Partial-Order Validation for Multi-Process Protocols Modeled as Communicating Finite State Machines*

Hong Liu†

Bell Communications Research
MCC 1H-212R
Morristown, NJ 07960

Raymond E. Miller

Department of Computer Science
University of Maryland at College Park
College Park, MD 20742

Abstract

In this paper, we adapt the partial order state reduction techniques developed for Petri nets and labeled transition systems to the validation of multi-process protocols modeled as communicating finite state machines. We identify two specific partial order reduction methods in this context: (1) maximal partial order reachability analysis, which is a generalization of maximal reachability analysis to protocols with $n \geq 2$ processes; and (2) simultaneous partial order reachability analysis, which is a counterpart of fair reachability analysis for protocols with $n \geq 2$ processes and arbitrary communication topologies. We study the logical error detection capabilities of each method and show that these two methods, when used together, provide complete logical error coverage for bounded multi-process protocols.

1 Introduction

It is well-known that *state explosion* is the universal limiting factor for state-based automatic verification of distributed/concurrent systems, regardless of the underlying model. To cope with the complexity, many state reduction techniques have been proposed. One class of techniques that has received considerable attention in recent years is called the *partial order* method. It is based on the observation that in many cases, state explosion is caused by the brute-force interleaving of independent transitions. For many interesting temporal properties such as safety and liveness, it is usually sufficient to explore only one such sequence. As a result, most of the intermediate states resulting from exploring other interleaving sequences can be eliminated during the process.

The key concept in the partial order method is the notion of *dependency* among transitions of a system. Informally, two transitions t_1 and t_2 in a state are said to be *independent* if the ending state of executing t_1 followed by t_2 is the same as that of executing t_2 followed by t_1; otherwise t_1 and t_2 are *dependent*. Based on the dependency among transitions, an equivalence relation can be defined over the set of execution sequences of the system, i.e., two execution sequences are *equivalent* if and only if (iff for short) one sequence can be derived from the other by successively shuffling two adjacent independent transitions in the second sequence. By definition, two equivalent execution sequences starting from the same state will end in the same state.

The basic idea of partial order state exploration is the following. In each state S, we compute a subset of executable transitions in S, denoted as V, such that all the transitions in V are dependent with each other and are independent with any other executable transition in S that is not in V. To generate the states directly following S, only those transitions in V are executed. Since V is usually a proper subset of the set of executable transitions in S, fewer states are explored in the next step from S. Depending on the model, V is called a *stubborn set* for *Petri nets* [15], and is called an *ample set* [7] or a *persistent set* [4] for *labeled transition systems* (LTS for short). Even though how V is computed is slightly different in each of the three methods, V has the same property that for any state S' reachable from S via the execution of those transitions outside of V, each transition in V remains executable in any intermediate state from S to S', including S' itself. For this reason, we choose to call this set a persistent set in this paper. It has been shown that using partial order state exploration, one can find all the deadlock states (i.e. states with

*Research reported in this paper was supported by NASA Grant No. NAG 5-2648 and NSF Grant No. NCR9506039.

†This work was performed while the author was with Univ. of Maryland at College Park.

0-8186-7453-9/96 $05.00 © 1996 IEEE

no executable transitions) in an LTS [4] or a Petri net [15].

For the LTS model, the partial order technique can also be combined with the on-the-fly *model-checking* technique to verify properties of concurrent systems specified as nexttime-free linear-time temporal logic formulas. In the context of model-checking, we are given a concurrent system A modeled as $n \geq 2$ finite automata $A_1, A_2, \ldots, A_n$ and a property f specified as a nexttime-free linear time temporal logic formula. The task is to check whether A satisfies f, or whether A is a *model* of f, denoted as $A \models f$. To do that, we first transform $\neg f$ into an equivalent Büchi automata $A_{\neg f}$ (which has a set of *acceptance states*) using the method described in [18]; then we compute the synchronous product A_G of $A_1, A_2, \ldots, A_n$, and $A_{\neg f}$. It is shown that $A \models f$ iff none of the acceptance states in $A_{\neg f}$ are reachable in A_G when f is a safety property or there is no path from the initial state in A_G that traverses a set of acceptance states in $A_{\neg f}$ infinitely many times when f is a liveness property. Consequently, checking whether A is a model of f is reduced to checking the reachability of an acceptance state of $A_{\neg f}$ in A_G or the reachability of an *acceptance cycle* of $A_{\neg f}$ (i.e. a cycle in $A_{\neg f}$ that contains at least on acceptance state) in A_G [2, 17]. It is shown that using certain *proviso* to ensure *unbiased* search, partial order reduction methods can be used to check both safety properties [3, 7] and liveness properties under certain fairness assumptions [4, 13] without constructing the whole synchronous product A_G. Similar results were also established for the Petri net model [15, 16]. We refer the interested readers to respective references for further details.

