each other for their execution. For each $(tr_p, tr_q) \in sync(P, Q)$, if there exists a transition tr(or tr') in P(or Q) such that $head(tr) = head(tr_p)$(or $head(tr') = head(tr_q)$), then (tr, tr_q)(or (tr_p, tr')) $\in sync(P, Q)$.

Now, the composite protocol $P|sync(P, Q)|Q$ is: $P|sync(P, Q)|Q \leftarrow [\cap_{(tr_p, tr_q) \in sync(P,Q)} P|(tr_p, tr_q)|Q] \cup [\cup_{(tr_p, tr_q) \in sync(P,Q)} \{tr_{pq}\}]$, where we use the operator $\cap$ to extract the common transitions of the machines and $\cup$ to unite the corresponding transitions.

We also need another restriction R_2 as follows: For any pair $(tr_p, tr_q) \in sync(P, Q)$, $[l, u](tr_p) = [l, u](tr_q) = [0, \infty)$. The restriction R_2 avoids the possibility that either tr_p or tr_q, $(tr_p, tr_q) \in sync(P, Q)$, is enabled and must be executed by its timing requirement before the other transition is ready to execute.

We illustrate the applicability of our technique by designing a data transfer protocol. The protocol is the same as the one given in [5] except that our protocol is modeled by TEFSMs. We start with two protocols $DATA(i, j)$ and $DATA(i, k)$ to form a composite protocol $DATA(i, j, k)$. $DATA(i, j)$ is a data transfer protocol with i as the sender and j as the receiver. Process i has a sequence of data $sd[1..max]$ to be sent to j and process j stores the received data in $rd[1..max]$. The protocol $DATA(i, k)$ is identical to $DATA(i, j)$ except that k instead of j is the receiver. Figure 1 shows $DATA(i, j)$. Initially, $snt =$

[1] It is assumed that the execution order of subactions in $action(tr_p)$(or $action(tr_q)$) is immaterial as far as the global states of the protocol are concerned. For example, both $a; b; c$ and $c; b; a$ result in the same global state when $action(tr_p) = a; b; c$.

$rcvd = 0$, $ack\_rcvd_{ij} = true$ and $c_{ij} = c_{ji} = \epsilon$. Now, we combine $DATA(i,j)$ and $DATA(i,k)$ to obtain $DATA(i,j,k)$ in which i sends the same data items to both j and k. Note that $DATA(i,j,k)_i = DATA(i,j)_i | \{(Send_{ij}, Send_{ik})\} | DATA(i,k)_i$.

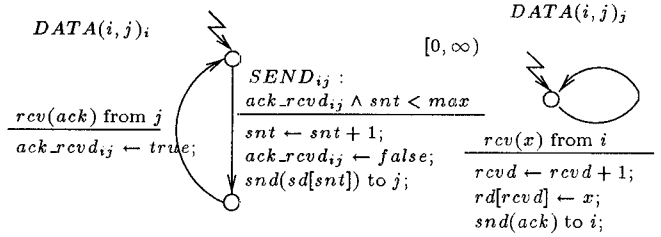

Figure 1: Data Transfer Protocol

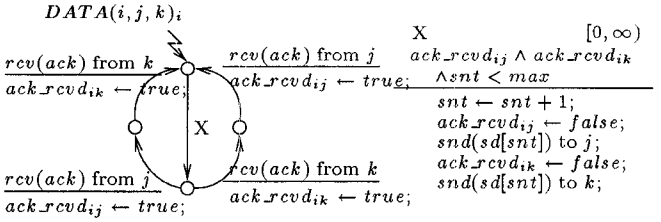

Figure 2: A Composition of Two Data Transfer Protocols

Lemma 3 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$ be the component protocols. If σ is a valid sequence in $N_p = (P_1 | sync(P_1, Q_1) | Q_1, P_2)$ (or $N_q = (P_1 | sync(P_1, Q_1) | Q_1, Q_2)$), where $sync(P_1, Q_1)$ satisfies R_1 and R_2, then $\sigma \downarrow_P$(or $\sigma \downarrow_Q$) is a valid sequence in P(or Q), where any transition tr_{pq} with $(tr_p, tr_q) \in sync(P_1, Q_1)$ is in both P and Q.

