15
Validating Protocol Composition for Progress by Parallel Step Reachability Analysis

Gurdip Singh*
Dept. of Computing and Information Sciences, Kansas State University
234 Nichols Hall, Manhattan, KS 66506, singh@cis.ksu.edu

Hong Liu
Bellcore Applied Research, MCC 1J-222R
445 South Street, Morristown, NJ 07960, lhong@bellcore.com

Abstract

In this paper, we propose a parallel step exploration technique for protocol validation in the context of protocol composition. A protocol is modeled as a network of extended communicating finite state machines (ECFSM's). A composite protocol is defined as an interleaved execution of a set of component protocols subject to a set of constraints such as *synchronization*, *ordering* and *inhibition*. By encoding the constraints into the component protocols and the analysis algorithm, our method keeps each process in the component protocols as a separate entity and performs validation without constructing the composite protocol explicitly. We show that our technique not only achieves significant state reduction but also preserves the progress property of the composite protocol in the reduced state space. To our best knowledge, this is the first attempt to adapt existing state reduction techniques to the validation of protocol composition.

1 INTRODUCTION

Designing a correct protocol is a challenging task due to the complex interactions among communicating entities. One way to tackle the complexity in protocol design and analysis is through *composition*, where one divides the functionality of a protocol into subfunctions, develops component protocols for the subfunctions, and then combines them to obtain the composite protocol for the original problem. [CGL85, CM86, LT93, S94a] discuss methods for constructing a multiphase protocol,whereas [Lin88, Lin91, S93, S94b] study techniques for constructing protocols which performed multiple functions at the same time.

All these techniques impose sufficient conditions on the component protocols so that properties of the composite protocol can be inferred from those of the component protocols (which are smaller in size and therefore easier to analyze). While the analysis of the composite protocol is avoided, the sufficient conditions restrict the class of protocols that can be composed – one might still be able to construct correct protocols from a set of component protocols which do not satisfy those conditions. In this setting, the composite protocol needs to be validated for correctness.

Many techniques have been proposed to tackle the *state explosion* problem in protocol validation by eliminating redundant interleaving of independent transitions in different processes during state exploration. (Informally, two transitions are independent if they cannot enable or disable each

*This work was supported by NSF under grants CCR9211621 and CCR9502506.

Formal Description Techniques and Protocol Specification, Testing and Verification
T. Mizuno, N. Shiratori, T. Higashino & A. Togashi (Eds.)

other; otherwise they are dependent.) The *partial order* based techniques [V90, HGW92, GW93, GW94, P93, P94, LM96c] select a representative sequential execution for each set of equivalent transition sequences, while other techniques allow more than one process to make progress in a single step [YG82, RW82, II83, GH85, ZB86, CR93, OU94, LM96a, LM96b, LM96c, SU96].

In this paper, a protocol is modeled as a set of extended communicating finite state machines (ECFSM's). A composite protocol is then modeled as an interleaved execution of a set of component protocols subject to a set of constraints such as *synchronization*, *ordering* and *inhibition*. The ordering constraint was used for sequential composition in [S94a], while the synchronization constraint was used for parallel composition in [S93, S94b]. These constraints can be combined to produce a variety of composite protocols, such as serial-parallel compositions. Other than those constraints, we impose no additional restrictions on the component protocols.

Although existing techniques can potentially reduce the state space drastically, they might not be most effective if applied to the composite protocol directly. Suppose we construct R from two component protocols P and Q, and a set of constraints by composing P_i and Q_i into R_i at each site i. By definition, independent transitions can only come from *different* processes. However, in constructing R_i from P_i and Q_i, we are in fact putting many originally independent transitions between P_i and Q_i into R_i to make them artificially dependent for subsequent validation.

To achieve greater state reduction, we propose a modification to the parallel step reachability analysis [OU94] that keeps P_i and Q_i as separate entities so that both can make progress in parallel during state exploration. This provides us with a much larger set of independent actions and allows us to exploit concurrency between actions in the component protocols. By encoding the constraints into each process and the validation algorithm, we are able to enforce the composition constraints on-the-fly. We show that our technique significantly reduces the state space explored while preserves the progress property of the composite protocol.

The rest of the paper is organized as follows: Section 2 introduces the extended communicating finite state machines as the model for protocol specification; Section 3 formally specifies the composition constraints and presents a algorithm to construct a composite protocol. Our parallel step reachability analysis technique is described in Section 4. An example is given Section 5. Conclusion and future work are given in Section 6. Due to space limitations, we only outline the algorithms and omit the proofs of theorems. Please refer to the full paper [SL97] for details.

2 THE ECFSM MODEL

A communication *protocol* P is modeled as a network of $n \geq 2$ *extended communicating finite state machines* (ECFSM's), denoted as $P = P_1||P_2||\ldots||P_n$. Each P_i is a finite state machine with local variables, denoted as $(A_i, V_i, X_i, T_i, x_i^0)$, where A_i is a finite set of *actions*, V_i is a finite set of *local variables*, X_i is a finite set of *local states*, T_i is the *transition relation*, and x_i^0 is the *initial* local state. Processes exchange messages via uni-directional FIFO *channels*. The channel from P_i to P_j is denoted as C_{ij}, bounded by a positive integer B_{ij}. The content of C_{ij} is denoted as c_{ij}. $c_{ij} = \epsilon$ if C_{ij} is empty. Define $first(c_{ij}) = m$ if $c_{ij} = m \cdot c'_{ij}$; $first(c_{ij}) = \epsilon$ if $c_{ij} = \epsilon$.

An action $a \in A_i$ is of the form $en(a) \longrightarrow a$, where $en(a)$ is a boolean function on V_i, called the *enabling condition* of a, and a is the associated computation consisting of a non-empty sequence of *statements* separated by ";". A statement is either a *local* statement involving only local variables, a *send* statement $P_j!m$ appending m at the end of channel C_{ij}, or a *receive* statement $P_j?m$ removing m from the head of channel C_{ji} if $first(c_{ji}) = m$. We assume that each action contains

at most one send or receive statement. We omit the enabling condition if it is identically true. The transition relation T_i consists of tuples of the form (s_i, a, s_i'), also denoted as $s_i \xrightarrow{a} s_i'$ or $succ(s_i, a) = s_i'$. T_i is assumed to be deterministic but can be partially defined. We usually refer to transition (s_i, a, s_i') as transition a defined at s_i, or transition a if s_i is clear from the context.