In protocol engineering, a different model called *communicating finite state machines* (CFSM) is often used to specify protocol behaviors [1]. In this model, communication is asynchronous and channels are assumed to be FIFO and potentially unbounded. For protocol validation, *reachability analysis* is used to check whether a protocol satisfies a set of logical correctness properties such as freedom from deadlock, unspecified reception, unboundedness and nonexecutable transitions. Similar to the Petri nets and the LTS model, many methods have been proposed to cope with the state explosion problem in the CFSM model. Among them, there are two approaches that also make use of equivalent execution sequences to reduce state generation. The first approach is called *maximal progress state exploration* or *maximal reachability analysis* [6]. For a two process protocol, maximal progress state exploration is conducted in two separate phases. In each phase, one process is chosen to maximally progress until it has to wait for a message from the other one. It is shown that with this method, one can detect deadlocks, unspecified receptions, and channel overflows when the protocol is bounded. However, this method is only applicable to two process protocols. Moreover, the amount of additional checking in each step seems to be too complicated and relies heavily on the two-process-two-channel topology of the protocol. It is not clear how it can be generalized to protocols with more than two processes.

The second approach is called *fair progress state exploration* [5, 14]. For a two process protocol, two processes are forced to progress at the same time. It is shown that with this method, one can detect deadlocks, unspecified receptions, and unboundedness. This method is further generalized to handle *cyclic* protocols and *multi-cyclic* protocols with more than two processes. For cyclic protocols, this method can be used to detect deadlocks, unspecified receptions, unboundedness, indefiniteness, and nonexecutable transitions [8, 9]. For multi-cyclic protocols, it can be used to detect deadlocks [10]. The strength of this method is that it can handle some unbounded protocols, provided that their fair reachable state spaces are finite. However, the class of multi-process protocols that are amenable to fair reachability analysis is still limited in terms of the communication topologies. It is conjectured that fair reachability analysis is only feasible for protocols that are multi-cyclic [10]. Again, we refer the interested readers to respective references for details.

In this paper, we investigate how partial order techniques developed in other models can be applied to protocol validation in the CFSM model. In this context, we define a dependency relation among transitions in a state that is an equivalence relation among the set of executable and "potentially" executable transitions in that state. The set of executable transitions in each equivalence class corresponds to a persistent set in that state. We identify two specific partial order techniques based on how these persistent sets are used during state exploration. The first approach is composed of n separate phases. In the i-th phase, only the persistent set whose transitions are dependent on those of process P_i is expanded. This approach can be regarded as generalized maximal reachability analysis for multi-process protocols, and is thus called *maximal partial order reachability analysis.* In the second approach, a progress vector space is formed by taking the cross product of all the persistent sets in a state. State exploration is performed by expanding each progress vector in that state. It can be regarded as a counterpart of fair reachability analysis for gen-

eral multi-process protocols, and is called *simultaneous partial order reachability analysis*. For bounded multi-process protocols, with the first approach, one can detect all the channel overflows and nonexecutable transitions; while with the second approach, one can detect all the deadlocks, indefiniteness, and blocking channels. As a result, these two approaches complement each other and can be used together to ensure a complete fault coverage for multi-process protocol validation.

Recently, Özdemir and Ural proposed a state reduction technique called *simultaneous reachability analysis* for bounded multi-process protocols [11, 12]. Since this technique is closely related to our simultaneous partial order analysis method, we will defer the discussion to Section 3.3, after our method is presented.

The rest of the paper is organized as follows. In the following section, the communicating finite state machines model is introduced. In the next section, a dependency relation is defined among transitions in a state, followed by the presentation of maximal and simultaneous partial order reachability analysis techniques. The conclusion and future work are given in Section 4.

2 The CFSM Model

Notation: (1) We use $\cdot$ to denote concatenation. Given a set M, M^* denotes its reflexive and transitive closure under concatenation; $|M|$ denotes its cardinality. (2) Given $Y, Y' \in M^*$, $|Y|$ denotes its length. ϵ denotes an empty string, $|\epsilon| = 0$. Define $head(Y) = m$ if $Y = m \cdot Y'$, where $m \in M$; $head(Y) = \epsilon$ if $Y = \epsilon$. (3) Given two sets A and B, we use $A \setminus B$ for the set of elements that are in A but not in B. (4) We designate n as the number of processes in a protocol. Unless otherwise specified, we assume $n \geq 2$, $i, j \in \{1..n\}$, and $i \neq j$. (5) We use $< a_i >$ to denote the n-tuple of elements in set $\{a_i | i \in \{1..n\}\}$, arranged by the increasing order of i; and use $< b_{ij} >$ to denote the $n(n-1)$-tuple of elements in set $\{b_{ij} | i, j \in \{1..n\}, i \neq j\}$, arranged first in the increasing order of j, then in the increasing order of i.

In the communicating finite state machine (CFSM) model, a protocol is specified as a set of n processes $P = (P_1, P_2, \ldots, P_n)$, where each process P_i is a finite state machine that can communicate with other processes via uni-directional FIFO channels. For each P_i, $\mathbf{S}_i$ denotes the set of local states in P_i. The *initial* local state of P_i is denoted as s_i^0. A channel from P_i to P_j is denoted as C_{ij}. C_{ij} is called an *output* channel of P_i and an *input* channel of P_j. Define $I_i = \{j | \text{there is a channel } C_{ji} \text{ in } P\}$ and $O_i = \{j | \text{there is a channel } C_{ij} \text{ in } P\}$. The content of C_{ij}, denoted as c_{ij}, is a sequence of messages sent from P_i to P_j. When C_{ij} is empty, $c_{ij} = \epsilon$.

