A Proof Technique for Liveness Properties of Multifunction Composite Protocols [*]

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

Abstract

In protocol composition techniques, component protocols are combined in various ways to obtain a complex protocol whose execution sequences consist of interleaved execution sequences of the component protocols. In this paper, we investigate the problem of verifying liveness properties of the composite protocol from the known properties of its components. We first characterize a class of composition techniques that encompasses almost all composition techniques that have appeared in the literature. We then develop a sufficient condition to ensure that certain liveness properties of the component protocols carry over to the composite protocol. A proof technique, based on this sufficient condition, is then used to determine whether the liveness properties of the component protocols also hold for the composite protocol. To demonstrate the usefulness of our technique, we use several protocols including synchronizing, ordering and disabling protocols as examples. The technique is applicable to any transition based protocol model as long as the model is susceptible to reachability analysis for the sake of correctness proofs.

1. Introduction

Many real-life communication protocols are very complex because they consist of a large number of distinct modules(protocols) interacting concurrently with one another. At some stage of a design process, hence, the specifications of the modules need to be put together in order to obtain an integrated multifunction protocol. This compositional approach may reduce the difficulty of verifying multifunction protocols as a whole. By giving a set of construction rules, this composition approach provides a basis for inferring safety and liveness properties of a composite protocol from those of the component protocols that are typically smaller in size and thus much easier to analyze. Therefore, verification effort for the composite protocol is greatly reduced. It also allows and encourages the reuse of component protocols which have been designed and analyzed. The composition techniques need to be able to specify various real-life behavioral relations among the component protocols, such as sequence(e.g., multiphase protocols like ISO's HDLC(High-level Data Link Control)), alternation(e.g., protocols executing only one function at a time like CCITT's X.21), blocking(e.g., data transfer and disconnection relation in a data link control protocol), synchronization(e.g., multicast protocols), parallelism(e.g., various transmission control protocols in a transport layer), etc. At the same time, however, these techniques should preserve safety and liveness properties of the component protocols throughout the compositions. As a consequence, these composite protocols retain certain desirable properties of the component protocols, from which various properties of the composite protocols are inferred or synthesized.

In this study, we first review a specification model called the timed extended finite state machine model and evolve our proof technique based on this model. However, the introduction of this model is for the purpose of illustrating the technique, and, as a matter of fact, the technique is applicable to any other model as long as the model is susceptible to reachability analysis for the sake of correctness proofs. We then propose a proof technique for liveness properties based on a sufficient condition which requires reachability analysis only for a component protocol. The technique thereby allows certain liveness properties of composite protocols to be inferred from those of the component protocols without investigating the reachability graph of

[*]This research was supported by NSF Grant No. NCR9506039.

the composite protocol. Hence, our approach involves a proof technique along with the synthesis method so that it can take full advantage of the convenience for building and verifying multifunction protocols. We characterize a class of composite protocols to which our proof technique can be applied and illustrate the applicability of the technique by providing various examples which have been used in previous works. Based on this technique, we show that certain liveness properties of a component protocol are preserved in the composite protocol, such as synchronizing [18] and ordering [23] protocols. We also show that the technique is useful for disabling protocols [18] as well, unless it fails to satisfy the sufficient condition. In such a case, we may need to resort to verification techniques on the composite protocol as well.

There have been various techniques for composing *peer* protocols [2, 3, 12, 25, 24, 11, 13, 21, 18]. Our proof system deals with the composite protocols designed by techniques belonging to this approach. These techniques can be divided into two categories: parallel composition [11, 21, 18] and sequential composition [3, 2, 12, 25, 24, 18]. The formalism for specifying the constraints on the actions of the component protocols can be used to produce a variety of composite protocols. Almost every technique in the literature, except in [24], is concerned with safety properties such as freedom from deadlocks, freedom from unspecified receptions, etc, but not with liveness properties. A liveness property asserts that protocol execution eventually reaches some desirable state, and is closely related to the functionality of a protocol. When we synthesize a protocol by combining several components in such a way as parallelism, ordering and alternation, we usually require that certain liveness properties of the component protocols hold in the composite protocol as well. In such a protocol, those liveness properties of the component protocols are synthesized in order to ensure certain liveness properties(or intended functions) of the composite protocols. The lack of results on liveness properties in the context of protocol composition is probably due to the difficulty in developing a construction method or a sufficient condition which allows liveness properties of a composite, especially parallel composite, protocol to be inferred from those of a component protocol without reachability analysis. In fact, some properties, including liveness properties, may require some amount of verification such as reachability analysis even though the properties hold for the components. Here, our technique addresses the problem of verifying liveness properties in the context of protocol composition. For certain liveness properties of a component protocol and composing constraints, we show that a reachability analysis of the component protocol without analyzing the whole reachability graph of the composite protocol is enough to determine whether or not the property holds for the composite protocol. In short, our approach is based on reachability analysis and can be easily automated.

Various works on proving liveness properties of concurrent programs and distributed systems have been proposed [5, 16, 15, 8]. On the other hand, numerous techniques for relieving the state space explosion problem arise in reachability analysis of complex systems [10, 20, 6, 7, 26, 27, 4, 14, 19, 17]. However, it is unclear whether any of these can be applied to a timed model like ours.