Let $< v_i >$ be the tuple of values of the local variables of V_i. A *state* of P_i is defined as $s_i = (x_i, < v_i >)$. The *initial* state of P_i is denoted as $s_i^0 = (x_i^0, < v_i^0 >)$, where $< v_i^0 >$ are the initial values of the local variables. We assume that each local variable has a finite domain. So each P_i has a finite number of states. A *global state* of P is defined as $S_P = (< s_i >, < c_{ij} >)$, where s_i denotes the state of P_i and c_{ij} denotes the content of channel C_{ij}. The *initial* global state of P is denoted as $S_P^0 = (< s_i^0 >, < \epsilon_{ij} >)$, where ϵ_{ij} denotes $c_{ij} = \epsilon$.

Given a global state $S_P = (< s_i >, < c_{ij} >)$ and a transition a defined at s_i, let *stmt* be a statement of a. *stmt* is *enabled* in S_P iff (1) $en(a)$ is true in S_P; (2) if $stmt = P_j!m$ then $|c_{ij}| < B_{ij}$ in S_P; and (3) if $stmt = P_j?m$ then $c_{ji} = m \cdot c'_{ji}$ in S_P. Transition a is *enabled* in S_P if all the statements in a are enabled in S_P; otherwise it is *disabled* in S_P. The set of enabled and disabled transitions in S_P are denoted as $enabled(P_i, S_P)$ and $disabled(P_i, S_P)$, respectively.

The execution of a in S_P is assumed to be *atomic*. If a is executed, it will result in a global state S'_P of P such that (1) $s_i' = succ(s_i, a)$; (2) $c'_{ij} = c_{ij} \cdot m$ if a contains a send statement $P_j!m$; (3) $c_{ji} = m \cdot c'_{ji}$ if a contains a receive statement $P_j?m$; and (4) the rest of the elements in S'_P remain the same as those in S_P. We say S'_P is *directly reachable* from S_P via a, denoted as $S_P \xrightarrow{a} S'_P$ or $S'_P = succ(S_P, a)$. S'_P is *directly reachable* from S_P, denoted as $S_P \mapsto S'_P$, iff $S'_P = succ(S_P, a)$ for some action a. Denote $\mapsto^*$ as the reflexive, transitive closure of $\mapsto$. S'_P is *reachable* from S_P iff $S_P \mapsto^* S'_P$. When $S_P = S_P^0$, S'_P is a reachable global state. A reachable global state S_P is *non-progress* if there is no transition enabled in S_P. The set of reachable global states of P is denoted as $\mathbf{R}_P$.

Suppose $S_P \mapsto^* S'_P$, an *execution sequence* from S_P to S'_P is a finite sequence $ex \triangleq Z_P^0 \xrightarrow{t_1} Z_P^1 \xrightarrow{t_2} \ldots \xrightarrow{t_k} Z_P^k, k \geq 0$, such that $Z_P^0 = S_P, Z_P^k = S'_P$, and $\forall h, 1 \leq h \leq k : Z_P^h = succ(Z_P^{h-1}, t_h)$. When $S_P = S_P^0$, ex is called an execution sequence for S'_P. The length of ex is defined as the number of transitions in ex, denoted as $|ex| = k \geq 0$. The set of execution sequences from S_P^0 is denoted as $behaviors(P)$.

Two transitions a and b are *independent* in a global state S_P if: (1) If a is enabled in S_P, then b is enabled in S_P iff it is also enabled in $succ(S_P, a)$; and (2) If b is enabled in S_P, then a is enabled in S_P iff it is also enabled in $succ(S_P, b)$; and (3) If both a and b are enabled in S_P, then $succ(succ(S_P, a), b) = succ(succ(S_P, b), a)$. Otherwise, a and b are *dependent*. By definition, all transitions in the same process are dependent.

Since we assume that each local variable in P_i has a finite domain, each channel has a finite capacity, and a send statement is blocked if the destination channel is full, it follows that $\mathbf{R}_P$ is finite. As a result, it is decidable whether P has the required progress property.

3 COMPOSITION OF PROTOCOLS

The composite protocol R from P and Q is defined as an interleaved execution of P and Q at each site subject to a set of constraints. Without loss of generality, we make the following assumptions: (1) P and Q have the same number of processes, with P_i and Q_i running at site i. R_i is constructed from P_i and Q_i and a set of constraints on their actions. (2) The send and receive statements

from P_i and Q_i operate on the same set of channels in R_i. So each send statement $P_j!m$ ($Q_j!m'$) from P_i (Q_i) is renamed as $R_j!m$ ($R_j!m'$), and each receive statement $P_j?m$ ($Q_j?m'$) from P_i (Q_i) is renamed as $R_j?m$ ($R_j?m'$); (3) The message sets of P_i and Q_i are disjoint, and so are the local variable sets. This can be ensured through proper renaming; (4) The bound on a channel in R is the sum of the bounds on the same channel in P and Q.

3.1 Specifying the Constraints

We first define a cross product operator $\times$ for P_i and Q_i. Let $P_i = (pA_i, pV_i, pX_i, pT_i, px_i^0)$* and $Q_i = (qA_i, qV_i, qX_i, qT_i, qx_i^0)$ Then $G_i = P_i \times Q_i$ is an ECFSM $(gA_i, gV_i, gX_i, gT_i, gx_i^0)$ such that $gA_i = pA_i \cup qA_i, gV_i = pV_i \cup qV_i, gX_i = \{(px_i, qx_i) | (px_i \in pX_i) \wedge (qx_i \in qX_i)\}$ and $gx_i^0 = (px_i^0, qx_i^0)$. A state of G_i is denoted as $gs_i = (gx_i, < v_i >)$. The initial state of G_i is denoted as $gs_i^0 = (gx_i^0, < v_i^0 >)$. gs_i and gs_i^0 can be rewritten as (ps_i, qs_i) and (ps_i^0, qs_i^0), respectively. gT_i consists of tuples of the form (gs_i, c, gs_i'), where $gs_i = (ps_i, qs_i)$ and $gs_i' = (ps_i', qs_i')$, such that if $c \in pA_i$, then $(ps_i, c, ps_i') \in pT_i$ and $qs_i' = qs_i$; if $c \in qA_i$, then $(qs_i, c, qs_i') \in qT_i$ and $ps_i' = ps_i$.

