1

Generalized Fair Reachability Analysis for Cyclic Protocols: Part 1 *

Hong Liu and Raymond E. Miller

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

Abstract

In this paper, the notion of fair reachability is generalized to cyclic protocols with $n > 2$ communicating finite state machines. An equivalence is established between the set of fair reachable states and the set of reachable states with equal channel length. As a result, *deadlock detection* is decidable for cyclic protocols with finite fair reachability graphs. The concept of *simultaneous unboundedness* is defined and the lack of it is shown to be a necessary and sufficient condition for a cyclic protocol to have a finite fair reachability graph. For the first time, we are able to exactly characterize the class of protocols whose fair reachability graphs are finite. As far as decidability of deadlock detection is concerned, our result extends the class of cyclic protocols studied by Peng & Purushothaman, and complements the one investigated by Pachl. More importantly, our decision procedure is much more straightforward and efficient, as compared to Pachl's and the one by Peng & Purushothaman. In this respect, we have improved the complexity of deadlock detection for the class of cyclic protocols with finite fair reachability graphs. To further demonstrate the strength of generalized fair reachability analysis, we also show that *livelock detection* is decidable for the class of cyclic protocols with finite fair reachability graphs.

Keyword Codes: C.2.2; D.2.1; D.2.4
Keywords: Network Protocols; Requirements/Specifications; Program Verification

1 Introduction

The *communicating finite state machine model* is one of the most widely used formal models for protocol specification and verification [1]. In this model, a protocol is specified as a set of finite state machines exchanging messages via $FIFO$ channels. A simple *state space exploration technique*, also known as *reachability analysis* is used to systematically generate the entire global state space reachable from the initial global state. Protocol validation is done by checking each reachable global state against progress criteria in terms of deadlock, unspecified reception

*Research reported in this paper was supported by NASA Center of Excellence in Space Data and Information Sciences Under USRA Subcontract No. 550-66.

S. T. Vuong et al. (eds.), *Protocol Specification, Testing and Verification XIV*

and boundedness. With this simple model and straightforward verification technique, some real world protocols have been successfully modeled and validated. However, there are two problems concerning this model that hampers its practical usefulness to industrial strength applications. First, progress properties are in general undecidable for protocols modeled as communicating finite state machines [1], in particular, exhaustive state enumeration is only feasible for bounded protocols. Second, even with bounded protocols, reachability analysis suffers from the *state explosion problem.* Most real world protocols are large and complex, with tens of thousands of global states. In this case, even though reachability graphs are finite, the analysis becomes very inefficient due to the brute-force state exploration.

Much research has been devoted to looking for classes of protocols whose progress properties are decidable and devising techniques to limit state explosion during analysis. As a result, many techniques have been proposed. These methods differ in the classes of protocols they can handle, the ease of being automated, and the overhead they incur. For a survey of these methods, please refer to [19].

One of the proposed improved techniques is called *fair reachability* [18, 10], where each machine is forced to make a move whenever possible during state exploration. In fair reachability, global state exploration is reduced by avoiding redundant exploration of equivalent interleaving execution sequences during the analysis. This technique has been shown effective in validation for protocols modeled as two communicating finite state machines [18, 10]. However, the concept of fair reachability and its effectiveness for general protocols with more than two machines has not been studied. To fill this gap, we investigate the generalization of this technique to cyclic protocols. Through the study, its effectiveness for cyclic protocol validation is shown.

The rest of the paper is organized as follows: In section 2, we briefly review previous research on fair reachability analysis and highlight our results presented in this paper. Then, the communicating finite state machine model is formally introduced in the following section. In section 4, we generalize fair reachability to cyclic protocols with $n \geq 2$ communicating finite state machines. Based on this, we present a sufficient condition for a cyclic protocol to have a finite fair reachability graph, which is a generalization of the one shown in [8]. To build a theoretical foundation for generalized fair reachability analysis, we study the characterization of fair reachable state space in section 5. From the investigation, we obtain the key results of this paper: an equivalence between the set of fair reachable states and the set of reachable states with equal channel length, and a necessary and sufficient condition for a cyclic protocol to have a finite fair reachability graph. To demonstrate the strength of our approach, we show in section 6 that both deadlock detection and livelock detection are decidable for the class of cyclic protocols with finite fair reachability graphs. We conclude the paper with open problems in section 7.