The organization of the paper is as follows. In section 2, we discuss the timed extended finite state machine model for protocol specification. Section 3 defines a class of composite protocols and then gives a sufficient condition for such a composite protocol to preserve liveness properties of its component protocols. Several constraints from previous works are presented in section 4 to demonstrate the applicability of our technique. The conclusion and areas for further work are presented in section 5. 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. Formally, a protocol P is a tuple $(\langle P_i \rangle, \langle C_{ij} \rangle)$, where $P_i, 1 \leq i \leq n$, is a process represented by a timed extended finite state machine and $C_{ij}, 1 \leq i, j \leq n, i \neq j$, is the FIFO channel from P_i to P_j.

Each TEFSM P_i is specified as a tuple $(S_{pi}, V_{pi}, E_{pi}, \delta_{pi}, s_{pi}^0)$, where S_{pi} is a finite set of *states*, V_{pi} is a finite set of *local variables*, E_{pi} is a finite set of *events*, $\delta_{pi} : S_{pi} \times E_{pi} \rightarrow S_{pi}$ is a partial state transition function, and s_{pi}^0 is the *initial state*.

Processes can communicate with each other by exchanging messages through unidirectional FIFO *channels*. The contents of a channel C_{ij}, denoted c_{ij}, represents the FIFO queue of messages being received by P_j from P_i, but not yet consumed by P_j. $c_{ij} = \epsilon$ if C_{ij} is empty. Each channel C_{ij} is bounded by a positive integer B_{ij} such that $|c_{ij}| \leq B_{ij}$ at any instance. Also, time bounds $d_{ij} \in \mathcal{N}$ and $D_{ij} \in \mathcal{N}$, where $0 < d_{ij} \leq D_{ij} < \infty$ and $\mathcal{N}$ is the set of integers, are associated with each channel C_{ij}. Any message sent from P_i to P_j is delivered within the interval $[d_{ij}, D_{ij}]$ on $\mathcal{Q}^+$, where $\mathcal{Q}^+$ is the set of non-negative rational numbers, after its transmission. The messages are in fact sent and received by the protocol entities running the processes, and thus the time bounds, in this model, depend on the sender and receiver sites, but not the processes running there. For convenience, however, we simply say "P_i sends(receives) a message to(from) P_j" in case the protocol entity running P_i sends(receives) a message to(from) the protocol entity running P_j.

Each event $e \in E_{pi}$ is associated with a tuple $(en, ac, [l, u])$, where en, called *enabling predicate*, is a boolean function on V_{pi} and/or a set of receptions of messages from the other machines[1], ac, called *action*, consists of a set of statements, possibly a null statement, and $[l, u]$ on $\mathcal{Q}^+$ with $l \in \mathcal{N}$ and $u \in \mathcal{N} \cup \{\infty\}$ is a *time interval* which specifies an upper and lower bound on the time that e may be enabled before occurring. If time limits are not specified, then default values of $[0, \infty)$ are presumed. Elapsed time is measured by a local clock to each process. In addition, the clocks of different processes are uncoupled. Our model itself does not require or assume that the rate of time passing between clocks in different processes must be the same. However, we believe it is reasonable to assume that the rate of time passing should be the same between the processes for reliable temporal operations. For instance, one second in a sender process is "expected" to be regarded as one second in a receiver process as well. For this reason, we further assume that the rate of time passing between clocks in different machines is the same. An event e may be executed when the following conditions hold: (1) the machine is in the head state of e; (2) the enabling predicate $en(e)$ is true; and (3) the timing requirement $[l, u](e)$ is satisfied. A transition e is *enabled* if the machine is in the head state of e and the predicate $en(e)$ is true. The time interval is measured from the point at which the transition e is enabled. An event e is a *non-receive event* if the enabling predicate $en(e)$ does not have any receive condition. A statement is classified as (1) a *local* statement that accesses only local variables in V_{pi}; (2) a *send* statement of the form $snd_{ij}(m)$ that appends a message $\langle m \rangle$ at the tail of the channel C_{ij}, i.e., $c_{ij} \leftarrow c'_{ij} \cdot \langle m \rangle$. An event e is a *receive event* if the enabling predicate $en(e)$ contains any receive condition of the form $rcv_{ji}(m)$. The receive condition $rcv_{ji}(m)$ is *true* if $c_{ji} = \langle m \rangle \cdot c'_{ji}$, i.e., the message m has arrived and is at the front of the channel. The receive event, if executed, deletes the message $\langle m \rangle$ from the head of the channel C_{ji}. If either the message at the head of the channel C_{ji} is $m', m' \neq m$, or $c_{ji} = \epsilon$, any receive event with the receive condition $rcv_{ji}(m)$ is blocked indefinitely. We observe that for the sake of verification [1, 9], it is convenient and sometimes necessary to assume a fictitious global clock that keeps advancing time with the same rate as the local clocks, and records the times of state changes(event occurrences). It is clear that the global clock is imaginary, and must be an aid for describing a global view of the protocol system for the purpose of verification only, but not of specification.