Let $G = G_1 || G_2 || \ldots || G_n$ be the resulting protocol, denoted as $G = P \times Q$. The set of constraints on P_i and Q_i are imposed on the set of behaviors of G. We have identified three types of constraints: *synchronization*, *ordering* and *inhibition*. These constraints are specified as pairs of actions (a, b) or (b, a), where a and b are actions of P_i and Q_i, respectively. Let $ex \triangleq S_G^0 \xrightarrow{t_1} S_G^1 \xrightarrow{t_2} \ldots \xrightarrow{t_k} S_G^k$ be an execution sequence of G, where S_G^0 is the initial global state of G. Let a^x and b^x be the x^{th} occurrence of a and b in ex, respectively; and l_a and l_b be the number of occurrences of a and b in ex, respectively.

- ex satisfies the *synchronization constraint* (a, b) between P_i and Q_i if (1) $|l_a - l_b| \leq 1$; and (2) $\forall x, 1 \leq x \leq min(l_a, l_b)$: Let $t_h = a^x$ and $t_l = b^x$, if $h < l$ then both a and b are enabled in S_G^{h-1} and $\forall j, h < j < l : t_j$ is not an action of P_i or Q_i. A similar condition must hold if $l < h$; and (3) If $l_a > l_b$ and the last occurrence of a in ex is t_h, then both a and b are enabled in S_G^{h-1} and $\forall j, h < j \leq k : t_j$ is not an action of P_i or Q_i. A similar condition must hold for $l_b > l_a$. The set of synchronization constraints of P_i and Q_i is denoted as $synch(P_i, Q_i)$.
- ex satisfies the *ordering constraint* (a, b) from P_i to Q_i if $0 \leq l_a - l_b \leq 1$ and $\forall x, 1 \leq x \leq l_b$: (1) If $b^x = t_h$ then $\exists t_l, l < h : t_l = a^x$; and (2) if $a^x = t_h$ and $x > 1$ then $\exists t_l, l < h : t_l = b^{x-1}$. The set of ordering constraints from P_i to Q_i is denoted as $order(P_i, Q_i)$. The set of ordering constraints from Q_i to P_i, denoted as $order(P_i, Q_i)$, can be defined similarly.
- ex satisfies the *inhibition constraint* (a, b) from P_i to Q_i if the following condition is satisfied: If $t_h = a^1$ then there is no t_l such that $l > h$ and $t_l = b^x$ for any $x \geq 1$. The set of inhibition constraints from P_i to Q_i is denoted as $inhibit(P_i, Q_i)$. The set of inhibition constraints from Q_i to P_i, denoted as $inhibit(Q_i, P_i)$, can be defined similarly.

Let $constraints(P_i, Q_i)$ be the set of constraints imposed on site i. Then $constraint(P, Q) = \bigcup_{i=1}^{n} constraints(P_i, Q_i)$ is the set of constraints for composing P and Q. Even though they are defined with respect to (w.r.t) a finite execution sequence, they also apply to infinite behaviors of G. We say an infinite execution sequence satisfies a constraint if every prefix of the sequence satisfies the constraint. Note that while the set of synchronization constraints is a symmetric relation, the sets of ordering and inhibition constraints are not. For the latter two types, we need to distinguish the cases where P_i takes precedence over Q_i from those where Q_i takes precedence

*We add a prefix p to all the elements of protocol P. The same convention applies to protocols G, H, Q and R.

over P_i. We impose the following four requirements for $constraints(P_i, Q_i)$ to be *well-specified* for site i: (1) The set of synchronization, ordering and inhibition constraints be mutually disjoint. (2) Each transition of P_i be synchronized with at most one transition of Q_i and vice versa. This restriction avoids cases during execution where a transition of P_i (Q_i) has to be synchronized with more than one transition of Q_i (P_i) in the same global state of the composite protocol. (3) If a and b are synchronized, they should not both contain receive statements expecting messages from the same channel, since at any global state of G, only one of the two receive statements will be enabled. (4) There be no cyclic dependency in the sets of ordering constraints and inhibition constraints†. $constraints(P, Q)$ is *well-specified* iff each $constraints(P_i, Q_i)$ is well-specified. In the rest of the paper, unless otherwise specified, we assume that $constraints(P, Q)$ is well-specified.

3.2 Constructing the Composite Protocol

The construction of R_i from P_i, Q_i and $constraints(P_i, Q_i)$ is composed of three steps. The first step introduces a set of new variables for each constraint. The next step adds new conjuncts and/or local statements to the transitions in P_i and Q_i. The last step computes R_i from $P_i \times Q_i$ by deleting and modifying those transitions involved in the synchronization constraints.

Step 1: we introduce a new local variable for each constraint (a, b) or (b, a) on P_i and Q_i as follows: (1) $\forall (a, b) \in synch(P_i, Q_i)$, add syn_i^{ab}; (2) $\forall (a, b) \in order(P_i, Q_i)$, add ord_i^{ab}; (3) $\forall (b, a) \in order(Q_i, P_i)$, add ord_i^{ba}; (4) $\forall (a, b) \in inhibit(P_i, Q_i)$, add inh_i^{ab}; (5) $\forall (b, a) \in inhibit(Q_i, P_i)$, add inh_i^{ba}. Except for syn_i^{ab}, which is a three value $\{0, 1, 2\}$ variable with initial value 0, all other variables are boolean variables with initial value *false*.

Let $synV_i$, $ordV_i$ and $inhV_i$ be the sets of synchronization, ordering and inhibition variables introduced, respectively. Let $gsyn_i$ be the conjuction of the propositions $syn_i^{ab} = 0$ for all synchronization variables syn_i^{ab} created above. For each $(a, b) \in synch(a, b)$, let $gsyn_i^{ab}$ be the proposition that is formed by deleting conjunct $syn_i^{ab} = 0$ from $gsyn_i$.