From Lemma 3, we can obtain the following result which is similar to the one in [5].

Corollary 2 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$ be the component protocols. If I is an invariant of P(Q, resp.) which refers only to $V_{p1} \cup V_{p2}$($V_{q1} \cup V_{q2}$, resp.), then I is an invariant of $N_p = (P_1 | sync(P_1, Q_1) | Q_1, P_2)$($(P_1 | sync(P_1, Q_1) | Q_1, Q_2)$, resp.), where $sync(P_1, Q_1)$ satisfies R_1 and R_2.

Lemma 4 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$ be the component protocols. If P and Q are safe, then $N_p = (P_1 | sync(P_1, Q_1) | Q_1, P_2)$ and $N_q = (P_1 | sync(P_1, Q_1) | Q_1, Q_2)$ are also safe, where $sync(P_1, Q_1)$ satisfies R_1 and R_2.

3.2 Strengthening predicates

In this section, we present a predicate strengthening technique to refine the composite protocol in a stepwise manner while preserving the invariants of the component protocols.

We illustrate the technique by designing a leader election protocol. Throughout this example, we denote $i \oplus 1$ and $i \ominus 1$ for $(i+1)$ modulo n and $(i-1)$ modulo n, respectively, where n is the number of processes in the network. Assume that a network consists of two processes. The protocol for process i is given in Figure 3(a),(b) and the protocol for process $i \oplus 1$ is similar.

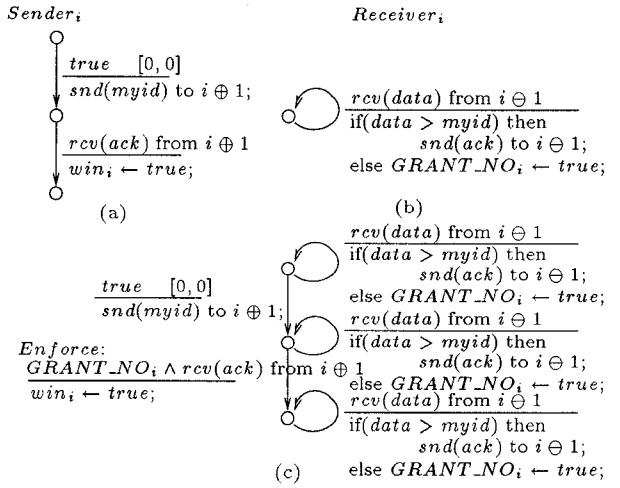

Figure 3: A Leader Election Protocols (a) $Sender_i$, (b) $Receiver_i$ and (c) A Refinement of the Composition $Sender_i ||| Receiver_i$.

Initially, $win_i = win_{i\oplus 1} = GRANT\_NO_i = GRANT\_NO_{i\oplus 1} = false$, and $c_{i,i\oplus 1} = c_{i\oplus 1,i} = \epsilon$. We also assume that the identities of the processes are different. To be the leader, each process sends a message $myid$ to the right process. Upon receiving a message, each process compares the incoming message with its own identity. If the received message is larger, an ack is sent back. Otherwise, $GRANT\_NO$ is set to true to represent that the incoming identity was less than mine. Each process sets win to true only if its identity is larger than that of the other process. Thus, it is easy to show that the following are invariants: $I_1 : (RCV\_ACK_i \Rightarrow Id(i) > Id(i \oplus 1))$ and $I_2 : (GRANT\_NO_i \Rightarrow Id(i) > Id(i \ominus 1))$, where RCV_ACK_i is an auxiliary variable which is true only if process i received an ack from process $i \oplus 1$, and $Id(i)$ is the identity of process i. Now, we will obtain a three-process leader election protocol using a predicate strengthening technique as follows. Consider a network with three processes. We first combine $Sender_i$ and $Receiver_i$ into $Sender_i ||| Receiver_i$. Unlike the previous case, process i cannot elect itself the leader only because it received an ack from the process $i \oplus 1$, since $n = 3$. Process i also needs to assure that the identity of process $i \ominus 1$ is less than its identity before it declares its leadership. Thus, we refine the transition $Enforce$ by adding a condition in a conjunctive manner. Now, by the invariants