[1]We assume that at most one reception of a message is able to appear in a predicate for each incoming channel of P_i.

The execution of any event is spontaneous in the sense that both the state change and the action associated with the event occur simultaneously, and take no time to complete. From now on, we use 'event' and 'transition' interchangeably since every event in a process is represented by a transition in the corresponding machine. Also, by 'the time at which an event occurs', we mean the time at which an event occurs in terms of the fictitious global clock.

2.2. Scheduling

A process in a protocol is executed by a protocol entity until it reaches the end or an error. 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. Since the time interval associated with each transition provides the upper bound by which the transition should be executed, we do not make any explicit assumption about fairness except that any continuously enabled transition with no finite upper bound must be executed in a finite amount of time(weak fairness). To measure elapsed time, each protocol entity executing its process keeps the elapsed time, called *elapsedtime*, since the protocol entity visited the current state of the process.

2.3. Definitions and Notations

In the rest of the paper, we use a tuple $P = (P_1, P_2, \ldots, P_n)$ to denote a protocol consisting of TEFSMs P_i, $1 \leq i \leq n$, that can communicate with each other via FIFO channels. In this section, we give some definitions and notations used in the rest of the paper.

A *system state* of $P_i = (S_{pi}, V_{pi}, E_{pi}, \delta_{pi}, s^0_{pi})$ is a tuple $(u_i, \langle v_i \rangle)$, where $u_i \in S_{pi}$ is the current state of P_i and $\langle v_i \rangle$ is the tuple of values of the local variables in V_{pi}. A *global state* of $P = (P_1, P_2, \ldots, P_n)$ is a tuple $(\langle x_i \rangle, \langle c_{ij} \rangle, A)$, where x_i is the system state of P_i, c_{ij} represents the contents of the channel C_{ij}, and A, the set of *enabled events* at the state, is $\{(e, [rl, ru]) |$ if $e \in E_{pi}$ then (1) u_i is the head state of e, where $x_i = (u_i, \langle v_i \rangle)$, (2) $en(e)$ is true at $(\langle x_i \rangle, \langle c_{ij} \rangle)$, and (3) the remaining time interval associated with e is $[rl, ru]$, which indicates that the execution of e will be somewhere in the interval $[rl, ru]$. $\}$. The *initial global state* g_0 of $P = (P_1, P_2, \ldots, P_n)$ is the tuple $(\langle s^0_{pi}, \langle v^0_{pi} \rangle \rangle, \langle \epsilon \rangle, A^0)$, where v^0_{pi} is the initial values of the local variables in V_{pi}, and A^0, the set of enabled events at the state g_0, is $\{(e, [rl, ru]) |$if $e \in E_{pi}$ then (1) s^0_{pi} is the head state of e, (2) $en(e)$ is true at $(\langle s^0_{pi}, \langle v^0_{pi} \rangle \rangle, \langle \epsilon \rangle)$, and (3) $[rl, ru] = [l, u](e)$. $\}$.

One of the concerns for dealing with global states is how to represent the evolution of time. Let us notice that if an event e is enabled at a global state g, then it is also enabled at the global state $g \oplus \tau$, where $g \oplus \tau$ is the evolution of g by $\tau > 0$ time units, as long as no other event disabled e in the meantime. Therefore, we do not lose any generality by making a single global state g represent the set of evolutions of g as long as the underlying items are the same.

The *reachability graph* $\mathbf{R}_P$ of $P = (P_1, P_2, \ldots, P_n)$ is the graph that is reachable from the initial global state g_0 of P such that the edges represent the state transitions between the global states due to the occurrence of certain events including sending/receiving messages, updating local variables in P. The readers may refer to [9] for a detailed method for constructing the reachability graph of a given time-dependent protocol.

Notation: (1) For a global state $g = (\langle x_i \rangle, \langle c_{ij} \rangle, A)$ of P, $(\langle x_i \rangle, \langle c_{ij} \rangle)$ is called the *structure tuple* of g. (2) We denote G_P to be the set of all global states in the reachability graph $\mathbf{R}_P$ of the protocol P. (3) The reachability graph $\mathbf{R}^{g_i}_P$ is the portion of the graph $\mathbf{R}_P$ that is reachable from the global state $g_i \in G_P$.

An *execution sequence* from g_0 to g_k in the protocol P is a finite sequence $g_0 \overset{(e_1,t_1)}{\longrightarrow} g_1 \overset{(e_2,t_2)}{\longrightarrow} \cdots \overset{(e_k,t_k)}{\longrightarrow} g_k$, $k > 0$, such that $\forall h, 1 \le h \le k$, e_h is enabled when P is at state g_{h-1} and its occurrence at time t_h results in P entering state g_h. We omit the transitions in an execution sequence $g_i \overset{(e_{i+1},t_{i+1})}{\longrightarrow} \cdots \overset{(e_k,t_k)}{\longrightarrow} g_k$ if they are either clear from the context or of no consequence, and denote the sequence as $g_i \rightarrow \cdots \rightarrow g_k$ or simply $g_i, \ldots, g_k$. An execution sequence in the protocol P can be infinite in which case every prefix of the sequence is a finite execution sequence in P. It is easy to see that an execution sequence $g_0 \overset{(e_1,t_1)}{\longrightarrow} g_1 \overset{(e_2,t_2)}{\longrightarrow} \cdots \overset{(e_k,t_k)}{\longrightarrow} g_k$ in P can be regarded as a path in the reachability graph $\mathbf{R}_P$ such that e_i is in the set of enabled events at the global state g_{i-1} and the occurrence time t_i satisfies the remaining time specification $[rl, ru]$ of e_i, $i \ge 1$.