The set of messages that P_i can send to P_j is denoted as M_{ij}. Let $\tilde{M}_i = (\bigcup_{j \neq i} \{-m | m \in M_{ij}\}) \cup (\bigcup_{j \neq i} \{+m | m \in M_{ji}\}) \cup \{\eta\}$, where η denotes the *null* message. τ denotes the partially defined *transition function*: $\bigcup_{i=1}^{n} (\mathbf{S}_i \times \tilde{M}_i \times (\{1..n\} \setminus \{i\}) \rightarrow 2^{\mathbf{S}_i})$. For each P_i, a transition defined at local state $s_i \in \mathbf{S}_i$ is denoted as $\tau(s_i, \sigma, j)$, where $\sigma \in \tilde{M}_i$. It is a *sending* transition if $\sigma = -m$, indicating a transmission of message m to channel C_{ij}; it is a *receiving* transition if $\sigma = +m$, indicating a reception of message m from channel C_{ji}; it is an *internal* transition if $\sigma = \eta$, denoted as $\mu(s_i)$ for short, indicating that P_i only makes a local state change but does not communicate with other processes. A *transition cycle* in P_i, denoted as C_i, is a cycle in the transition graph of P_i. It is an *internal* cycle if each transition in the cycle is an internal transition; it is a *sending* (*receiving*) cycle if it is not an internal cycle and each transition in the cycle is either a sending (receiving) transition or an internal transition. As a convention, we use $t \triangleq \tau(s_i, \sigma, j)$ to give a name t for this transition, and use $s_i' \in \tau(s_i, \sigma, j)$ to mean that s_i' is a local state resulting from the execution of this transition. s_i is a *receiving* local state iff each transition defined at s_i is a receiving transition. By definition, each P_i is nondeterministic but partially defined.

A *global state* (*state* for short) of P is represented as $S = (< s_i >, < c_{ij} >)$, where s_i is the local state of P_i in S and c_{ij} is the content of channel C_{ij} in S. The *initial state* S^0 is denoted as $(s_1^0, s_2^0 \ldots, s_n^0, \overbrace{\epsilon, \ldots, \epsilon, \ldots, \epsilon}^{(n-1)n})$. As a convention, we use capital letters S, S' to denote a state and small letters s_i, s_i' to denote a local state of P_i. We also use s_i **in** S to mean s_i is the local state of P_i in S, and (s_i, m, j) **in** S to mean that s_i **in** S and $m = head(c_{ji})$.

The *reachability relation* among states is formulated as follows. Given two states $S = (< s_i >, < c_{ij} >)$ and $S' = (< s_i' >, < c_{ij}' >)$, S' is *directly reachable* from S, denoted as $S \mapsto S'$, iff $\exists i \in \{1..n\}$ such that the elements of S' can be derived from S by executing one of the following transitions: (1) $s_i' \in \tau(s_i, -m, j)$ and $c_{ij}' = c_{ij} \cdot m$. (2) $s_i' \in \tau(s_i, +m, j)$ and $c_{ji} = m \cdot c_{ji}'$. (3) $s_i' \in \tau(s_i, \eta, j)$. Except for the elements affected by the one transition applied, all other elements of S' remain the same as those in S. Denote $\mapsto^*$ as the reflexive, transitive closure of $\mapsto$. S' is *reachable* from S iff $S \mapsto^* S'$. When $S = S^0$, we say S' is a *reachable state*. The set of reachable states in P is denoted as $\mathbf{R}$, called the *reachable state space* of P. A local state

s_i is *reachable* iff there is a reachable state S' such that s_i **in** S'. (s_i, m, j) is *reachable* iff there is a reachable state S' such that s_i **in** S' and $m = head(c'_{ji})$. C_i is *reachable* iff one of the local states in C_i is reachable.

Suppose $S \mapsto^* S'$. An *execution sequence* from S to S' is a sequence $e = X^0 \xrightarrow{t_1} X^1 \xrightarrow{t_2} \cdots \xrightarrow{t_k} X^k, k \geq 0$, such that (i) $X^0 = S$; (ii) $\forall\, l, 0 < l \leq k : X^{l-1} \mapsto X^l$ via transition t_l; and (iii) $X^k = S'$. The length of e is defined as the number of transitions in e, denoted as $|e| = k$. When $S = S^0$, e is called an execution sequence for S'. Let e_i be the corresponding local execution sequence of P_i. $\{e_1, e_2, \ldots, e_n\}$ is called a *local execution sequence set* from S to S'. We use $e \triangleq \{e_1, e_2, \ldots, e_n\}$ to denote the correspondence. When $S = S^0$, $\{e_1, e_2, \ldots, e_n\}$ is called a local execution sequence set for S'. We use $e_i(S)$ to denote the transition associated with e_i in S. If e_i is at its last local state in S, then $e_i(S) = \lambda$, where λ stands for a *null* transition, indicating no progress from P_i.

For protocol validation, we check $\mathbf{R}$ against the following *logical errors*. Given a reachable state S, S is a *deadlock* state iff each s_i is a receiving local state and each $c_{ij} = \epsilon$; S is an *unspecified reception* state iff $\exists\, i, j : m = head(c_{ji})$ and $\tau(s_i, +m, j)$ is not defined; S is an *indefinite* state iff $\exists\, i : s_i$ is in an internal cycle of P_i. Given a channel C_{ij} in P, C_{ij} *overflows* iff $\exists S \in \mathbf{R} : |c_{ij}| > B_{ij}$, where B_{ij} is the bound imposed on the length of channel C_{ij}; C_{ij} is *unbounded* iff $\forall K \geq 0, \exists S \in \mathbf{R} : |c_{ij}| > K$; C_{ij} is a *blocking channel* iff $m = head(c_{ji})$, and for each S' reachable from $S : m = head(c'_{ji})$ and $\tau(s'_i, +m, j)$ is not defined. Given a transition $t \triangleq \tau(s_i, \sigma, j)$, t is *executable* in S iff $\sigma = -m$, or $\sigma = \eta$, or $\sigma = +m$ and $head(c_{ji}) = m$; t is *nonexecutable* iff it is not executable in any $S \in \mathbf{R}$. It can be shown that none of the above logical errors is decidable for protocols in general, using the results in [1].