Step 2: We modify each action in P_i and Q_i by adding conjunct(s) and/or local statement(s) to its enabling condition and computation. Specifically, for each $a \in pA_i$ and $b \in qA_i$, we modify a and b as follows:

(1) a (b) is not involved in any constraint in $constraints(P_i, Q_i)$. Then add $gsyn_i$ to $en(a)$ ($en(b)$) as a conjunct if it is not already there.

(2) $(a, b) \in synch(P_i, Q_i)$. Then for a, add $gsyn_i^{ab}$ and $en(b)$ as conjunts to $en(a)$ and add statement "if $syn_i^{ab} = 0$ then $syn_i^{ab} := 1$ else $syn_i^{ab} := 0$" to its computation. For b, add $gsyn_i^{ab}$ and $en(a)$ as conjuncts to $en(b)$ and add statement "if $syn_i^{ab} = 0$ then $syn_i^{ab} := 2$ else $syn_i^{ab} := 0$" to its computation.

(3) $(a, b) \in order(P_i, Q_i)$. Then for a, add conjunct $\neg ord_i^{ab}$ to $en(a)$ and add statement "$ord_i^{ab} := true$" to its computation. For b, add conjunct ord_i^{ab} to $en(b)$ and add statement "$ord_i^{ab} := false$" to its computation. The case for $(b, a) \in order(Q_i, P_i)$ can be carried out similarly.

(4) $(a, b) \in inhibit(P_i, Q_i)$. Then for a, add statement "$inh_i^{ab} := true$" to its computation. For b, add conjunct $\neg inh_i^{ab}$ to $en(b)$. The case for $(b, a) \in inhibit(Q_i, P_i)$ can be carried out similarly.

Intuitively, item (3) enforces the ordering constraints; and item (4) implements the inhibition constraints. However, items (1) and (2) together only partly enforce the synchronization constraints. The missing part will be filled in the next step.

---

†Please refer to the full paper for how these cyclic dependency can be statically checked.

Step 3: We compute R_i from $P_i \times Q_i$ as follows: for each $(a,b) \in synch(P_i, Q_i)$, for each transition $t_a = (px_i, a, px_i')$ in P_i and $t_b = (qx_i, b, qx_i')$ in Q_i, let $rx_i^1 = (px_i, qx_i)$, $rx_i^2 = (px_i', qx_i)$, $rx_i^3 = (px_i, qx_i')$, and $rx_i^4 = (px_i', qx_i')$, then except for these four states, t_a and t_b are removed from any other states in R_i as outgoing transitions. There are four cases to consider:

(1) $(px_i = px_i') \wedge (qx_i = qx_i')$. Both t_a and t_b are self loops in P_i and Q_i, respectively. The two corresponding transitions in R_i are also self loops: (rx_i^1, a, rx_i^1) and (rx_i^1, b, rx_i^1). We add a conjunct $syn_i^{ab} \neq 1$ to $en(a)$ and a conjunct $syn_i^{ab} \neq 2$ to $en(b)$.

(2) $(px_i \neq px_i') \wedge (qx_i = qx_i')$. t_a is not a self loop in P_i but t_b is a self loop in Q_i. The corresponding transition for t_a in R_i is (rx_i^1, a, rx_i^2) and $en(a)$ is enhanced with a new conjunct $syn_i^{ab} \neq 1$. The corresponding transitions of t_b in R_i are (rx_i^1, b, rx_i^1) and (rx_i^2, b, rx_i^2), both of which are self loops. We add a conjunct $syn_i^{ab} = 0$ to $en(b)$ of the first transition and a conjunct $syn_i^{ab} = 1$ to $en(b)$ of the second one.

(3) $(px_i = px_i') \wedge (qx_i \neq qx_i')$. t_a is a self loop in P_i but t_b is not a self loop in Q_i. The corresponding transitions for t_a in R_i are (rx_i^1, a, rx_i^1) (rx_i^3, a, rx_i^3), both of which are self loops. We add a conjunct $syn_i^{ab} = 0$ to $en(a)$ of the first transition and a conjunct $syn_i^{ab} = 2$ to $en(a)$ of the second one. The corresponding transition of t_b in R_i is (rx_i^1, b, rx_i^3) and $en(b)$ is enhanced with a new conjunct $syn_i^{ab} \neq 2$.

(4) $(px_i \neq px_i') \wedge (qx_i \neq qx_i')$. Neither t_a nor t_b is a self loop in P_i or Q_i, respectively. The corresponding transitions of t_a in R_i are (rx_i^1, a, rx_i^2) and (rx_i^3, a, rx_i^4). We add a new conjunct $syn_i^{ab} = 0$ to $en(a)$ in the first transition and a new conjunct $syn_i^{ab} = 2$ to $en(a)$ in the second one. The corresponding transitions of t_b in R_i are (rx_i^1, b, rx_i^3) and (rx_i^2, b, rx_i^4). We add a new conjunct $syn_i^{ab} = 0$ to $en(b)$ in the first transition and a new conjunct $syn_i^{ab} = 1$ to $en(b)$ in the second one.

End of Algorithm

Let $S_R = (< rs_i >, < c_{ij} >)$ be a global state of R. Then $rs_i = ((ps_i, qs_i), < sv_i >, < ov_i >, < iv_i >)$, where $< sv_i >, < ov_i >$, and $< iv_i >$ are local variables values of $synV_i, ordV_i$, and $inhV_i$, respectively. The initial global state of R is denoted as $S_R^0 = (< rs_i^0 >, < \epsilon_{ij} >)$. Here $rs_i^0 = ((ps_i^0, qs_i^0), < sv_i^0 >, < ov_i^0 >, < iv_i^0 >)$, where $< sv_i^0 >, < ov_i^0 >$, and $< iv_i^0 >$ are initial values of variables of $synV_i$ (all 0), $ordV_i$ (all *false*) and $inhV_i$ (all *false*), respectively. Let ex be an execution sequence of R and $ex|_{R_i}$ be the projection of ex on rA_i. Clearly, ex satisfies $constraints(P_i, Q_i)$ iff $ex|_{R_i}$ satisfies each constraint specified in $constraints(P_i, Q_i)$. It can be shown that each R_i thus constructed does ensure that R satisfies $constraints(P, Q)$.