A *state formula* in $P = (P_1, P_2, \ldots, P_n)$ is a formula that can be evaluated in each structure tuple of a global state of P to be either *true* or *false*. We say that Q *noninterferes with P with respect to(w.r.t., for short)* α, where P and Q are protocols, and α is a state formula in P, iff for every transition e in Q and for each $A = \alpha, \neg\alpha$, the Hoare triple $\{A \wedge en(e)\} e \{A\}$ holds. α *leads-to* β, denoted as $\alpha \leadsto \beta$, where α and β are state formulas in P, iff for any execution sequence $\sigma = g_0, g_1, \ldots$ in P, if α is true at g_i in σ, then there exists a state g_j, $j \ge i$, such that β is true at g_j. In this paper, we consider the liveness properties of the form $\alpha \leadsto \beta$ only.

Notation: (1) Given a global state g and a state formula α in P, we denote $\alpha \in valid(g)$ iff α is true in the state g. (2) Given a transition tr, $head(tr)$ and $tail(tr)$ are respectively the head and tail state of tr. (3) Given a transition tr, $en(tr)$, $ac(tr)$, and $[l, u](tr)$ are respectively the enabling predicate, action, and time interval associated with tr. (4) Given a finite execution sequence σ, $first(\sigma)$ and $last(\sigma)$ denote the first and last transition of σ, respectively. (5) Given an execution sequence σ, we denote $\sigma \downarrow_P$ for the projection of σ onto the transitions of P.[2] (6) Given two execution sequences σ and δ, we denote $\sigma \circ \delta$ for the concatenation of the sequences when $tail(last(\sigma)) = head(first(\delta))$.

3. Preservation of Liveness Properties

In this section, we derive a sufficient condition for a composite protocol to preserve a liveness property of a component protocol. The condition only requires reachability analysis [9, 1] of the component protocol along with the investigation of the constraints imposed by the composite protocol. For ensuring liveness properties, we assume that the composite protocol as well as the component protocols has been proved to progress, that is, be free from deadlocks, unspecified receptions, and channel overflows. We have assumed that every variable in a global state has a finite domain and each channel has a finite bound such that no process in the protocol sends messages over the bound. Also, it is clear that for any structure tuple of a global state, all global states with the same structure tuple have the same set of enabled events. In such a case, the possible number of global states with different set of remaining time intervals for the enabled events is finite, because the number of events capable of generating a different

[2]The sequence of transitions $\sigma \downarrow_P$ is not necessarily a execution sequence in P if σ is an execution sequence of a composite protocol from a set of component protocols including P.

set of remaining time intervals is also finite. Therefore, the reachability graph of any protocol in this paper will be finite. As a result, reachability analysis can decide any logical property including liveness and safety properties of a protocol.

A composite protocol $C = (C_1, C_2, \ldots, C_n)$ achieves an interleaved execution of the component protocols $P = (P_1, P_2, \ldots, P_n)$ and $Q = (Q_1, Q_2, \ldots, Q_n)$ at each site $i, 1 \leq i \leq n$, subject to a set of constraints. Each state in a composite process C_i can be represented as a combined state of the form (u_{pi}, w_{qi}), where u_{pi} and w_{qi} represent the state of P_i and Q_i, respectively. In a composition of the component protocols P and Q, we allow the transitions in P_i,$1 \leq i \leq n$, to affect the execution of Q_i at the same site by accessing the common local variables or messages, or by shifting the control point(current state) of Q_i so that the set of transitions incident from the control point of Q_i can be changed, and vice versa. The constraints, denoted as $const(P, Q)$, are specified as pairs of transitions (tr_p, tr_q) or (tr_q, tr_p), where tr_p and tr_q are transitions of P and Q at the same site, respectively. To synthesize various behaviors of a composite protocol, several constraints were proposed: the ordering constraint [25, 23], the synchronization constraint [21, 22, 18, 23], the disabling constraint [11, 25, 24], etc.

Before we discuss the sufficient condition, we need to define a class of composite protocols.

Definition 1
A composite protocol $C = (C_1, C_2, \ldots, C_n)$ with the constraints *const* (P, Q) from the component protocols $P = (P_1, P_2, \ldots, P_n)$ and $Q = (Q_1, Q_2, \ldots, Q_n)$ is *canonical* iff the following conditions hold:

1. if there exists an execution sequence $r : \cdots \longrightarrow g_{k-1} \xrightarrow{tr} g_k \longrightarrow \cdots, k > 0$, in C such that the occurrence of $tr \in E_{qj}(tr \in E_{pj}$, resp.) has *blocked* an otherwise enabled transition $tr' \in E_{pj}$ $(tr' \in E_{qj}$, resp.) at the state $g_{k+m-1}, m \geq 0$,[3] then $(tr, tr') \in const(P, Q)$.