We remark that an unspecified reception in S does not necessarily cause nonprogress from S since P_i could have more than one input channel. On the other hand, if a nonprogress state is not a deadlock state, then all the nonempty channels in that state are blocked. For this reason, in this paper, we do not consider unspecified reception detection during validation. Unless otherwise specified, we assume P is bounded (i.e., each channel in P is bounded) in the rest of the paper.

3 Partial Order Protocol Validation

Given a protocol $P = (P_1, P_2, \ldots, P_n)$, let $S = (< s_i >, < c_{ij} >)$ be a state of P. Let D_i be the set of transitions defined at s_i. Define $E_i^- = \{\tau(s_i, -m, j) \in D_i\}$, $E_i^\mu = \{\mu(s_i) \in D_i\}$, and $E_i^+ = \{\tau(s_i, +m, j) \in D_i | head(c_{ji}) = m\}$. Let $E_i = E_i^- \cup E_i^\mu \cup E_i^+$. Then E_i is the set of *executable* transitions of P_i in S. Let $E_i^p = \{\tau(s_i, +m, j) \in D_i | c_{ji} = \epsilon\}$. In this case, $\tau(s_i, +m, j)$ is said to be *potentially executable* in S, meaning that it *might* become executable in a state reachable from S. E_i^p is the set of potentially executable transitions of P_i in S. Denote $\tilde{E}_i = E_i \cup E_i^p$, $\tilde{E} = \bigcup_{i=1}^n \tilde{E}_i$, $E^p = \bigcup_{i=1}^n E_i^p$, and $E = \bigcup_{i=1}^n E_i$.

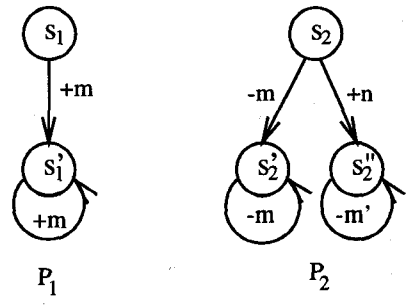

Figure 1: Dependency among transitions in a state

We remark that the notation defined above are implicitly bound to a state S. Whenever distinction is necessary, the binding arguments, such as S, will be put into the notation. For example, we use $E_i(S^1)$ and $E_i(S^2)$ for the set of executable transitions of P_i in S^1 and S^2, respectively. This convention is adopted throughout the paper whenever a new notation is introduced.

The key concept in partial order analysis is the notion of dependency among transitions in a state. Let t and t' be two transitions defined in S. Informally, t is dependent on t' in S if the execution of t' either disables the execution of t in S when $t \in E(S)$, or could disable/enable the execution of t in a state S' reachable from S when $t \notin E(S)$.

Example: Consider a state $S = (s_1, s_2, c_{12}, c_{21})$ of a protocol $P = (P_1, P_2)$, where $c_{12} = n$ and $c_{21} = \epsilon$. Figure 1 shows only parts of the state transition graphs of P_1 and P_2. Let $t \triangleq \tau(s_1, +m, 2)$, $t' \triangleq \tau(s_2, -m, 1)$ and $t'' \triangleq \tau(s_2, +n, 1)$. Then $t \in E_1^p(S)$ and $t', t'' \in E_2(S)$. Clearly, the execution of t' disables the execution of t'' in S and vice versa. Hence t' and t'' are mutually dependent. On the other hand, the execution of t' enables the execution of t in $S' = (s_1, s'_2, n, m)$, while the execution of t'' in S prevents t of becoming executable in $S'' = (s_1, s''_2, \epsilon, \epsilon)$. Hence the executability of t is dependent on t' and t'' in S. *End of Example*

Definition 3.1 Given $S = (< s_i >, < c_{ij} >)$, let $t \in \tilde{E}_i$ and $t' \in \tilde{E}_j$ be two transitions defined at s_i and

s_j, respectively. t and t' are *directly dependent* in S, denoted as $t \sim t'$, iff $(i = j) \vee ((t \triangleq (s_i, +m_{ji}, j)) \wedge (c_{ji} = \epsilon) \wedge (t' \in \tilde{E}_j)) \vee ((t' \triangleq (s_j, +m_{ij}, i)) \wedge (c_{ij} = \epsilon) \wedge (t \in \tilde{E}_i))$. Denote $\sim^*$ as the reflexive, transitive closure of $\sim$. t and t' are *dependent* in S if $t \sim^* t'$; otherwise t and t' are *independent* in S, denoted as $t \not\sim^* t'$.