Theorem 1 $\forall ex \in behaviors(R) : \forall i : ex|_{R_i}$ satisfies $constraints(P_i, Q_i)$.

4 PARALLEL STEP REACHABILITY ANALYSIS

There are two major sources that cause dependency between a transition a in P_i and a transition b in Q_i: (1) If (a,b) or (b,a) belongs to $constraints(P_i, Q_i)$, a and b cannot be executed in arbitrary order except for synchronization; (2) If both a and b involve sending a message to the same channel, the channel content will differ by the order in which a and b are executed. To take into account these dependency, we encode $constraints(P_i, Q_i)$ into P_i and Q_i, as was done Step 1 and 2 in Section 3, except that for each $(a,b) \in synch(P_i, Q_i)$, we add a conjunct $syn_i^{ab} \neq 1$ to $en(a)$ and $syn_i^{ab} \neq 2$ to $en(b)$. Then instead of constructing R_i explicitly in Step 3, we view P_i

and Q_i as two *threads* of process H_i in a hypothetical protocol $H = H_1||H_2||\cdots||H_n$ that share the same set of channels and local variables. We then apply parallel step state exploration to validate H, where both P_i and Q_i can make progress from a global state. Finally, we show that the hypothetical protocol H and the composite protocol R have the same progress property.

4.1 The Hypothetical Protocol H

A global state of H is denoted as $S_H = (< hs_i >, < c_{ij} >)$, where hs_i is of the form (ps_i, qs_i). Since hs_i has the same component structure as rs_i, so are S_H and S_R. Define $S_H = \alpha(S_R)$ (or $S_R = \alpha^{-1}(S_H)$) iff S_H and S_R have the same component values. α is a one-to-one mapping from the set of global states in R and the set of global states in H. Since a transition is enabled in $\alpha(S_R)$ if it is enabled in S_R, α is a homomorphism from $\mathbf{R}_R$ to $\mathbf{R}_H$ w.r.t the reachability relation $\mapsto^*$. Let $\alpha(\mathbf{R}_R)$ be the image of $\mathbf{R}_R$ in $\mathbf{R}_H$. Then $\alpha(\mathbf{R}_R) \subseteq \mathbf{R}_H$.

Suppose $S_H = \alpha(S_R)$. It can be shown that S_H is reachable via an execution sequence ex in H if S_R is reachable via ex in R, by induction on $|ex|$. However, the converse is not always true. The main reason is that the additional conjuncts we put on actions a and b for $(a, b) \in synch(P_i, Q_i)$ are not sufficient to ensure that a (or b) occurs first iff b (or a) occurs next. For example, consider case (4) in Step 3. At rx_i^2, the conjunct added to $en(b)$ is $syn^{ab} \neq 2$ instead of $syn_i^{ab} = 1$. When $syn_i^{ab} = 0$, b can be enabled at rx_i^2 in H without executing a at rx_i^1.

To avoid the above situation, we select a subset of $enabled(P_i, S_H)$, denoted as $enabled_p(P_i, S_H)$. A transition $a \in enabled_p(P_i, S_H)$ iff (1) a is not involved in any synchronization constraint, or (2) $(a, b) \in synch(P_i, Q_i)$ and $b \in enabled(Q_i, S_H)$, or (3) $(a, b) \in synch(P_i, Q_i)$, $enabled(P_i, S_H) = \{a\}$, $enabled(Q_i, S_H) = \emptyset$, and $syn_i^{ab} = 2$. a is called a *valid* transition of P_i in S_H. Similarly, a transition $b \in enabled_p(Q_i, S_H)$ iff (1) b is not involved in any synchronization constraint, or (2) $(a, b) \in synch(P_i, Q_i)$ and $a \in enabled(P_i, S_H)$, or (3) $(a, b) \in synch(P_i, Q_i)$, $enabled(P_i, S_H) = \emptyset$, $enabled(Q_i, S_H) = \{b\}$, and $syn_i^{ab} = 1$. b is called a *valid* transition of Q_i in S_H.

Suppose $S_H = \alpha(S_R)$. It can be shown that $enabled_p(P_i, S_H) \cup enabled_p(Q_i, S_H) = enabled(R_i, S_R)$. Define $\mathbf{R}_H^p$ as the set of global states reachable from S_H^0 via only valid transitions. If $S_H \in \mathbf{R}_H^p$ and ex is an execution sequence for S_H composed of only valid transitions, then ex is also an execution sequence for S_R. On the other hand, if ex is an execution sequence for S_R, then it is also an execution sequence for S_H with only valid transitions. Hence $\mathbf{R}_H^p = \alpha(\mathbf{R}_R)$, i.e., α is an isomorphism from $\mathbf{R}_R$ to $\mathbf{R}_H^p$ w.r.t $\mapsto^*$. In particular, a non-progress global state S_R is reachable in R iff $S_H = \alpha(S_R)$ is reachable in H by valid transitions only. So to study the progress property of R, it is sufficient to only generate $\mathbf{R}_H^p$. In the following, we are going to show that it actually suffices to generate only a subset of $\mathbf{R}_H^p$ via parallel step state exploration.

4.2 Parallel Step State Exploration

We partition the set of transitions for P_i defined in a global state S_H as follows. First, the set of enabled transitions is divided into two sets: $local\_enabled(P_i, S_H)$ and $global\_enabled(P_i, S_H)$. Transition $a \in enabled(P_i, S_H)$ is *locally* enabled if a does not contain a send statement that sends a message to the same channel as another transition $b \in enabled(Q_i, S_H)$; otherwise it is *globally* enabled. Then the set of disabled transitions is also partitioned into two sets: $local\_disabled(P_i, S_H)$ and $global\_disabled(P_i, S_H)$. A transition $a \in disabled(P_i, S_H)$ is *locally* disabled if $en(a) = false$, or $en(a) = true$, a contains a receive statement $P_j?m$, and $(first(c_{ji}) = m' \neq m) \wedge (m' \in pM_{ji})$; otherwise it is *globally* disabled. Let $local\_enabled_p(P_i, S_H)$ and $global\_enabled_p(P_i, S_H)$ be the

set of locally and globally enabled valid transitions of P_i in S_H, respectively. The transitions for Q_i in S_H can be partitioned similarly into these four sets.