I_1 and I_2, we can deduce the correctness of the three-process leader election protocol as follows: $win_i \Rightarrow GRANT\_NO_i \wedge rcv(ack)$ from $i \oplus 1$ (by $Enforce$) $\Rightarrow GRANT\_NO_i \wedge RCV\_ACK_i$ (by the definition of RCV_ACK_i) $\Rightarrow (Id(i) > Id(i \ominus 1)) \wedge RCV\_ACK_i$ (by I_2) $\Rightarrow (Id(i) > Id(i \ominus 1)) \wedge (Id(i) > Id(i \oplus 1))$ (by I_1) $\Rightarrow Id(i) = max(Id(i \ominus 1), Id(i), Id(i \oplus 1))$. Thus, we know that $(win_i \Rightarrow (\neg win_{i \oplus 1} \wedge \neg win_{i \ominus 1}))$ is an invariant, as desired.

Definition 4 Let tr be a transition in a process P. A process P' is a refinement of P if P' is the same as P except that tr is replaced by tr_{ref} which has $cond(tr_{ref}) = \alpha \wedge cond(tr)$, $[l, u](tr_{ref}) = [l, u](tr)$ and $action(tr_{ref}) = action(tr)$, respectively, where α is a predicate.

As shown in the above example, the proper choice of α makes it possible to refine the composite protocol in a stepwise manner while preserving the invariants of the component protocols.

Theorem 1 If I is an invariant of $P|||Q$, then I is an invariant of $P'|||Q$, where P' is a refinement of P.

4 Sequential composition

In this section, we specify a set of interactions, alternating, ordering and disabling, to represent the execution structures between the component protocols, which do not share any message or variable.

In the previous works [4, 7, 9, 10], the safety of the composite protocol depends upon the requirement: If the execution of one machine reaches a "transit" state, then the other machine will eventually reach the matching transit state while consuming the current input messages. In one way or another, therefore, all of the methods mentioned have provided sufficient conditions to avoid the situation that the message transmitted by one machine while executing in one component protocol arrives at the other machine executing in the other component protocol.

On the other hand, we adopt the idea of allowing the above situation to happen and then recovering from it by resuming the protocol's normal operation from the initial state, which is possible due to the expressive power of our extended finite state machine based model. As a result, our sufficient conditions are weaker than those given in [4, 7, 9, 10].

4.1 Alternating functions

An alternating protocol $A = (P_1[]Q_1, P_2[]Q_2)$ consists of $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$ and has the ability to perform the functions of either P or Q, but not both simultaneously.

The recovery scheme in the algorithm works as follows. The algorithm uses an auxiliary variable *mode* to represent the condition of the machine $P_1[]Q_1$. The variable *mode* can be either *normal* or *error*, and is set to *normal*, initially. The algorithm detects an error when the execution of the composite protocol encounters a situation that a message transmitted by one machine executing in one component protocol arrives at the head of the channel connected to the other machine executing in the other component protocol. To recover from the error, the algorithm(step 3) adds a receive transition at each receive state in each component protocol which will be able to handle any message from the other component protocol. Upon receiving such an unexpected message, the composite machine sends the message ABORT to the other composite machine, sets *mode* to *error* to prevent any transitions except for the receive transition generated in step 4 from being enabled, and waits for the message ABORT_ACK from the other composite machine. In the meantime, the other composite machine eventually receives the message ABORT, sends back the message ABORT_ACK, and goes to its initial state. Hence, the composite protocol will resume its execution from the initial state. Similarly, the algorithm can handle a situation that a message transmitted by one machine executing in P arrives at the head of the channel connected to the other machine executing in Q *and vice versa.* Finally, the transitions in step 5 avoid the possibility of the composite protocol being blocked by reaching the receive states $r_p \in S_{pi}$ and $r_q \in S_{qj}$ in $P_i[]Q_i$ and $P_j[]Q_j$, $\{i, j\} = \{1, 2\}$, respectively, which are reachable from (s_{pi}, s_{qi}) and (s_{pj}, s_{qj}) via one(or more) internal transition, respectively.