2. for every execution sequence σ in C, $\sigma \downarrow_P (\sigma \downarrow_Q$, resp.) is an execution sequence in $P(Q$, resp.).

[3]The occurrence of $tr \in E_{qj}$ here can block the occurrence of $tr' \in E_{pj}$ by either (a) making $en(tr')$ false, or (b) jumping to g_k (and thus discarding tr' that had been enabled at g_{k-1}) such that tr' is not incident from g_k through $g_{k+x} (x \geq 0)$, where g_{k+x} is either a state from which a transition in Q_j incident upon the initial state s^0_{qj} is enabled, a state from which a transition in Q_j incident upon a final state of Q_j is enabled(if Q is terminating), or a state at which a transition in P_j other than tr' is enabled. Informally, case (b) says that the occurrence of tr in Q_j shifts the control point of P_j so that tr' is no longer available after that until the disabling process Q_j finishes its current iteration.

Certainly, any undisciplined interleaving of the executions of the component protocols makes it almost impossible to deduce the properties of the composite protocol from those of the component protocols. Condition 1 of the definition is enforced to ensure that any transition of one component protocol capable of blocking a transition of the other component protocol during execution is specified by the constraints of the composite protocol. Also, condition 2 guarantees that during execution no transition of P can make a disabled, if not interleaved, transition of Q enabled and vice versa, which will be stated in the next lemma.

Lemma 1 Let $C = (C_1, C_2, \ldots, C_n)$ be a composite protocol from the component protocols $P = (P_1, P_2, \ldots, P_n)$ and $Q = (Q_1, Q_2, \ldots, Q_n)$. If there exists an execution sequence $\sigma = \sigma_1 \xrightarrow{tr'} g_x$ in C that satisfies the following conditions, then C is not canonical. (1) $tr' \in E_{pj}(E_{qj}$, resp.), and (2) (a) $\sigma_1 \downarrow_P (\sigma_1 \downarrow_Q$, resp.) is an execution sequence in $P(Q$, resp.) and $en(tr')$ is not true at $tail(last(\sigma_1 \downarrow_P))$ $(tail(last(\sigma_1 \downarrow_Q$)), resp.), or (b) there exists $tr \in E_{qj}(E_{pj}$, resp.) in σ_1 such that if $g_{k-1} \xrightarrow{tr} g_k$, then $u^{k-1}_{pj} \neq u^k_{pj} (w^{k-1}_{qj} \neq w^k_{qj}$, resp.) and tr' is incident from $u^k_{pj}(w^k_{qj}$, resp.), where $g_x = (\langle (u^x_{pi}, w^x_{qi}), \langle v^x_{ci} \rangle \rangle, \langle c^x_{ij} \rangle, A^x)$ for $x = k-1, k$.

Lemma 1 ensures that any execution sequence in a canonical composite protocol is an interleaving of a set of execution sequences in the component protocols. It certainly provides an expectation that certain liveness properties of a canonical composite protocol might be able to be inferred from those of its component protocols.

In the next section, we will present various composition constraints that are applied to component protocols to obtain canonical composite protocols.

Lemma 2 Let P and Q be the component protocols of a canonical composite protocol C. Assume that Q noninterferes with P w.r.t. α and β, respectively. If $\alpha \leadsto \beta$ in P, but $\alpha \not\leadsto \beta$ in C, then $\exists u_0 \in G_C$: $\alpha \in valid(u_0)$, $\exists$ a path in $\mathbf{R}_C$ $r : u_0 \longrightarrow \cdots \longrightarrow u_k, k \geq 0$, where

1. $\forall i, 0 \leq i \leq k$: $\beta \notin valid(u_i)$,

2. no transition from P is in $\mathbf{R}^{u_k}_C$, and

3. $true \leadsto \beta$ in $\mathbf{R}^{tail(last(r \downarrow_P))}_P$.

Lemma 3 Let P and Q be the component protocols of a canonical composite protocol C. Assume that Q noninterferes with P w.r.t. α and β, respectively. If $\alpha \leadsto \beta$ in P, but $\alpha \not\leadsto \beta$ in C, then $\exists v_0 \in G_P$: $\alpha \in valid(v_0)$,

$\exists$ a path in $\mathbf{R}_P$ $r' : v_0 \rightarrow \cdots \rightarrow v_m, m \geq 0$, where $\forall i, 0 \leq i \leq m$: $\beta \notin valid(v_i)$ and every transition incident from v_m is either subject to the blocking constraints imposed by C(at least one transition falls into this category) or disabled.

From Lemma 3, we have a sufficient condition for a composite protocol to preserve a liveness property of a component protocol as follows.