To define parallel progress, we first compute U_i, the set of *valid transition pairs* for P_i and Q_i in S_H from transitions in $enabled_p(P_i, S_H) \cup global\_disabled(P_i, S_H)$ of P_i and $enabled_p(Q_i, S_H) \cup global\_disabled(Q_i, S_H)$ of Q_i.[‡] There are four cases to consider:

(1) For each $a \in enabled_p(P_i, S_H)$ and $b \in enabled_p(Q_i, S_H)$, $(a, b) \in U_i$, i.e. P_i and Q_i can execute a and b in parallel if (i) $(a, b) \in synch(P_i, Q_i)$ and a and b do not send messages to the same channel, or (ii) neither a nor b is involved in any synchronization or inhibition constraint. Otherwise, $(a, \lambda), (\lambda, b) \in U_i$, where λ is a *null* transition, indicating no progress.

(2) For each $a \in enabled_p(P_i, S_H)$ and $b \in global\_disabled(Q_i, S_H)$, $(a, \lambda) \in U_i$ if a is not involved in any synchronization constraint or a is the only enabled transition.

(3) For each $a \in global\_disabled(P_i, S_H)$ and $b \in enabled_p(Q_i, S_H)$, $(\lambda, b) \in U_i$ if b is not involved in any synchronization constraint or b is the only enabled transition.

(4) If $global\_disabled(P_i, S_H) \neq \emptyset$ and $global\_disabled(Q_i, S_H) \neq \emptyset$, then $(\lambda, \lambda) \in U_i$.

Note that when $(a, b) \in synch(P_i, Q_i)$ but both a and b send messages to the same channel, only one of them can be executed. Suppose a (or b) is chosen, then in the following global state, $syn_i^{ab} = 1$ (or $syn_i^{ab} = 2$). Hence $gsyn_i = false$, which implies that only b (or a) can be enabled. In this case, even b (or a) is not paired with a (or b), it should still be chosen to execute. Also, when $(a, b) \in inhibit(P_i, Q_i)$, even though both are enabled, only one of them can be executed.

Now let $\vec{u}$ be a $2n$-tuple $(< pu_i, qu_i >)$ such that $(pu_i, qu_i) \in U_i$. $\vec{u}$ is a *parallel progress vector* in S_H iff $\exists i : (pu_i, qu_i) \neq (\lambda, \lambda)$. Hence in a parallel progress vector, at least one thread must make progress. Since all the non-null transitions in $\vec{u}$ are independent of each other, the resulting global state S'_H is the same irrespective of the order of execution. In this case, we say that S'_H is *directly parallel* reachable from S_H via $\vec{u}$, denoted as $S_H \xrightarrow{\vec{u}} S'_H$ or $S'_H = succ(S_H, \vec{u})$. With this, we can define parallel reachability relations $\mapsto_p$ and $\mapsto_p^*$, and parallel execution sequence accordingly. Denote $\mathbf{PR}_H$ as the set of parallel reachable states in H and $behaviors_p(H)$ as the set of parallel execution sequences from S_H^0 in H.

Let $linear(\vec{u})$ be the set of permutations on the non-null transitions in $\vec{u}$. Let $\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_k$, $k \geq 0$, be the sequence of progress vectors in a parallel execution sequence pex. Then $linear(pex)$ is defined as $\{\epsilon\}$ if $k = 0$; and as $linear(\vec{u}_1) \cdot linear(\vec{u}_2) \cdots linear(\vec{u}_k)$ if $k \geq 1$, where $\cdot$ is generalized to handle two sets of sequences, i.e., $A \cdot B = \{a \cdot b | (a \in A) \wedge (b \in B)\}$. Since each $ex \in linear(pex)$ is an execution sequence from S_H to S'_H composed of only valid transitions, $\mathbf{PR}_H \subseteq \mathbf{R}_H^p$. Moreover, pex satisfies $constraints(P, Q)$ iff $\forall ex \in linear(pex) : ex$ satisfies $constraints(P, Q)$. Since each ex is also an execution sequence for $S_R = \alpha^{-1}(S_H)$ in R, by Theorem 1, ex satisfies $constraints(P, Q)$ in R.

Theorem 2 $\forall pex \in behaviors_p(H) : pex$ satisfies $constraints(P, Q)$.

Now suppose $S_H \in \mathbf{R}_H^p$, S_H is a *pseudo* non-progress global state iff it has no valid transitions. From the above discussion, we know that S_H is pseudo non-progress global state in H iff $\alpha^{-1}(S_H)$ is a non-progress global state in R. Hence we only need to focus on $\mathbf{SNR}_H$, the set of pseudo non-progress global states in H. On the other hand, suppose $S_H \in \mathbf{PR}_H$, S_H is a *parallel* non-progress global state in H iff it has no parallel progress vectors. By construction, S_H has no parallel progress vectors iff it has no valid transitions. Hence S_H is also a pseudo non-progress global state in H. Let $\mathbf{PNR}_H$ be the set of parallel non-progress global state in H. Then $\mathbf{PNR}_H \subseteq \mathbf{SNR}_H$.

[‡] Due to space limitation, we only highlight the key points here, please refer to the full paper [SL97] for details.

To show the converse, let ex be an execution sequence for a pseudo non-progress global state S_H. Each transition in ex is a valid transition. Denote $pe_i = ex|_{P_i}$ and $qe_i = ex|_{Q_i}$. Then $(< pe_i, qe_i >)$ is a local execution sequence set for S_H. From $(< pe_i, qe_i >)$, we construct a parallel execution sequence pex for S_H as follows. Starting from S_H^0, in Step $k \geq 0$, for each i, we compute (pu_i^{k+1}, qu_i^{k+1}) for P_i and Q_i in global state S_H^k based on the transitions from pe_i and qe_i in S_H^k. Since ex has only a finite number of transitions, the algorithm must terminate in a finite number of steps. Moreover, since each intermediate global state contains at least one valid enabled transition, at the end of the algorithm, the final global state must be S_H. So pex thus constructed is a parallel execution sequence for S_H. (Please refer to the full paper [SL97] for details.) Hence we have $\mathbf{SNR}_H \subseteq \mathbf{PNR}_H$, and thus $\mathbf{PNR}_H = \mathbf{SNR}_H$. Since $\mathbf{SNR}_H$ is exactly the set of non-progress states in R, we have the following result on fault coverage of $\mathbf{PR}_H$.