We give an example to demonstrate the applicability of our construction method. The protocols in Figure 4 are from [4] to compare our construction with the one given in the paper.

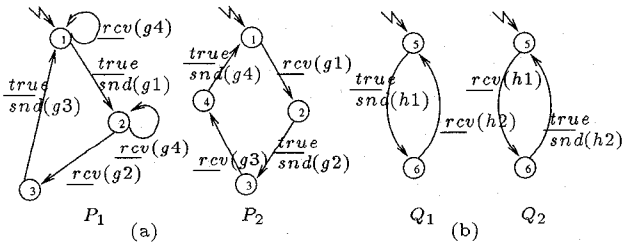

Figure 4: Component Protocols (a) $P = (P_1, P_2)$ and (b) $Q = (Q_1, Q_2)$

Algorithm $P_1[]Q_1$
Input: $P_1 =< S_{p1}, M_p, V_{p1}, s_{p1}, \delta_{p1} >$ and $Q_1 =< S_{q1}, M_q, V_{q1}, s_{q1}, \delta_{q1} >$

Output: $P_1[]Q_1$
Auxiliary Variables: $mode \in \{normal, error\}$, *normal*, initially. $t_p = t_q = 0$, initially.

1. Combine s_{p1} and s_{q1} into (s_{p1}, s_{q1}), the initial state of $P_1[]Q_1$. We assume that $(s_{p1}, s_{q1}) \in S_{p1} \cap S_{q1}$.
2. **for** each transition tr incident from (s_{p1}, s_{q1}) **do**:
   - $cond(tr) \leftarrow [mode = normal \wedge cond(tr)]$;
   - **if** tr is P-inherited **then** $< [l, u](tr) \leftarrow [max(0, l - t_p), u - t_p]$; **if** tr is a self-loop **then** $action(tr) \leftarrow [action(tr); t_p \leftarrow 0; t_q \leftarrow t_q + elapsedtime] >$;
   - **if** tr is Q-inherited **then** $< [l, u](tr) \leftarrow [max(0, l - t_q), u - t_q]$; **if** tr is a self-loop **then** $action(tr) \leftarrow [action(tr); t_q \leftarrow 0; t_p \leftarrow t_p + elapsedtime] >$
3. **for** each receive transition tr **do**:
   (a) **if** $head(tr) \in S_{p1}(S_{q1}$, resp.$)\backslash\{(s_{p1}, s_{q1})\}$ **then** add a transition from $head(tr)$ to (s_{p1}, s_{q1}) labeled $[rcv(m \in M_q(M_p$, resp.$) \cup \{$ABORT$\}) \rightarrow$ **if** (m=ABORT) **then** $\{snd($ABORT_ACK$)$ to $P_2[]Q_2$; INIT;$\}$ **else** $\{snd($ABORT$)$ to $P_2[]Q_2$; $mode \leftarrow error$; INIT;$\}]$, where INIT initiates the variables in $V_{p1} \cup V_{q1}$.
   (b) **if** $head(tr) = (s_{p1}, s_{q1})$ **then** add a self-loop transition at (s_{p1}, s_{q1}) labeled $[rcv($ABORT$) \rightarrow \{snd($ABORT_ACK$)$ to $P_2[]Q_2$; INIT;$\}]$
4. Add a self-loop transition at (s_{p1}, s_{q1}) labeled $[mode = error \wedge rcv(m \in *) \rightarrow$ **if** (m =ABORT_ACK) **then** $\{mode \leftarrow normal;\}$ **if** (m = ABORT) **then** $\{snd($ABORT_ACK$)$ to $P_2[]Q_2;\}]$
5. (a) **if** P_1 and Q_2 have receive states r_{p1} and r_{q2}, respectively, such that all transitions incident from them are receive transitions and $r_{p1}(r_{q2}$, resp.$)$ is reachable from (s_{p1}, s_{q1}) ((s_{p2}, s_{q2}), resp.) via one(or more) internal transition **then** add a transition from r_{p1} to (s_{p1}, s_{q1}) with a timeout interval $[to, to]$ for a sufficiently large to and labeled $[snd($ABORT$)$ to $P_2[]Q_2; mode \leftarrow error$; INIT;$\}]$
   (b) A similar work must be done if P_2 and Q_1 have such receive states.