2 Previous Work

Fair reachability analysis was proposed as a strategy for reducing state explosion during validation of protocols modeled as two communicating finite state machines. Rubin and West first observed the redundancy of state exploration in reachability analysis due to equivalent sequences of interleaving transitions [18]. Based on this observation, they proposed a canonical sequence technique that forces the two machines to progress at the same speed during state exploration. They reported a large percentage reduction in state generation when this technique was incorporated into reachability analysis. For protocols whose reachability graph is finite, they proved that both the deadlock detection and unspecified reception detection problems are decidable. In [10], Gouda and Han named this technique *fair reachability analysis.* The reachability graph thus generated is termed a *fair reachability graph.* They showed that for protocols

whose fair reachability graph is finite, the boundedness detection problem is also decidable. A sufficient condition for protocols to have a finite reachability graph was also established in [8]; namely, if a protocol has one of the two channels bounded, then its fair reachability graph is finite. Therefore, for $n = 2$, the detection of deadlock, unspecified reception, and unboundedness are all decidable for the class of protocols with at least one bounded channel. Recently, Cacciari and Rafiq extended the above idea to protocols with "internal" transitions, where an internal transition of a process is defined as a transition that changes the local state of the process but does not change the content of any channel associated with that process [2, 3]. They called their technique *reduced reachability analysis*. In [2], they showed that using this technique, both deadlock and unspecified reception, among other properties, are decidable for protocols whose *reduced reachability graphs* are finite. In [3], they showed that it is undecidable whether a protocol has a finite reduced reachability graph. However, it is not clear what class of protocols are amendable for reduced reachability analysis [3].

One important aspect about fair reachability analysis is that in each fair reachable state, the length of each channel is equal [18, 8]. We call this property the *equal channel length* property of fair reachability analysis. On one hand, reduced reachability analysis by Cacciari and Rafiq resembles fair reachability analysis in that it forces two machines to move at the same time if the *parallelwise* condition is satisfied [2]. On the other hand, if the parallelwise condition is not satisfied, only one machine is allowed to move at one time. As a result, the set of reduced reachable states no longer has the equal channel property. This is, we feel, one of the major reasons that makes it more difficult to find a (sufficient) condition for the class of protocols with finite reduced reachability graphs.

Fair reachability analysis is of importance not only because it can reduce the number of global states explored, but also because it has the capability to handle some protocols with unbounded channels. Although in [18], the authors claimed to extend this technique to protocols with $n > 2$ communicating finite state machines, so far, we have not seen any follow-up reports on this issue.

It should be noted that for bounded protocols, the classic reachability technique can be used for protocols with $n > 2$ communicating finite state machines. But research in analysis of protocols with unbounded channels has been mostly limited to only cyclic protocols [13, 14, 16, 17]. Jan Pachl is probably the first person who formalized and investigated the class of cyclic protocols. His method is based on the channel expression concept [13, 14]. In [14], he showed that the deadlock detection problem is decidable for the class of cyclic protocols with recognizable channel expressions. But many of his important results on cyclic protocols are contained in his unpublished research report [13], in which he showed that the deadlock detection problem and the unspecified reception detection problem are decidable for the class of cyclic protocols with one channel whose channel expressions are regular. However, he wrote in [13] that the decision procedure is hopelessly inefficient for any practical purpose. In [16], Peng and Purushothaman showed that for the class of cyclic protocols with exactly one unbounded channel, the deadlock detection problem is decidable. Their method relied on the construction of a "stable cover set" and the construction of a finite automaton to recognize the stable cover set. It is not clear, however, that this procedure can be automated efficiently. In [17], they proposed a data flow approach to analyzing deadlock and unspecified reception for a protocol with $n \geq 2$ machines by computing a superset of the set of reachable states as an approximated solution for a set of data flow equations. If there are no deadlock or unspecified reception states in the superset, then the protocol in question has no deadlock or unspecified reception. However, if there is a deadlock or unspecified reception state in the superset, then the protocol under analysis "might" have a deadlock or unspecified reception, depending on whether such a

state is indeed reachable. Thus, while this approach works for general protocols, the result of the analysis is incomplete. It is unknown for what class of protocols the data flow analysis can yield an exact solution. Furthermore, this approach also suffers from state explosion, as stated by the authors in [17]. Therefore, for the analysis of cyclic protocols with $n > 2$ communicating finite state machines, only the decidability aspect has been studied. The complexity of decision procedures has been largely ignored. For practical analysis, it is highly desirable that the decision procedure be efficient. Moreover, all these techniques proposed for cyclic protocol validation analyze global states from the channel language viewpoint [12]. Reachability analysis, which has been a main focus in the analysis of protocols with two machines, has not been integrated into any of these approaches at all. As a matter of fact, it seems that there is a gap between protocols with two machines and protocols with more than two machines. Most of the methods, if not all, that have been proposed for the two machine case have not yet been carried over to the $n > 2$ case.