It is not difficult to show that $\sim^*$ is an equivalence relation on $\tilde{E}$. Since each transition of $\tilde{E}_i$ belongs to the same equivalence class, we can define $\tilde{E}_i \sim^* \tilde{E}_j$ iff $\exists t \in \tilde{E}_i, t' \in \tilde{E}_j : t \sim^* t'$. Let Q be the number of equivalence classes in $\tilde{E}$. Denote $H_h, h \in \{1..Q\}$ as the index set for those $\tilde{E}_i$'s that are in the h-th equivalence class. Denote the set of executable transitions in the h-th equivalence class as V_h, i.e. $V_h = \bigcup_{i \in H_h} E_i$. V_h is called a *persistent set* in S iff $V_h \neq \emptyset$. Let q be the number of persistent sets in S and let h range over $\{1..q\}$. The construction of a persistent set in S is straightforward from the above definition.

Note that if $q > 1$ in S, there is more than one way to choose a persistent set. In the following two subsections, we identify two specific partial order techniques based on how these persistent sets are used during state exploration. By combining these two methods, complete protocol validation is obtained.

3.1 Maximal Partial Order Method

Partial order state exploration can be carried out in n separate phases. In the i-th phase, only the transitions dependent on those of P_i are expanded during state exploration. Since P_i is always given the priority to execute first, this strategy is a generalization of maximal progress reachability analysis for two process protocols [6] to protocols with $n \geq 2$ processes. We call this technique *maximal partial order reachability analysis.*

Definition 3.2 Given a state S, let V_h be the persistent set in S such that $i \in H_h$. S' is *directly maximal reachable* from S w.r.t P_i, denoted as $S \mapsto_i S'$, iff S' is directly reachable from S via a transition in V_h. V_h is called the *maximal transition set* in S w.r.t P_i, denoted as VM_i. Let $\mapsto_i^*$ be the reflexive, transitive closure of $\mapsto_i$. S' is *maximal reachable* from S w.r.t P_i iff $S \mapsto_i^* S'$. When $S = S^0$, S' is said to be maximal reachable w.r.t P_i. The set of maximal reachable states w.r.t P_i is denoted as $\mathbf{R}_i$.

Note that it is possible that there is no $V_h \neq \emptyset$ such that $i \in H_h$ in S. In this case, we simply stop the maximal progress state exploration w.r.t P_i from S. The maximal reachability of $s_i, (s_i, m, k)$, and C_i (from S) w.r.t P_i can be defined accordingly.

Suppose $S \mapsto_i^* S'$. A *maximal execution sequence* from S to S' w.r.t P_i is a sequence $me = X^0 \xrightarrow{t_1} X^1 \xrightarrow{t_2} \cdots \xrightarrow{t_k} X^k, k \geq 0$, such that (i) $X^0 = S$; (ii) $\forall\, l, 1 \leq l \leq k : X^{l-1} \mapsto_i X^l$ via transition $t_l \in VM_i(X^{l-1})$; and (iii) $X^k = S'$. The length of me is defined as the number of transitions in me, denoted as $|me| = k$. When $S = S^0$, me is called a maximal execution sequence for S' w.r.t P_i.

Note that not every reachable state is maximal reachable w.r.t P_i. However, given a reachable state S, let $\{e_1, e_2, \ldots, e_n\}$ be an execution sequence for S, we can construct a *partial maximal execution sequence* $pme = X^0 \xrightarrow{t_1} X^1 \xrightarrow{t_2} \cdots \xrightarrow{t_k} X^k, k \geq 0$, such that (i) $X^0 = S^0$; (ii) $\forall\, l, 1 \leq l \leq k, \exists j : X^{l-1} \mapsto_i X^l$ via $t_l = e_j(X^{l-1}) \in VM_i(X^{l-1})$; (iii) $\forall j : e_j(X^k) \notin VM_i(X^k)$. X^k is called a *maximal precursor* of S w.r.t P_i and $\{e_1, e_2, \ldots, e_n\}$, denoted as $mp_i = (< s_i^{mp_i} >, < c_{ij}^{mp_i} >)$. We remark that mp_i is not always unique w.r.t P_i and $\{e_1, e_2, \ldots, e_n\}$.[1]

Lemma 3.1 Suppose $S \mapsto^* S'$ via $\{e_1, e_2, \ldots, e_n\}$. If $e_i(S) \neq \lambda$, then $\exists j : e_j(S) \in VM_i(S)$. Furthermore, if $S \mapsto_i S''$ via transition $e_j(S) \in VM_i(S)$, then $S'' \mapsto^* S'$ via the remaining transitions of $\{e_1, e_2, \ldots, e_n\}$ in S''.

Lemma 3.2 Let $\{e_1, e_2, \ldots, e_n\}$ be a local execution sequence set for a reachable state S and mp_i be a maximal precursor of S w.r.t P_i and $\{e_1, e_2, \ldots, e_n\}$. Then $e_i(mp_i) = \lambda$, i.e. s_i **in** mp_i. Moreover, If $mp_i \neq S$, then $\exists j : e_j(mp_i) \neq \lambda$ and $mp_i \mapsto^* S$ via the remaining transitions of $\{e_1, e_2, \ldots, e_n\}$ in mp_i.

Lemma 3.3 Given a bound B_{ij} on channel C_{ij}, if there is a reachable state S such that $|c_{ij}| > B_{ij}$, then there is a state S' maximal reachable w.r.t P_i such that $|c'_{ij}| > B_{ij}$.