Theorem 3 Given $S_R \in \mathbf{R}_R$, S_R is non-progress global state in R iff $S_H = \alpha(S_R)$ is a parallel non-progress global state in H.

4.3 Discussion

The parallel step technique described in this section was adapted from the simultaneous reachability analysis method in [OU94] to fit the context of protocol composition. Similar to [SU96], we can use the "sleep-set" concept [GW93, GW94] to further eliminate redundant transitions in computing the set of parallel progress vectors. We can also correlate transitions from different processes, as was done in fair reachability analysis [RW82, GH85, LM96a, LM96b]. Doing so might result in fewer global states, but the computation in each global state becomes more elaborate.

In this paper, we assume that the component protocols have the required progress property. If the composite protocol has non-progress global states, then it is most likely that the composition constraints are not consistent with each other. Hence in analyzing error scenarios, we should focus on the set of constraints involved. Note that not all the non-progress global states are semantically incorrect. For example, if one action inhibits the other and that action corresponds to an exception in the protocol, the protocol may halt in response to that exception. So it is up to the designer to decide whether a non-progress global state is acceptable or not. However, the imposed constraints are not the only cause for non-progress in the composite protocol. This point is more subtle. Recall that in our model, a send statement is blocked if the destination channel is full. In the composite protocol, the bound on a channel might be enlarged. So it is possible that a send action that is not enabled in the original component protocol becomes enabled in the composite protocol. So a process may exhibit new behaviors after the composition. These new behaviors, together with their interactions may also cause non-progress in the composite protocol.

Last but not least, even though we presented our technique in the context of two component protocols P and Q, it can be easily extended to handle cases with more than two component protocols. Furthermore, our technique does not require that all component protocols have the same number of processes, nor does it require that the composition of processes be fixed w.r.t the indices of the processes in each component protocol. All is required is that at most one process from each component protocol can participate in each site in the composite protocol. However, not every component protocol is required to participate in the composition for a site. What we need is a composition schema that describes which process from which component protocol is needed to participate at each site. Once the schema is given, we can define composition constraints for each site with more than one process, and the rest of the work can proceed as described above. In the next section, we will give an example in this general setting.

5 EXAMPLE

Consider a network of four sites shown in Figure 1(a). We want to design a data transfer protocol in which site 1 first establishes connection with sites 2, 3 and 4 and then transfers a sequence of data items to them. Site 1 send the items directly to 2 and 4 and site 2 forwards the data items to 3. We want a stop&wait protocol in which 1 sends the next data item only after all sites have received the previous data item. Finally, site 1 may send a disconnect message at any time after the connection establishment to break the connection.

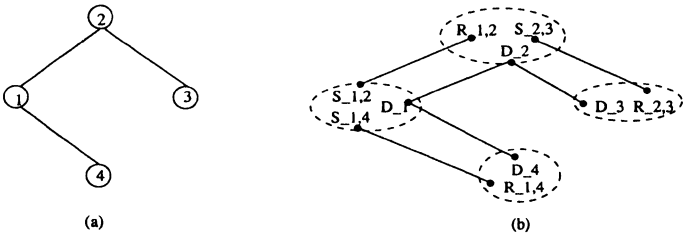

Figure 1 Topology for the data transfer protocol.

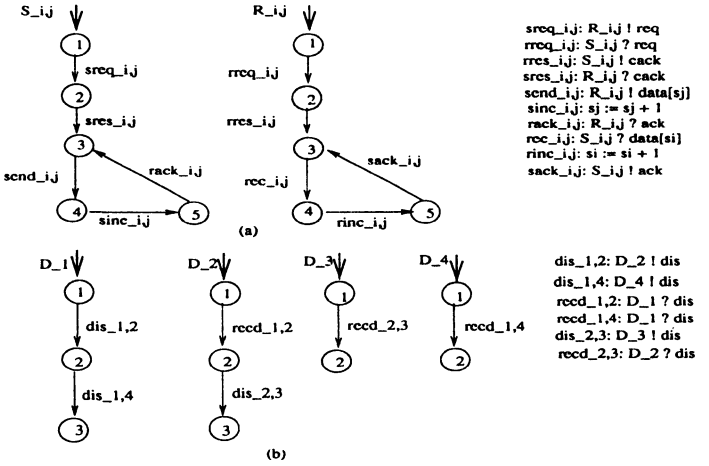

Figure 2 Data transfer component protocols.

Figure 2(a) gives a stop&wait protocol $(S_{i,j}, R_{i,j})$ with $S_{i,j}$ at site i as the sender and $R_{i,j}$ at site j as the receiver. Figure 2(b) gives a disconnect protocol in which 1 simply sends a disconnect message to 2 and 4, and 2 forwards it to 3. We will design protocols using four component protocols: $(S_{1,2}, R_{1,2})$, $(S_{1,4}, R_{1,4})$, $(S_{2,3}, R_{2,3})$ and (D_1, D_2, D_3, D_4) (see Figure 1(b)).

As a simpler case, we first compose $(S_{1,2}, R_{1,2})$ and $(S_{1,4}, R_{1,4})$ with a synchronization constraint $(send_{1,2}, send_{1,4})$ at site 1 to ensure that the first data item is sent after connection with

both 2 and 4 has been established, and subsequent data items are sent only after acknowledgements from both 2 and 4 are received for the previous data item. For the composite protocol built by the algorithm in section 3, the standard reachability analysis explores 126 states, the partial order method in [HGW92] finds 81 states, whereas our method has only 9 reachable states (here we view $inc_{i,j}$ as an internal action; otherwise, the number of states are unbounded). In fact, our method explores 9 states irrespective of the number of receivers.