In this paper, we bridge this gap by looking into the possibility of applying the fair reachability technique to progress analysis for cyclic protocols with $n > 2$ communicating finite state machines. This includes some new results. Our contributions in this paper are summarized as follows: (1) Fair reachability is formalized in terms of synchronization and concurrency, providing a deeper insight into the interactions among processes. (2) An equivalence is established between the set of fair reachable states and the set of reachable states with equal channel length. As a result, deadlock detection is decidable for the class of cyclic protocols whose fair reachability graphs are finite. (3) A necessary and sufficient condition is presented for cyclic protocols to have finite fair reachability graphs. This condition ensures that for the class of cyclic protocols whose channels are *not simultaneously unbounded*, the deadlock detection problem is decidable. For the first time, the class of cyclic protocols with finite fair reachability graphs can now be exactly characterized. (4) For completeness, we also show that it is undecidable whether a cyclic protocol has a finite fair reachability graph. (5) Regarding the class of cyclic protocols whose deadlock detection is decidable, for $n = 2$, our result properly includes the one studied in [18, 10]; for $n > 2$, our result properly contains the one examined in [16] and complements the one investigated in [13, 14]. More importantly, our decision procedure is much more straightforward and efficient for practical analysis, which was lacking in both [16] and [13, 14]. (6) To further demonstrate the power of our generalized fair reachability analysis technique, we prove that livelock detection is also decidable for the class of cyclic protocols with finite fair reachability graphs, an easy generalization from the one established for $n = 2$ in [8]

Generalized fair reachability analysis for cyclic protocols was first reported in [5], along with the decidability result of deadlock detection for the class of cyclic protocols with finite fair reachability graphs. Since then, the fair reachability notion has been revised to achieve further state reduction and allow for easier proofs. Most importantly, we have discovered a necessary and sufficient condition for the class of cyclic protocols with finite fair reachability graphs, and proved the undecidability of this condition, a key contribution to the study of cyclic protocols.

It should be clear that in this paper, we only study detection of deadlocks and livelocks in cyclic protocols using generalized fair reachability analysis. For detection of other logical errors in cyclic protocols, *pure* fair reachability analysis is not sufficient, as will be addressed in another paper [7].

3 Communicating Finite State Machines

In this section, we briefly introduce the communicating finite state machine model. Due to space limitations, some of the common definitions in the model are omitted. For a complete treatment of the model, please refer to [1, 4] and the full version of this paper [6].

Notation: (1) We use $\cdot$ to denote concatenation. Given a set M, M^* denotes the reflexive and transitive closure of M under concatenation. $|M|$ denotes the cardinality of set M. For any $Y \in M^*$, $|Y|$ denotes the length of Y. We use ϵ to denote an empty string. By definition, $|\epsilon| = 0$. (2) We define two operators, $\oplus$ and $\ominus$. Given n, for any $1 \leq i \leq n$, $0 \leq j < n$, $i \oplus j = i + j$ if $i + j \leq n$ else $i \oplus j = (i + j) \bmod n$; $i \ominus j = i - j$ if $i > j$ else $i \ominus j = i - j + n$, where mod stands for the modulo operation. (3) We define an *interval* $[i..j]$ for an ordered set of at most n consecutive integers $i, i \oplus 1, \ldots, i \oplus k = j$, where $1 \leq i \leq n \wedge 0 \leq k < n$. The corresponding (unordered) set is denoted as $\{i..j\}$. The cardinality of $[i..j]$, denoted as $|[i..j]|$, is defined as $k + 1$. We define $[i'..j'] \subseteq [i..j]$ if and only if $\{i'..j'\} \subseteq \{i..j\}$ and $[i'..j'] \subset [i..j]$ if and only if $\{i'..j'\} \subset \{i..j\}$. Also we denote $P_{[i..j]}$ as the set of processes indexed by $[i..j]$, called a *process interval.* (4) We designate n as the number of processes in a protocol. Unless otherwise specified, we assume $n \geq 2$ and let i, j range over $[1..n]$.

In the communicating finite state machine model, a protocol is specified as a set of n finite state machines, where each machine communicates with other machines via $FIFO$ channels.