Theorem 1 Let P and Q be the component protocols of a canonical composite protocol C with the set of constraints $const(P, Q)$. Assume that $\alpha \leadsto \beta$ in P and Q noninterferes with P w.r.t. α and β, respectively. If there exists no path $r : v_0 \rightarrow \cdots \rightarrow v_m, m \geq 0$, in $\mathbf{R}_P$ such that (1) $\alpha \in valid(v_0)$, (2) $\forall i, 0 \leq i \leq m$: $\beta \notin valid(v_i)$, and (3) every outgoing transition from v_m is either subject to the blocking constraints in $const(P, Q)$ or disabled, then $\alpha \leadsto \beta$ in C.

It is worthwhile to mention that the sufficient condition is conservative in the sense that even if a transition incident from v_m is subject to the blocking constraints, it is still possible for the transition to be executed during execution of C given that a matching blocking transition from Q has not occurred. However, it is certainly undecidable whether a transition from Q will be enabled or not along a certain path in $\mathbf{R}_C$ by simply looking into the reachability graph of P.

4. Protocol Composition Constraints

In this section, we present various composition constraints including those presented in some previous works [11, 21, 25, 23, 24, 18]. We then illustrate the applicability of the sufficient condition given in section 3 for these compositions.

4.1. Synchronizing Constraints

In the following, we discuss a conjunctive constraint which requires the execution of events in two component protocols be delayed until both protocols are ready to execute the respective events. This constraint was studied in [18, 21], and the protocol model in [21] represents each process in a protocol as a set of guarded actions and thus one can be mapped to a process specification in the modeling formalism in [18].

The constraint is for constructing a composite protocol when the component protocols may share messages and update common variables. We start with an algorithm to obtain a composite process given a pair of transitions to be synchronized. The algorithm is from [18].

Algorithm $P_i|(tr_p, tr_q)|Q_i$
Input: $P_i = \langle S_{pi}, V_{pi}, E_{pi}, \delta_{pi}, s^0_{pi}\rangle$, $Q_i = \langle S_{qi}, V_{qi}, E_{qi}, \delta_{qi}, s^0_{qi}\rangle$, and a pair of transitions (tr_p, tr_q) to be synchronized, where $[l, u](tr_p) = [l, u](tr_q) = [0, \infty)$.
Output: $P_i|(tr_p, tr_q)|Q_i$
Auxiliary Variables: $t_p = t_q = 0$, initially.[4]

1. $P_i|(tr_p, tr_q)|Q_i \leftarrow P_i|||Q_i$, where $|||$ is the cross product operator in [18].
2. Let $ac(tr_c)$ be the common part, if any, of $ac(tr_p)$ and $ac(tr_q)$. $ac(tr_{p'})$ and $ac(tr_{q'})$ are the remaining action parts, respectively, such that $ac(tr_p) = [ac(tr_c); ac(tr_{p'})]$ and $ac(tr_q) = [ac(tr_c); ac(tr_{q'})]$.[5]
3. Add a transition tr_{pq} from $(head(tr_p), head(tr_q))$ to $(tail(tr_p), tail(tr_q))$, where $en(tr_{pq}) = [en(tr_p) \wedge en(tr_q)]$ and $ac(tr_{pq}) = [ac(tr_c); ac(tr_{p'}); ac(tr_{q'}); t_p \leftarrow t_q \leftarrow 0;]$.
4. Remove the transitions labeled tr_p or tr_q, and isolated states, if generated, from $P_i|(tr_p, tr_q)|Q_i$.

Figure 1 shows an example for illustrating the algorithm.

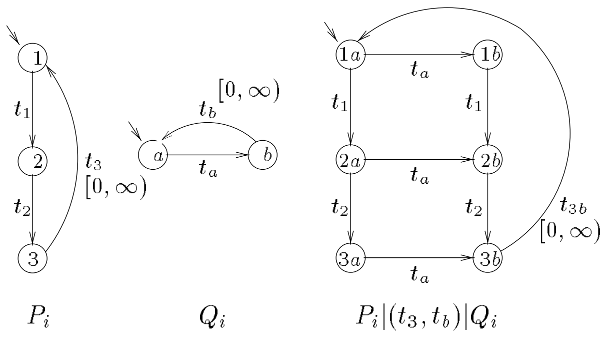

Figure 1. A Composite Process $P_i|(t_3, t_b)|Q_i$

We impose two restrictions here. Let $sync(P_i, Q_i)$ be the set of transition pairs that need to be synchronized with each other for their execution. The restriction R_1 says: for each $(tr_p, tr_q) \in sync(P_i, Q_i)$, if there exists a transition tr(or tr') in P_i(or Q_i) such that $head(tr) = head(tr_p)$(or $head(tr') = head(tr_q)$), then

[4] t_p(t_q, resp.) represents the delay, from P(Q, resp.)'s point of view, taken for the execution of a series of transitions in Q(P, resp.), if any. The variables are necessary to correctly measure the elapsed time for the execution of each transition, originally from P_i or Q_i, in the cross product process $P_i|||Q_i$. Refer to [18] for detail.

[5] It is assumed that the execution order of subactions in $ac(tr_p)$(or $ac(tr_q)$) 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 $ac(tr_p) = a; b; c$.