Lemma 3.4 Given S and (s'_i, m, j), suppose $\tau(s'_i, +m', j)$ is defined, then (s'_i, m, j) is reachable from S iff it is maximal reachable from S w.r.t P_i.

Lemma 3.2 ensures that all the reachable internal cycles of P_i are preserved in $\mathbf{R}_i$; while Lemma 3.2 and Lemma 3.4 together imply that all nonexecutable transitions of P_i can be detected within $\mathbf{R}_i$. To summarize, we have the following theorem on the fault coverage of $\mathbf{R}_i$.

Theorem 3.1 Detection of the following logical errors are decidable within $\mathbf{R}_i$ of P: (1) indefiniteness in P_i; (2) channel overflow of those output channels of P_i; and (3) nonexecutable transition of P_i.

[1] As was observed by one of the referees.

Therefore, if one performs maximal partial order reachability analysis n times, each for one P_i, then one will be able to detect all the indefinite states, channel overflows, and nonexecutable transitions in P. However, one cannot detect the all deadlock states and blocking channels in P using this method.

3.2 Simultaneous Partial Order Method

Let $V_h, h \in \{1..q\}$ be the q persistent sets in S. Denote $V = \times_{h=1}^{q} V_h$ as the *progress vector space* in S. For each vector $\vec{\mathbf{v}} = (v_1, v_2, \ldots, v_q) \in V$, let $T(\vec{\mathbf{v}})$ be the set of (executable) transitions in $\vec{\mathbf{v}}$, the *execution* of $\vec{\mathbf{v}}$ in S is defined as the sequential execution of the transitions in $T(\vec{\mathbf{v}})$ in arbitrary order. Since q processes are forced to progress at the same time, this approach is called *simultaneous partial order reachability analysis*. Each $\vec{\mathbf{v}} \in V$ is called an *spo-progress vector* in S. For notational convenience, an *spo*-progress vector $\vec{\mathbf{v}} = (v_1, v_2, \ldots, v_q)$ is often rewritten in its *extended form* $(v'_1, v'_2, \ldots, v'_n)$ such that $\forall i : v'_i = v_h$ if $v_h \in E_i$; $v'_i = \lambda$ otherwise. In the rest of this section, we use the extended form for an *spo*-progress vector, unless otherwise stated.

Definition 3.3 Let S' be the resulting state by the execution of $\vec{\mathbf{v}}$ in S. S' is said to be *directly spo-reachable* from S, denoted as $S \mapsto_{spo} S'$. Let $\mapsto^*_{spo}$ be the reflexive and transitive closure of $\mapsto_{spo}$. S' is *spo-reachable* from S iff $S \mapsto^*_{spo} S'$. When $S = S^0$, S' is said to be *spo-reachable*. The set of *spo*-reachable states is denoted as $\mathbf{R}_{spo}$

Note that when $q = 1$, $V = V_1 = E$ in S. In this case, *spo*-progress in S degenerates into conventional state exploration in S. The *spo*-reachability of s_i, (s_i, m, j), and $\mathcal{C}_I$ can also be defined accordingly.

Suppose $S \mapsto^*_{spo} S'$. An *spo-execution sequence* from S to S' is a sequence $se = X^0 \xrightarrow{\vec{\mathbf{v}}^1} X^1 \xrightarrow{\vec{\mathbf{v}}^2} \cdots \xrightarrow{\vec{\mathbf{v}}^k} X^k, k \geq 0$, such that (i) $X^0 = S$; (ii) $\forall\, l, 1 \leq l \leq k : X^{l-1} \mapsto_{spo} X^l$ via *spo*-progress vector $\vec{\mathbf{v}}^l$; and (iii) $X^k = S'$. The length of se is defined as the number of *spo*-progress vectors in se, denoted as $|se| = k$.

Suppose $S \mapsto_{spo} S'$ via $\vec{\mathbf{v}}$. Since the transitions in $T(\vec{\mathbf{v}})$ are independent of each other, S' is *unique* regardless the sequential execution order of the transitions in $T(\vec{\mathbf{v}})$. Hence $\mapsto_{spo}$ is *well-defined*. Inductively, it should be clear that if $S \mapsto^*_{spo} S'$, then S' is also unique. As a result, $\mapsto^*_{spo}$ is also well-defined.

Lemma 3.5 Suppose $S \mapsto^* S'$ via $\{e_1, e_2, \ldots, e_n\}$. Then $\exists \vec{\mathbf{v}} \in V(S) : \forall i : (v_i \neq \lambda) \Rightarrow (v_i = e_i(S))$ if (i) $\forall i : e_i(S) \neq \lambda$, or (ii) $\exists j : e_j(S) \neq \lambda$ and $\forall i : (e_i(S) = \lambda) \Rightarrow (E_i(S) = \emptyset)$.