The next lemma shows that the recovery scheme in the algorithm works by resuming the protocol's normal operation from the initial state.

Lemma 5 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$, where P and Q do not share any variable or message, be the safe component protocols of an alternating protocol $A = (P_1[]Q_1, P_2[]Q_2)$. If a reachable state $g =< u, w, x, y >$ in A satisfies the condition $[(u \in S_{\alpha 1}\backslash\{(s_{p1}, s_{q1})\} \wedge first(x) \in M_\beta) \vee (w \in S_{\beta 2}\backslash\{(s_{p2}, s_{q2})\} \wedge first(y) \in M_\alpha)]$, where $\{\alpha, \beta\} = \{p, q\}$, and none of the component machines have sending/internal cycles, then $P_1[]Q_1$ and $P_2[]Q_2$ proceed to (s_{p1}, s_{q1}) and (s_{p2}, s_{q2}), respectively, with empty incoming channels.

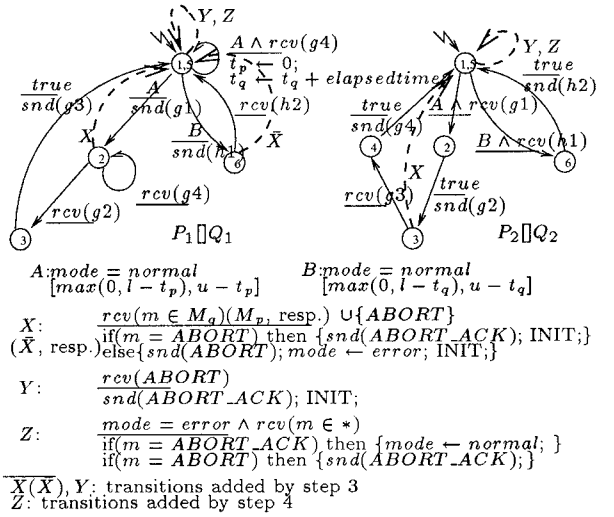

Figure 5: The Resulting Protocol $A = (P_1[]Q_1, P_2[]Q_2)$

Notation 1 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$, where P and Q do not share any variable or message, be the component protocols of an alternating protocol $A = (P_1[]Q_1, P_2[]Q_2)$. Let $g =< u, w, x, y >$ be a reachable state in A. $u^p = u$ if $u \in S_{p1}\backslash\{(s_{p1}, s_{q1})\}$, $u^p = s_{p1}$ otherwise. $u^q = u$ if $u \in S_{q1}\backslash\{(s_{p1}, s_{q1})\}$, $u^q = s_{q1}$ otherwise. $w^p = w$ if $w \in S_{p2}\backslash\{(s_{p2}, s_{q2})\}$, $w^p = s_{p2}$ otherwise. $w^q = w$ if $w \in S_{q2}\backslash\{(s_{p2}, s_{q2})\}$, $w^q = s_{q2}$ otherwise. $x^p(y^p$, resp.$)$ is $x(y$, resp.$)$ except that all messages not in M_p are discarded. $x^q(y^q$, resp.$)$ is $x(y$, resp.$)$ except that all messages not in M_q are discarded.

As long as any message which has been sent by one machine executing in one component protocol can be received by the other machine executing in the same component protocol, the composite protocol is safe provided the component protocols are safe.

Lemma 6 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$, where P and Q do not share any variable or message, be the safe component protocols of an alternating protocol $A = (P_1[]Q_1, P_2[]Q_2)$. If a reachable state $g =< u, w, x, y >$ in A satisfies the condition

KEEP: $[(u \in S_{\alpha 1} \wedge (first(x) \in M_\alpha \vee x = \epsilon)) \wedge (w \in S_{\beta 2} \wedge (first(y) \in M_\beta \vee y = \epsilon))]$, where $\alpha, \beta \in \{p, q\}$, then (1) g is safe, and (2) g^p is reachable in P and g^q is reachable in Q.