Definition 3.1 A *protocol* $P = (P_1, P_2, \ldots, P_n), n \geq 2$, is a four-tuple $(\mathbf{S}, \mathbf{M}, \mathbf{O}, \tau)$, where

- Each P_i is a process represented as a finite state machine.
- $\mathbf{S} = (\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n)$, where $\mathbf{S}_i$ represents the finite set of local states of process P_i.
- $\mathbf{M} = (M_1, M_2, \ldots, M_n)$, where $M_i = (M_{1i}, M_{2i}, \ldots, M_{i-1i}, M_{i+1i}, \ldots, M_{ni}), i \in [1..n]$. Each $M_{ji}, j \neq i$, represents the set of messages that can be sent from P_j to P_i.
- $\mathbf{O} = (s_1^0, s_2^0, \ldots, s_n^0)$, where $s_i^0 \in \mathbf{S}_i$ is the *initial local state* for P_i.
- τ, a partially defined transition function: $\bigcup_{i=1}^{n}(\mathbf{S}_i \times \tilde{M}_i \rightarrow \mathbf{S}_i)$, where $\tilde{M}_i = (\bigcup_{j \neq i}\{-m | m \in M_{ij}\}) \cup (\bigcup_{j \neq i}\{+m | m \in M_{ji}\})$.

A channel $C_{ij}, i \neq j$, is modeled as a $FIFO$ queue connecting P_i to P_j. The contents of C_{ij} is denoted as c_{ij}, which is a sequence of messages $m \in M_{ij}$. If C_{ij} is empty, $c_{ij} = \epsilon$.

For each P_i, a transition defined at local state $s_i \in S_i$ is denoted as $\tau(s_i, \sigma)$, where $\sigma \in \tilde{M}_i$. When $\sigma = -m$, it is a *sending* transition, representing the transmission of message m by P_i. When $\sigma = +m$, it is a *receiving* transition, representing the reception of message m by P_i. We use the notation $\tau_i = \tau(s_i, \sigma)$ to give a name τ_i for this transition, and use the notation $s_i' = \tau(s_i, \sigma)$ to denote that s_i' is the local state resulting from the execution of the transition. A local state s_i in P_i is a *receiving* local state if and only if all transitions defined in s_i are receiving transitions. By definition, each P_i is deterministic but partially defined.

Given a protocol $P = (P_1, P_2, \ldots, P_n)$, a *communication topology graph* of P is a directed graph such that each node of the graph is labeled as one process P_i, and there is an directed edge from node P_i to node $P_j, i \neq j$, if and only if there is a $FIFO$ channel C_{ij} from process P_i to process P_j. A protocol is *cyclic* if and only if its communication topology graph is a ring in which there is a directed edge from each node P_i to $P_{i \oplus 1}$. Thus, in a cyclic protocol, each P_i has only one input channel $C_{i \ominus 1 i}$ and only one output channel $C_{i i \oplus 1}$.

From now on, we are dealing with cyclic protocols. Although concepts and notations introduced in the remainder of this section are presented in the context of cyclic protocols, they can be adapted to general protocols without significant changes. However, for results established later in this paper, it should be clear that they apply to cyclic protocols only.

Given a cyclic protocol $P = (P_1, P_2, \ldots, P_n)$, a *global state* S is represented as a $2n$-tuple $(s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots c_{n-1n})$, where s_i is the local state of P_i in global state S, and $c_{i \ominus 1 i}$ is the content of channel $C_{i \ominus 1 i}$ in global state S. In particular, the *initial global state* S^0 is denoted as $(s_1^0, s_2^0, \ldots, s_n^0, \epsilon, \ldots, \epsilon)$.

For the sake of brevity, a global state is called a state for short. As a convention, we use capital letters S, X to denote a state and small letters s_i, x_i to denote a local state of P_i.

Definition 3.2 Given two states $S = (s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots, c_{n-1n})$ and $S' = (s'_1, s'_2, \ldots, s'_n, c'_{n1}, c'_{12}, \ldots, c'_{n-1n})$. S' is *directly reachable* from S, denoted as $S \mapsto S'$, if and only if $\exists i \in [1..n]$ such that the elements of S' can be derived from S by executing one of the following transitions: (1) $s'_i = \tau(s_i, -m)$ and $c'_{ii\oplus 1} = c_{ii\oplus 1} \cdot m$. (2) $s'_i = \tau(s_i, +m)$ and $c_{i\ominus 1i} = m \cdot c'_{i\ominus 1i}$ 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 $\mapsto$. Given two states S and S', S' is *reachable* from S if and only if $S \mapsto^* S'$. In this case, local state s'_i is also said to be reachable from s_i in P_i. When $S = S^0$, we say S' is a *reachable state.* The set of reachable states is called the *reachability state space.*

For protocol validation, we classify reachable states according to some general error conditions. Given a reachable state S, S is a *receiving* state if and only if all local states in S are receiving local states. S is a *deadlock* state if and only if it is a receiving state and all the channels in S are empty. Similarly, we can define *unspecified reception*, *nonexecutable transition*, and *channel/protocol unboundedness* for reachable states in a cyclic protocol [7].