Clearly, not every reachable state is *spo*-reachable. However, given a local execution sequence $\{e_1, e_2, \ldots, e_n\}$ for a reachable state S, by Lemma 3.5, we can construct from $\{e_1, e_2, \ldots, e_n\}$ an *spo*-execution sequence $spe = X^0 \xrightarrow{\vec{\mathbf{v}}^1} X^1 \xrightarrow{\vec{\mathbf{v}}^2} \cdots \xrightarrow{\vec{\mathbf{v}}^k} X^k, k \geq 0$, such that (i) $X^0 = S^0$; (ii) $\forall l, 1 \leq l \leq k : X^{l-1} \mapsto_{spo} X^l$ via *spo*-progress vector $\vec{\mathbf{v}}^l$ in X^{l-1} such that $\forall i : (v^l_i \neq \lambda) \Rightarrow (v^l_i = e_i(S))$; and (iii) there is no $\vec{\mathbf{v}} \in V(X^k)$ satisfying $\forall i : (v_i \neq \lambda) \Rightarrow (v_i = e_i(S))$. It can be shown that X^k is unique and is "closest to" S w.r.t $\{e_1, e_2, \ldots, e_n\}$. X^k is called the *spo-precursor* of S w.r.t $\{e_1, e_2, \ldots, e_n\}$, denoted as $sp = (< s^{sp}_i >, < c^{sp}_{ij} >)$. spe is called a *partial spo*-execution sequence for S w.r.t $\{e_1, e_2, \ldots, e_n\}$.

Lemma 3.6 Given a reachable state S, let sp be the *spo*-precursor for S w.r.t a local execution sequence set $\{e_1, e_2, \ldots, e_n\}$ for S. Then $\exists i : e_i(sp) = \lambda$. Furthermore, if $sp \neq S$, then $\exists j : e_j(sp) \neq \lambda$ and $sp \mapsto^* S$ via the remaining transitions in $\{e_1, e_2, \ldots, e_n\}$.

Lemma 3.7 Each deadlock state is *spo*-reachable.

For detection of logical errors other than deadlocks, the *spo*-precursor construction is usually not sufficient. As was the case of fair reachability analysis for cyclic protocols [8], it is sometimes necessary to continue the construction along the remaining transitions in $\{e_1, e_2, \ldots, e_n\}$ beyond the *spo*-precursor. The following two lemmas concern the extended construction[2]. We point out that Lemma 3.9 is crucial to showing that detection of blocking channels are decidable within $\mathbf{R}_{spo}$.

Lemma 3.8 Suppose $S \mapsto^* S'$ via $\{e_1, e_2, \ldots, e_n\}$ such that $\exists i : e_i(S) \neq \lambda$. Then $\exists \vec{\mathbf{v}} \in V(S) : \forall j : ((v_j \neq \lambda) \wedge (e_j(S) \in E_j(S))) \Rightarrow (v_j = e_j(S))$. Furthermore, assume $S \mapsto_{spo} X$ via $\vec{\mathbf{v}}$, then there exists an X' such that $X \mapsto^* X'$ via the remaining transitions in $\{e_1, e_2, \ldots, e_n\}$ from X and $S' \mapsto^* X'$ via those transitions in $\vec{\mathbf{v}}$ that are not from $\{e_1, e_2, \ldots, e_n\}$.

Lemma 3.9 Given two states S and S', if $S \in \mathbf{R}_{spo}$ and $S \mapsto^* S'$, then $\exists S'' \in \mathbf{R}_{spo} : (S' \mapsto^* S'') \wedge (S \mapsto^*_{spo} S'')$.

The following two lemmas state that the reachability of s_i and (s_i, m, j) are preserved in $\mathbf{R}_{spo}$, respectively.

Lemma 3.10 Given a state S and a local state s'_i, s'_i is reachable from S iff it is *spo*-reachable from S.

[2]Similar results were also reported in [12].

The fault coverage of $\mathbf{R}_{spo}$ is summarized in the following theorem. A couple of remarks are in order here. First, not all the nonexecutable transitions can be detected using this method since the reachability of (s_i, m, j) is not preserved in $\mathbf{R}_{spo}$.[3] Second, $\mathbf{R}_{spo}$ by itself is not sufficient to detect all the channel overflows, as was the case for unboundedness detection in fair reachability analysis for cyclic protocols [8]. The main reason is that the sending transitions of P_i over channel C_{ij} may always pair up with the receiving transitions in P_j. For the same reason, simultaneous partial order reachability analysis can also handle those unbounded protocols whose $\mathbf{R}_{spo}$'s are finite. However, it is not clear how unboundedness detection can be conducted via the finite extension of $\mathbf{R}_{spo}$.

Theorem 3.2 Detection of the following logical errors are decidable within $\mathbf{R}_{spo}$: (1) deadlock; (2) indefiniteness; and (3) blocking channel.

3.3 Discussion

From the above discussion, we know that either method has its strengths and weaknesses in terms of logical error detection, but neither can cover all the logical errors for bounded multi-process protocols. For complete analysis, we should combine these two methods. Given a protocol P, we may first use simultaneous partial order reachability analysis to detect all the deadlock and indefinite states, and all the blocking channels in P; then use maximal partial order reachability analysis to detect all the channel overflows and nonexecutable transitions in P. Or we can use maximal partial order analysis first, followed by simultaneous partial order analysis. The trust is that whatever order we choose to apply these two methods, they, when used together, guarantee the complete logical error coverage of the protocol under study.

Our maximal partial order reachability analysis technique can be regarded as a generalization of maximal reachability analysis for protocols with $n = 2$ processes [6] to protocols with $n \geq 2$ processes. Our state exploration algorithm is also simpler than the one in [6]. However, the fault coverage capability is decreased as we move from $n = 2$ to $n \geq 2$, primarily due to the fact each process can have more than one input channel in a protocol.