The next protocol with all four sites is obtained by combining all four protocols with seven more constraints: (1) An ordering constraint $(rec_{1,2}, send_{2,3})$ on 2 to ensure that a data item is forwarded to 3 only after it has been received from 1; (2) An order constraint $(rack_{2,3}, sack_{1,2})$ on 2 to ensure the stop&wait discipline w.r.t 1 and 3; (3) Two ordering constraints $(sres_{1,2}, dis_{1,2})$ and $(sres_{1,4}, dis_{1,4})$ on site 1 to ensure 1 can send a disconnect message only after connection setup; (4) Three inhibition constraints $(dis_{1,2}, send_{1,2})$, $(dis_{1,4}, send_{1,4})$ and $(dis_{2,3}, send_{2,3})$ to ensure no more data items are to be sent after the disconnect message is sent. These constraints allow the messages that have already been sent to be received and acknowledged. Although the final composite protocol is a complex one, our method explores only 47 reachable states.

6 CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of validating protocol composition for progress based on the set of component protocols and a set of composition constraints. By encoding the constraints into the processes of component protocols and the analysis algorithm, we are able to perform parallel step state exploration for the composite protocol without constructing it explicitly. As a result, we are able to perform validation for the composite protocol in a significantly reduced global state space. As far as we know, this is the first attempt to adapt existing state reduction techniques to protocol composition.

However, we have just scratched the surface in this direction. First, the composite protocol construction algorithm given in Section 3 may not be the most efficient one, and the R_i constructed may not be the minimum state machine for the composite process. How to build a minimum state composite process is an interesting problem that requires further study. Second, It would be interesting to investigate other encoding schemes to fit the partial order techniques so that more general properties can be validated. We also want to include more constraint types to allow more flexible compositions. Finally, we plan to implement the parallel step method and experiment it with complex examples.

Acknowledgement

The authors would like to thank Raymond E. Miller and Jun-Cheol Park for their constructive comments on the earlier drafts of this paper.

REFERENCES

[CR93] L. Cacciari and O. Rafiq, "On Improving Reduced Reachability Analysis," Proc. FORTE'92, Perros-Guirec, France, October 13-16, 1992, pp. 137–152.

[CGL85] C.H. Chow, M.G. Gouda and S.S. Lam, "A Discipline for Constructing Multi-Phase Communicating Protocols," ACM Trans. Comput. Syst., 3(4), 1985, pp. 315–343.

[CM86] T.Y. Choi and R.E. Miller, "Protocol Analysis and Synthesis by Structured Partitions", Computer Networks and ISDN Systems, 11, 1986, pp. 367–381.

[GH85] M. Gouda and J.Y. Han, "Protocol Validation by Fair Progress State Exploration," Computer Networks and ISDN Systems, 9, 1985, pp. 353–361.

[GW93] P. Godefroid and P. Wolper, "Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties," Formal Methods in System Design, 2(2), 1993.

[GW94] P. Godefroid and P. Wolper, "A Partial Approach to Model Checking," Information and Computation, 110(2), 1994, pp. 305–326.

[HGW92] G. Holzmann, P. Godefroid and P. Wolper, "Coverage Preserving Reduction Strategies for Reachability Analysis," PSTV'92.

[II83] M. Itoh and H. Ichikawa, "Protocol Verification Algorithm Using Reduced Reachability Analysis," Trans. IECE of Japan, E66(2), 1983, pp. 88–93.

[Lin88] H.A. Lin, "A Methodology for Constructing Communication Protocols with Multiple Concurrent Functions," Distributed Computing, 3(1), 1988, pp. 23–40.

[Lin91] H.A. Lin, "Constructing Protocols with Alternative Functions," IEEE Transactions on Computers, 40(4), 1991, pp. 376–386.

[LT93] H.A. Lin and C.L. Tarng, "An Improved Method for Constructing Multiphase Communications Protocols," IEEE Transactions on Computers, 42(1), 1993, pp. 15–26.

[LM96a] H. Liu and R. Miller, "Generalized Fair Reachability for Cyclic Protocols," IEEE/ACM Transactions on Networking, 4(2), April 1996, pp. 192–204.

[LM96b] H. Liu and R.E. Miller, "An Approach to Cyclic Protocol Validation," Computer Communications, 19(14), 1996, pp. 1175-1187.

[LM96c] H. Liu and R.E. Miller, "Partial-Order Validation for Multi-Process Protocols Modeled as Communicating Finite State Machines," Proc. ICNP'96, Oct. 29 – Nov. 1, 1996, pp. 76–83.

[OU94] K. Özdemir and H. Ural, "Deadlock Detection in CFSM Models via Simultaneously Executable Sets," ICCI'94, Peterborough, Ontario, Canada, May 1994, pp. 673–688.

[P93] D. Peled, "All from One, One for All: On Model Checking Using Representatives," CAV'93.

[P94] D. Peled, "Combining Partial Order Reduction with On-the-fly Model-Checking," CAV'94.

[RW82] J. Rubin and C.H. West, "An Improved Protocol Validation Technique," Computer Networks, 6, 1982, pp. 65–73.

[S93] G. Singh, "A Compositional Approach for Designing Protocols," Proc. ICNP'93, San Francisco, CA, October 19–22, 1993, pp. 98–105.

[S94a] G. Singh and M. Sammeta, "On the Construction of Multiphase Protocols," Proc. ICNP'94, Boston, MA, October 25–28, 1994, pp. 151–158.

[S94b] G. Singh, "A Methodology for Constructing Communication Protocols," Proc. ACM SIGCOMM'94, August 31 – September 2, 1994, London, U.K., pp. 245–255.

[SL97] G. Singh and H. Liu, "Validating Protocol Composition for Progress by Parallel Step Reachability Analysis," *in preparation.*

[SU96] H.v.d. Schoot and H. Ural, "Protocol Verification by Leaping Reachability Analysis," Proc. IC3N'96, Rockville, MD, USA, October 16-19, 1996, pp. 334-339.

[V90] A. Valmari, "A Stubborn Attack on State Explosion," Proc. CAV'90.

[YG82] Y.T. Yu and M.G. Gouda, "Deadlock Detection for a Class of Communicating Finite State Machines," IEEE Transactions on Communications, 30(12), 1982.

[ZB86] J.R. Zhao and G.v. Bochmann, "Reduced Reachability Analysis of Communication Protocols: a New Approach," Proc. PSTV'86, pp. 243–254.