Deadlock, unspecified reception, nonexecutable transition, and channel unboundedness are called *logical errors* in a protocol. A protocol is said to have the *required progress properties* if it does not contain any unspecified receptions or deadlocks. A protocol is said to be *logically correct* if and only if it does not have any logical errors.

Logical correctness of a protocol P can be determined by constructing the *reachability graph* for P and checking each node for logical errors. This state exploration technique is called *reachability analysis.* Obviously, in order for this technique to be useful, the reachability graph must be finite. In fact, it was shown that for protocols with $n = 2$, none of the logical errors are decidable [1]. Therefore, logical correctness is not decidable for cyclic protocols. For completeness of this paper, we present this general result as a theorem below.

Theorem 3.1 For cyclic protocols, detection of deadlock, unspecified reception, nonexecutable transition, and channel/protocol unboundedness are all undecidable.

4 Generalized Fair Reachability

In this section, we extend the fair reachability notion for cyclic protocols with $n \geq 2$ machines. The concepts of concurrency and synchronization are described to provide better understanding of the interactions among processes and both are incorporated into the formation of fair progress vectors. With that, the generalized fair reachability relation is formulated. Based on this relation, we are able to show that all fair reachable states are reachable states with equal channel length. A sufficient condition is established for a cyclic protocol to have a finite fair reachability graph. This condition is a generalization of the one in [8]. Due to space limitations, lemmas and theorems presented in the rest of the paper are stated without proof. Please refer to the full paper [6] for details.

4.1 Fair Progress Vector Space

Given a cyclic protocol $P = (P_1, P_2, \ldots, P_n)$. Let $S = (s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots, c_{n-1n})$ be a state and $\tau_i = \tau(s_i, \sigma)$ be a transition defined at local state s_i. τ_i is *executable* at s_i in S if and only if $(\sigma = -m) \vee ((\sigma = +m) \wedge (c_{i\ominus 1i} = m \cdot c'_{i\ominus 1i}))$, where $(m \in M_{i\ominus 1i}) \wedge (c'_{i\ominus 1i} \in M^*_{i\ominus 1i})$. τ_i is *enabled* at s_i in S if and only if $(\sigma = +m) \wedge (c_{i\ominus 1i} = \epsilon)$ and $\tau(s_{i\ominus 1}, -m)$ is defined at local state $s_{i\ominus 1}$ for some $m \in M_{i\ominus 1i}$. Note that in S, s_i can have more than one enabled transitions.

The set of all executable transitions at s_i in S is denoted as $E_i(S) = E_i^-(S) + E_i^+(S)$, where $E_i^-(S)$ stands for the set of all executable sending transitions at s_i in S, while $E_i^+(S)$ for the

set of all executable receiving transitions at s_i in S. The set of all enabled transitions at s_i in S is denoted as $E_i^{++}(S)$. When S is given and no confusion arises, we drop S from the above notations. By definition, for any state S of P, the following formula is true: $\forall i \in [1..n] : (0 \leq |E_i^+| \leq 1) \wedge (E_i^+ = \emptyset \vee E_i^{++} = \emptyset)$.

Given a state S, a *pseudo transition vector* in S is a n-tuple $\vec{t} = (t_1, t_2, \ldots, t_n)$ such that $\forall i \in [1..n] : t_i \in E_i \cup E_i^{++} \cup \{\lambda\}$, where λ stands for a *null* transition in P_i. Denote $TV = \{\vec{t} = (t_1, t_2, \ldots, t_n) | \forall i \in [1..n] : t_i \in E_i \cup E_i^{++}$ if $E_i \cup E_i^{++} \neq \emptyset$; $t_i = \lambda$ otherwise$\}$. TV is a subset of all the pseudo transition vectors in state S. For each pseudo transition vector $\vec{t} \in TV$, we compute a pseudo transition vector $\vec{v} = (v_1, v_2, \ldots, v_n)$ from $\vec{t}$ according to one of the following three cases:

(1) $\vec{t} \in (\mathbf{X}_{i=1}^n E_i^-) \cup (\mathbf{X}_{i=1}^n E_i^+)$. In this case, set $\vec{v} = \vec{t}$. $\vec{v}$ is called a *concurrency vector* in S.