(tr, tr_q)(or $(tr_p, tr'))$ $\in$ $sync(P_i, Q_i)$, whereas the restriction R_2 says: for any pair $(tr_p, tr_q) \in sync(P_i, Q_i)$, $[l, u](tr_p) = [l, u](tr_q) = [0, \infty)$. The R_2 avoids the possibility that either tr_p or tr_q, $(tr_p, tr_q) \in sync(P_i, Q_i)$, is enabled and must be executed by its bounded upper-bound before the other transition is ready to execute.

The process $P_i|sync(P_i, Q_i)|Q_i$ can be obtained as follows: $P_i|sync(P_i, Q_i)|Q_i \leftarrow [\cap_{(tr_p,tr_q)\in sync(P_i,Q_i)} P_i|(tr_p, tr_q)|Q_i] \cup [\cup_{(tr_p,tr_q)\in sync(P_i,Q_i)}\{tr_{pq}\}]$, where we use the operator $\cap$ to extract the common transitions of the machines and $\cup$ to unite the corresponding transitions. Intuitively, as observed in [21], the composition requires that any transition in P_i and Q_i must be executed asynchronously as long as it does not access any shared variable or message, and the execution of P_i and Q_i must be synchronized at events updating a shared variable or sending/receiving a shared message.

As mentioned before, in order to prove a liveness assertion of a protocol, one has to show that certain safety properties hold along the way. In what follows, hence, we assume that there is no circularity among the synchronizing pairs in $const(P, Q)$ and the composite protocol is guaranteed to progress.

Lemma 4 Let $P = (P_1, P_2, \ldots, P_n)$ and $Q = (Q_1, Q_2, \ldots, Q_n)$ be the component protocols. Then any composite protocol with the constraints $const(P, Q) = sync(P, Q) = \cup_{i=1}^{n} sync(P_i, Q_i)$, where $sync(P_i, Q_i)$ satisfies R_1 and R_2, is canonical.

Now, we are ready to show that any composite protocol with a set of synchronizing pairs preserves certain liveness properties of a component protocol.

Theorem 2 Let $P = (P_1, P_2, \ldots, P_n)$ and $Q = (Q_1, Q_2, \ldots, Q_n)$ be the component protocols. If $\alpha \leadsto \beta$ in P and Q noninterferes with P w.r.t α and β, respectively, then $\alpha \leadsto \beta$ in any composite protocol with the constraints $const(P, Q) = sync(P, Q) = \cup_{i=1}^{n} sync(P_i, Q_i)$, where $sync(P_i, Q_i)$ satisfies R_1 and R_2.

4.2. Ordering Constraints

An ordering constraint is useful when we need to order actions of one component protocol, the leading phase, before those of the other component protocol, the trailing phase. To this end, several different semantics have been proposed [25, 24, 18, 12, 13, 23]. A simple ordering scheme was presented in [23] which allows an interleaving of arbitrary pairs of transitions from the component protocols, while all other interpretations implicitly assumed that the only transitions that are subject to the constraints from the trailing protocol are the ones incident from the initial state of the protocol.

We show that the proof technique in section 3 can be applied to any composite protocol based on the ordering scheme in [23] in the full paper. It should be easy to show that the technique can also be applied to the cases where other interpretations are adopted.

4.3. Disabling Constraints

We use a disabling constraint when the execution of a transition in one protocol will inhibit the execution of a transition in the other protocol. Most of the previous works restricted the application of the constraint as follows: the "initiation" of one component protocol will abort the execution of the other component protocol, because in real-life protocols it seems to be unlikely that a transition other than the ones incident from the initial state of a protocol will abort the execution of the other running protocol.

We show that the proof technique in section 3 can be applied to any composite protocol based on the disabling constraint in [18] in the full paper. Moreover, if we restrict the states in one component protocol P that are vulnerable for interruption by the other component protocol Q as a proper subset of the set of states in P, there exist some liveness properties whose validity in such a composite protocol can be decided by the sufficient condition.

5. Conclusion

We studied the problem of verifying a liveness property of a composite protocol by having this liveness property of the component protocols be preserved during composition. We characterized a class of composite protocols so that the proposed technique could be applied to the class of protocols. The proof technique is based on a sufficient condition which ensures that the liveness properties of such a composite protocol can be inferred from those of its component protocols. We reviewed various composition constraints from previous works and proved that for synchronizing [18] and ordering [23] constraints, some liveness properties of the component protocol are preserved in any composite protocol based on the constraint. Also, for a disabling constraint [18], the technique can be tried for verifying certain liveness properties. If it fails to satisfy the sufficient condition, expensive methods involving direct verification of the composite protocol may be needed. The technique was developed for a timed protocol model based on extended finite state machines, but it is directly applicable to any other model as long as the model is susceptible to reachability analysis for the sake of correctness proofs.

There are several ways for extending the work in this paper. First, it would be nice to see whether we can identify the maximal class of composite protocols and/or constraints to which our technique is applicable. This is related to the problem of finding the weakest sufficient condition. Second, it might be possible to develop a technique that investigates not only P's but also Q's reachability graph in a disciplined manner. The technique in [23] might be useful to this end. Finally, one could further study whether any 'synthesized' liveness properties could be validated by a similar technique. For example, if $\alpha \leadsto \beta$ in P and $\beta \leadsto \gamma$ in Q, then a composite protocol with an ordering constraint might be able to satisfy $\alpha \leadsto \gamma$ under some restrictions.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments.