Theorem 2 Let $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$, where P and Q do not share any variable or message, be the component protocols of an alternating protocol $A = (P_1 [] Q_1, P_2 [] Q_2)$. If P and Q are safe, and none of the component machines have sending/internal cycles, then A is also safe.

4.2 Ordering functions

An ordering constraint is useful when we design a multiphase protocol in the following manner: when the execution of one component protocol $P = (P_1, P_2)$,"the leading phase," reaches some termination(or connection) point, the other component protocol $Q = (Q_1, Q_2)$, "the trailing phase," is activated.

We devise a construction method for an ordering constraint which can be found in the full paper with the proof of the safety of the resulting protocol.

4.3 Disabling functions

A disabling constraint is useful when the initiation of one component protocol aborts the execution of the other component protocol if it is already running.

We devise a construction method for a disabling constraint which can be found in the full paper with the proof of the safety of the resulting protocol.

5 Conclusion

We have presented a framework for building communication protocols which perform several functions, where each function corresponds to a component protocol.

For parallel composition, we specified a conjunctive relation which requires that the execution of events in two component protocols be delayed until both protocols are ready to execute the respective events. We also proposed a predicate strengthening technique to refine the composite protocol in a stepwise manner while preserving the invariants of the component protocols. The technique allows protocols to be developed in a stepwise manner, where at each step, one may add a correspondence and check whether the addition preserves the correctness of the protocol.

For sequential composition, we presented a set of constraints, alternating, ordering and disabling, to represent the execution structures between the component protocols and gave sufficient conditions for the composite protocol to retain the safety properties, such as freedom from unspecified receptions and freedom from deadlocks. The sufficient conditions are weaker than those given in previous works.

We believe that design of many complex multifunction protocols can be simplified using the techniques. Furthermore, this compositional approach to protocol design makes it possible to reuse the existing protocols which have been designed and analyzed.

However, the techniques given in this paper have some limiting factors. We would like to have techniques from which we can systematically infer liveness properties as well as safety properties from those of the component protocols. The restriction R_2 which confines the type of the transition that can be synchronized in parallel composition needs to be relaxed in some way. The recovery scheme in the sequential composition technique is based on the error-free FIFO channels assumption. It would be desirable to extend the scheme to more general cases. Also, it would be worthwhile considering the "completeness", if any, of the constraints for sequential composition.

References

[1] C. Chow, G. M. Gouda and S. Lam, "A Discipline for Constructing Multiphase Communicating Protocols," *ACM Trans. on Computer Systems*, vol. 3, no. 4, 1985.

[2] T. Y. Choi and R. E. Miller, "A Decomposition Method for the Analysis and Design of Finite State Protocols," *Proc. the 8th Data Communication Symposium*, 1983.

[3] H. Lin, "A Methodology for Constructing Communication Protocols with Multiple Concurrent Functions," *Distributed Computing*, vol. 3, 1988.

[4] H. Lin, "Constructing Protocols with Alternative Functions," *IEEE Trans. on Computers*, vol. 40, no. 4, 1991.

[5] G. Singh, "A Compositional Approach for Designing Protocols," *Proc. IEEE Int'l Conference on Network Protocols*, 1993.

[6] L. Cacciari and O. Rafiq, "A Temporal Reachability Analysis," *Proc. XV IFIP Symp. Protocol Specification, Testing and Verification*, 1995.

[7] H. Lin and C. Tarng, "An Improved Method for Constructing Multiphase Communications Protocols," *IEEE Trans. on Computers*, vol. 42, no. 1, 1993.

[8] C. M. Huang and S. W. Lee, "Timed Protocol Verification for Estelle-Specified Protocols," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 3, 1995.

[9] G. Singh and M. Sammeta, "On the Construction of Multiphase Communication Protocols," *Proc. IEEE Int'l Conference on Network Protocols*, 1994.

[10] G. Singh and Z. Mao, "Structured Design of Communication Protocols," *Proc. IEEE Int'l Conference on Distributed Computing Systems*, 1996.