(2) $\exists j : (t_j \in E_j^-) \wedge (t_{j\oplus 1} \in E_{j\oplus 1}^+ \cup E_{j\oplus 1}^{++})$. In this case, $\forall i \in [1..n]$: if $((t_i \in E_i^-) \wedge (t_{i\oplus 1} \in E_{i\oplus 1}^+ \cup E_{i\oplus 1}^{++})) \vee ((t_{i\ominus 1} \in E_{i\ominus 1}^-) \wedge (t_i \in E_i^+ \cup E_i^{++}))$, then set $v_i = t_i$; else set $v_i = \lambda$. Each such pair $(v_i, v_{i\oplus 1})$ is called a *send-receive pair* in $\vec{v}$. $\vec{v}$ is called a *synchronization vector* in S.

(3) Neither condition for Case 1 nor condition for Case 2 holds. In this case, set each $v_i = \lambda$. The resulting pseudo transition vector is called the *null vector*, indicating no progress from any process P_i.

For each pseudo transition vector $\vec{v}$ thus computed, $\vec{v}$ is a *fair progress vector* in S if and only if it is either a concurrency vector or a synchronization vector. Denote $V_c(S)$ $(V_s(S))$ as the set of concurrency (synchronization) vectors in S. Let $V(S) = V_c(S) \cup V_s(S)$. $V(S)$ is called the *fair progress vector space* in S. If $V(S) = \emptyset$, then S is a *dead end state*; otherwise, it is not a dead end state. When S is given and no confusion arises, we drop S from the above notations.

By definition, we have $V_c \cap V_s = \emptyset$ and $V \subseteq \mathbf{X}_{i=1}^n (E_i \cup E_i^{++} \cup \{\lambda\})$. Hence, V is finite and can be effectively computed.

Note that given S, if $\forall i \in [1..n] : E_i \neq \emptyset$, then any pseudo transition vector $\vec{t} \in TV$ can produce a fair progress vector. Thus, if S is a dead end state, then $\exists i : E_i \cup E_i^{++} = \emptyset$. In this case, if $c_{i\ominus 1 i} \neq \epsilon$, then the dead end state S must have an unspecified reception at s_i.

For a dead end state S, it is "dead" in the sense that no fair progress vectors can be derived from S. However, S might still have some transition executable at some local state s_i. The notion of *extendibility* is captured by the following definition.

A state S is an *extendible* state if and only if $\exists i \in [1..n] : E_i \neq \emptyset$. In this case, S is extendible in P_i. A state S is extendible in $P_{[i..j]}$ for some interval $[i..j]$ if and only if $\forall k \in [i..j] : S$ is extendible in P_k. S is *maximal extendible* in $P_{[i..j]}$ if and only if S is extendible in $P_{[i..j]}$ and there is no interval $[i'..j'] \supset [i..j]$ such that S is extendible in $P_{[i'..j']}$. Thus, a dead end state S can be an extendible state. Moreover, if S is an extendible dead end state, then S must be maximal extendible in some process interval $P_{[i..j]}$.

Let's study in more detail the relationship among transitions in each process in a state S from the fair progress vector generation point of view. Let $[i..j]$ be an interval. A vector $\vec{u}_{[i..j]} = (u_i, u_{i\oplus 1}, \ldots, u_j)$ is called a *transition vector* in S if and only if $\forall k \in [i..j] : (E_k \neq \emptyset) \wedge (u_k \in E_k)$. When $[i..j] = [1..n]$, $\vec{u}_{[i..j]}$ is simplified as $\vec{u}$. Therefore, a pseudo transition vector $\vec{t}$ is a transition vector if and only if $\forall i \in [1..n] : (E_i \neq \emptyset) \wedge (t_i \in E_i)$.