Our simultaneous partial order reachability analysis technique is similar to fair reachability analysis in that it also forces more than one process to progress in the current state that has more than one persistent set. However, it is different from fair reachability analysis in the following ways. First, a fair progress vector may contain transitions that are not executable but are "enabled" by other transitions in the vector [8, 10]. Second, a fair progress vector may also contain two transitions of the same persistent set. Third, this technique cannot guarantee that each state maintains a certain channel invariant. For the class of multi-cyclic protocols, fair reachability analysis generally provides better state reduction since each fair reachable state satisfies the "equal channel length property" w.r.t each component cyclic protocols [10]. However, since it is unlikely that fair reachability can be generalized to handle protocols beyond multi-cyclic protocols [10], one can regard our simultaneous partial order reachability analysis technique as a counterpart state reduction method for general multi-process protocols with arbitrary communication topologies.

In [11, 12], Özdemir and Ural also proposed a *simultaneous reachability analysis* method for general multi-process protocols. Their approach is similar to our simultaneous partial order method in that it forces more than one process to make progress at the same time. In addition, they also take into account the potentially executable transitions in computing the set of progress vectors. Moreover, both methods can be used to detect deadlocks. However, there are four subtle differences between these two approaches. First, their approach computes the progress vector space of a state based on the cross product of executable transitions in each processes. Hence, even when a state has only one persistent set, their method may produce progress vectors that allow more than one process to progress. Second, their method can detect all the nonexecutable transitions, while ours cannot. Third, their method can also detect unspecified reception with augmentation. However, it is not clear how their method can be adapted to detect blocking channels in a protocol. Finally, to detect channel overflow, their method also requires augmentation and has to conduct analysis once for each channel; while our maximal partial order analysis, when used as an augmentation to simultaneous partial order analysis, only needs to run once for each process.

4 Conclusion

In this paper, we proposed two state reduction techniques for validation of multi-process protocols modeled as CFSM's, based on the partial order method developed for the Petri nets and the LTS models. The first technique generalized the maximal progress state exploration for protocols with $n = 2$ processes to protocols with $n \geq 2$; while the other can serve as a counterpart of fair reachability analysis for protocols

[3] As pointed out by one of the referees.

with arbitrary communication topologies. We showed that these two techniques, when used together, provide a complete logical error coverage for bounded multi-process protocols. This study shows that techniques developed in one model can be adapted to another model without too much difficulty.

There are several issues remained to be solved. First, we would like to further investigate the relationship between maximal and simultaneous partial order reachability analysis. It would be nice if we could establish similar fault coverage results as those for fair reachability analysis for cyclic protocols [8]. Second, we need to conduct experiments in order to fully assess the state reduction capability of simultaneous partial order reachability analysis and simultaneous reachability analysis[12]. Third, we are currently investigating how simultaneous partial order reachability analysis can be applied to the Petri nets and the LTS models.

Acknowledgement

The first author would like to thank Gerard J. Holzmann and Doron Peled for several interesting discussions on partial order reduction methods, Hasan Ural for many suggestions on the early draft of this paper, and the anonymous referees for their comments that greatly improve the quality of the paper.

References

[1] D. Brand and P. Zafiropulo, "On Communicating Finite-State Machines," Journal of ACM, Vol. 30, No. 2, April 1983, pp. 323–342.

[2] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis, "Memory-Efficient Algorithms for the Verification of Temporal Properties," Formal Methods in System Design, Vol. 1, 1992, pp. 275–288.

[3] P. Godefroid and P. Wolper, "Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties," Formal Methods in System Design, Vol. 2, No. 2, 1993, pp. 149–164.

[4] P. Godefroid and P. Wolper, "A Partial Approach to Model Checking," Information and Computation, Vol. 110, No. 2, 1994, pp. 305–326.

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

[6] M.G. Gouda and Y.T. Yu, "Protocol Validation by Maximal Progress State Exploration," IEEE Transactions on Communications, Vol. COM-32, No. 1, 1984, pp. 94–97.

[7] G.J. Holzmann and D. Peled, "An Improvement in Formal Verification," FORTE'94, Bern, Switzerland, Oct. 4–7, 1994.

[8] H. Liu and R.E. Miller, "Generalized Fair Reachability Analysis for Cyclic Protocols," IEEE/ACM Transactions on Networking, Vol. 4, No. 2, April 1996, pp. 192–204.

[9] H. Liu and R.E. Miller, "Generalized Fair Reachability Analysis for Cyclic Protocols with Nondeterministic and Internal Transitions," ICNP'95, Tokyo, Nov. 7–10, 1995, pp. 6–13.

[10] H. Liu, R.E. Miller, H.v.d. Schoot, and H. Ural, "Deadlock Detection by Fair Reachability Analysis: from Cyclic to Multi-Cyclic Protocols (and beyond?)," ICDCS'96, Hong Kong, May 27–30, 1996, pp. 605–612.

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

[12] 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.

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

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

[15] A. Valmari, "Stubborn Sets for Reduced State Space Generation," 10th International Conference on Application and Theory of Petri Nets," Vol. 2, 1989, pp. 1–22.

[16] A. Valmari, "On-the-fly Verification of Stubborn Sets," CAV'93.

[17] M. Vardi and P. Wolper, "Reasoning about Infinite Computations," Information and Computation, Vol. 115, 1994, pp. 1–37.

[18] A. Thayse (Ed.), "From Modal Logic to Deductive Databases," Chapter 4, "Temporal Logic," John Wiley & Sons, 1989.