References

[1] L. Cacciari and O. Rafiq. A Temporal Reachability Analysis. In *Proc. XV IFIP Symp. Protocol Specification, Testing and Verification*, pages 35–49, 1995.

[2] T. Y. Choi and R. E. Miller. A Decomposition Method for the Analysis and Design of Finite State Protocols. In *Proc. 8th Data Communication Symp.*, pages 167–176, October 1983.

[3] C. Chow, M. G. Gouda, and S. Lam. A Discipline for Constructing Multiphase Communication Protocols. *ACM Trans. of Computer Systems*, 3(4):315–343, 1985.

[4] P. Godefroid and P. Wolper. A Partial Approach to Model Checking. *Information and Computation*, 110(2):305–326, 1994.

[5] M. G. Gouda and C. K. Chang. Proving Liveness for Networks of Communicating Finite State Machines. *ACM Trans. on Programming Languages and Systems*, 8(1):154–182, 1986.

[6] M. G. Gouda and J. Y. Han. Protocol Validation by Fair Progress State Exploration. *Computer Networks and ISDN Systems*, 9:353–361, 1985.

[7] M. G. Gouda and Y. T. Yu. Protocol Validation by Maximal Progress State Exploration. *IEEE Trans. on Communications*, COM-32(1):94–97, 1984.

[8] B. T. Hailpern and S. S. Owicki. Modular Verification of Computer Communication Protocols. *IEEE Trans. on Communications*, COM-31(1):56–68, 1983.

[9] C.-M. Huang and S.-W. Lee. Timed Protocol Verification for Estelle-Specified Protocols. *ACM SIGCOMM Comput. Commun. Review*, 25(3):5–32, July 1995.

[10] S. S. Lam and A. U. Shankar. Protocol Verification via Projections. *IEEE Trans. on Software Engineering*, 10(4):325–342, 1984.

[11] H. Lin. A Methodology for Constructing Communication Protocols with Multiple Concurrent Functions. *Distributed Computing*, 3:23–40, 1988.

[12] H. Lin. Constructing Protocols with Alternative Functions. *IEEE Trans. on Computers*, 40(4):376–386, 1991.

[13] H. Lin and C. Tarng. An Improved Method for Constructing Multiphase Communications Protocols. *IEEE Trans. on Computers*, 42(1):15–26, 1993.

[14] H. Liu and R. E. Miller. Partial-Order Validation for Multi-Process Protocols Modeled as Communicating Finite State Machines. In *Proc. IEEE Int'l Conf. on Network Protocols*, pages 76–83, 1996.

[15] J. Misra and K. M. Chandy. Proof of Networks of Processes. *IEEE Trans. on Software Engineering*, 7(4):417–426, 1981.

[16] S. S. Owicki and L. Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Trans. on Programming Languages and Systems*, 4(3):455–495, 1982.

[17] K. Özdemir and H. Ural. Protocol Validation by Simultaneous Reachability Analysis. Technical Report TR-95-09, Dept. of Computer Science and Information, Univ. of Ottawa, March 1995.

[18] J. C. Park and R. E. Miller. A Compositional Approach for Designing Multifunction Time-Dependent Protocols. In *Proc. IEEE Int'l Conf. Network Protocols*, pages 105–112, 1997.

[19] D. Peled. Combining Partial Order Reductions with On-the-fly Model-Checking. In *Proc. 6th Int'l Conf. on Computer Aided Verification*, pages 377–390. Springer-Verlag, 1994.

[20] J. Rubin and C. H. West. An Improved Protocol Validation Technique. *Computer Networks and ISDN Systems*, 6:65–73, 1982.

[21] G. Singh. A Compositional Approach for Designing Protocols. In *Proc. IEEE Int'l Conf. Network Protocols*, pages 98–105, 1993.

[22] G. Singh. A Methodology for Constructing Communication Protocols. In *Proc. ACM SIGCOMM Symp.*, pages 245–255, 1994.

[23] G. Singh and H. Liu. Validating Protocol Composition for Progress by Parallel Step Reachability. In *Proc. IFIP Int'l Conf. on FORTE/PSTV'97*, 1997.

[24] G. Singh and Z. Mao. Structured Design of Communication Protocols. In *Proc. IEEE Int'l Conf. Dist. Comp. Syst.*, pages 360–367, 1996.

[25] G. Singh and M. Sammeta. On the Construction of Multiphase Communication Protocols. In *Proc. IEEE Int'l Conf. Network Protocols*, pages 151–158, 1994.

[26] A. Valmari. Stubborn Sets for Reduced State Space Generation. In *Proc. 10th Int'l Conf. on Application and Theory of Petri Nets*, volume 2, pages 1–22, 1989.

[27] A. Valmari. On-the-fly Verification of Stubborn Sets. In *Proc. 5th Int'l Conf. on Computer Aided Verification, LNCS 697*, pages 397–408. Springer-Verlag, 1993.