Let $\vec{u}_{[i..j]}$ be a transition vector in S. $\vec{u}_{[i..j]}$ is an *incompatible* transition vector in S if and only if no fair progress vector can be derived from pseudo transition vector $\vec{t} = (t_1, t_2, \ldots, t_n)$, where $\forall k \in [1..n] : t_k = u_k$ if $k \in [i..j]$; $t_k = \lambda$ otherwise. $\vec{u}_{[i..j]}$ is a *proper* incompatible transition vector in S if and only if it is an incompatible transition vector in S, S does not have a concurrency vector, $(u_i \in E_i^+) \Rightarrow (E_{i\ominus 1}^- = \emptyset)$, and $(u_j \in E_j^-) \Rightarrow (E_{j\oplus 1}^+ \cup E_{j\oplus 1}^{++} = \emptyset)$. $\vec{u}_{[i..j]}$ is

a *maximal* proper incompatible transition vector in S if and only if it is a proper incompatible transition vector in S and there is no proper incompatible transition vector $\vec{\mathbf{u}}'_{[i'..j']}$ in S such that $\vec{\mathbf{u}}'_{[i'..j']} \supset \vec{\mathbf{u}}_{[i..j]}$, i.e., $([i'..j'] \supset [i..j]) \wedge (\forall k \in [i..j] : u'_k = u_k)$. For notation convenience, we also denote $\vec{\mathbf{u}}'_{[i'..j']}$ as $\vec{\mathbf{u}}_{[i'..j']}$ when $\vec{\mathbf{u}}'_{[i'..j']} \supseteq \vec{\mathbf{u}}_{[i..j]}$.

Lemma 4.1 For any state S, let $\vec{\mathbf{u}}_{[i..j]} = (u_i, u_{i\oplus 1}, \ldots, u_j)$ be an incompatible transition vector in S, $\vec{\mathbf{t}}$ be any pseudo transition vector in S. The following statements hold: (1) If $\vec{\mathbf{t}}$ is a transition vector, then a fair progress vector can be derived from $\vec{\mathbf{t}}$. (2) $1 \leq |[i..j]| < n$. (3) $(u_i \in E_i^+) \vee (u_j \in E_j^-)$. In other words, it is always true that either u_i is a receiving transition or u_j is a sending transition. (4) For any $\vec{\mathbf{u}}_{[i'..j']} \subseteq \vec{\mathbf{u}}_{[i..j]}$, $\vec{\mathbf{u}}_{[i'..j']}$ is also an incompatible transition vector in S. (5) $\vec{\mathbf{u}}_{[i..j]}$ is a maximal proper incompatible transition vector if and only if $(E_{i\ominus 1} = \emptyset) \wedge (E_{j\oplus 1} \cup E_{j\oplus 1}^{++} = \emptyset)$. Note that when $|[i..j]| = n-1$, $i = j \oplus 1$.

4.2 Generalizing Fair Reachability Relation

In this subsection, we generalize the fair reachability notion from (cyclic) protocols with two communicating finite state machines to cyclic protocols with $n > 2$ communicating finite state machines. The validity of this extension is also discussed.

Defintion 4.1 Given a protocol $P = (P_1, P_2, \ldots, P_n)$, for any two global states $S = (s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots, c_{n-1n})$, and $S' = (s'_1, s'_2, \ldots, s'_n, c'_{n1}, c'_{12}, \ldots, c'_{n-1n})$, S' is *directly fair reachable* from S, denoted as $S \mapsto_f S'$, if and only if there exists a fair progress vector $\vec{\mathbf{v}} \in V(S)$ such that the execution of $\vec{\mathbf{v}}$ in S leads the system from global state S to S'. Specifically, there are three cases to consider:

(1) $\vec{\mathbf{v}} \in V_s(S)$. For each send-receive pair $(v_i, v_{i\oplus 1}), i \in [1..n]$, there are two subcases to consider:

(a) $c_{ii\oplus 1} = \epsilon$. Let $v_i = \tau(s_i, -m)$ and $v_{i\oplus 1} = \tau(s_{i\oplus 1}, +m)$ for some $m \in M_{ii\oplus 1}$. Execution of $(v_i, v_{i\oplus 1})$ will cause transition $\tau(s_i, -m)$ to be taken, followed by transition $\tau(s_{i\oplus 1}, +m)$, where $s'_i = \tau(s_i, -m)$ and $s'_{i\oplus 1} = \tau(s_{i\oplus 1}, +m)$.

(b) $c_{ii\oplus 1} \neq \epsilon$. Let $v_i = \tau(s_i, -m)$, $v_{i\oplus 1} = \tau(s_{i\oplus 1}, +m')$, and $c_{ii\oplus 1} = m' \cdot c''_{ii\oplus 1}$ for some $m, m' \in M_{ii\oplus 1}$ and $c''_{ii\oplus 1} \in M^*_{ii\oplus 1}$. Execution of $(v_i, v_{i\oplus 1})$ will cause transitions $\tau(s_i, -m)$ and $\tau(s_{i\oplus 1}, +m')$ to be taken in arbitrary order, where $s'_i = \tau(s_i, -m)$, $s'_{i\oplus 1} = \tau(s_{i\oplus 1}, +m')$, and $c'_{ii\oplus 1} = c''_{ii\oplus 1} \cdot m$.

Except for the elements affected by the transitions applied in each of the send-receive pairs, all other elements of S' remain the same as those in S.

(2) $\vec{\mathbf{v}} \in V_c(S) \wedge (\forall i \in [1..n] : v_i = \tau(s_i, -m_i) \in E_i^-)$, where $\forall i \in [1..n] : m_i \in M_{ii\oplus 1}$. The result of applying $\vec{\mathbf{v}}$ on S is such that $\forall i \in [1..n] : s'_i = \tau(s_i, -m_i)$ and $c'_{ii\oplus 1} = c_{ii\oplus 1} \cdot m_i$.

(3) $\vec{\mathbf{v}} \in V_c(S) \wedge (\forall i \in [1..n] : v_i = \tau(s_i, +m_{i\ominus 1}) \in E_i^+)$, where $\forall i \in [1..n] : m_{i\ominus 1} \in M_{i\ominus 1}$. Assume that before applying $\vec{\mathbf{v}}$, $\forall i \in [1..n] : c_{i\ominus 1i} = m_{i\ominus 1} \cdot c''_{i\ominus 1i}$ for some $c''_{i\ominus 1i} \in M^*_{i\ominus 1i}$. The result of applying $\vec{\mathbf{v}}$ on S is such that $\forall i \in [1..n] : s'_i = \tau(s_i, +m_{i\ominus 1})$ and $c'_{ii\oplus 1} = c''_{ii\oplus 1}$.

Denote $\mapsto_f^*$ as the reflexive and transitive closure of $\mapsto_f$. Given two states S and S', S' is *fair reachable from* S if and only if $S \mapsto_f^* S'$. When $S = S^0$, we say S' is fair reachable. Unless otherwise stated, when we say S' is a fair reachable state, we mean it is fair reachable from S^0.

Given a protocol P, the set of fair reachable states, denoted as $\mathbf{F}$, is called the *fair reachable state space* of P. As is the case for reachable states, we can define logical errors for fair reachable states. Given $S \in \mathbf{F}$, S is a *fair* deadlock state if and only if S is a deadlock state. Similarly, we can define unspecified reception, nonexecutable transition and unbounded channel for S [7].

Since in a fair progress vector, multiple processes can make a move, we want to make sure that such concurrent transition execution is *well-defined* in the sense that any executable

interleaving sequence of these concurrent transitions will lead to the same state. Careful study on the formulation of the synchronization vector and the concurrency vector shows that both do satisfy the above requirement. Therefore, $\mapsto_f$ is well-defined for cyclic protocols. Inductively, the generalized fair reachability relation $\mapsto_f^*$ is also well-defined.

A state $S = (s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots, c_{n-1n})$ is a state with *equal channel length* if and only if $|c_{n1}| = |c_{12}| = \ldots = |c_{n-1n}|$. Note that any deadlock state is a state with equal channel length of zero. Note also that the initial state S^0 is a state with equal channel length of zero. Moreover, any fair progress vector in S^0 maintains the equal channel length property in the resulting state. Using this argument inductively, we arrive at the conclusion that the set of fair reachable states is included in the set of reachable states with equal channel length, as stated in the following theorem.

Theorem 4.1 Any fair reachable state S is a reachable state with equal channel length.

As a result, the set of all fair reachable states $\mathbf{F}$ is closed under application of fair progress vectors from their respective fair progress vector space. In section 5, we will also show that any reachable state with equal channel length is also fair reachable, and it is this result that leads to deadlock detection using fair reachability analysis.

Based on this theorem, we can partition the fair reachable state space $\mathbf{F}$ into subsets by channel length. Let $\mathbf{F}_k, k \geq 0$, be the set of fair reachable states whose channel length is k. Note that the set of fair deadlock states is included in $\mathbf{F}_0$.

Lemma 4.2 Given a fair reachable state space $\mathbf{F}$, the following statments hold: (1) $\forall k, k' : k, k' \geq 0 \wedge k \neq k', \mathbf{F}_k \cap \mathbf{F}_{k'} = \emptyset$. (2) $\mathbf{F} = \bigcup_{k=0}^{\infty} \mathbf{F}_k$. (3) $\forall S \in \mathbf{F}_k, k \geq 0$, if $S \mapsto_f S'$, then $S' \in \mathbf{F}_0 \cup \mathbf{F}_1$ when $k = 0$; $S' \in \mathbf{F}_{k-1} \cup \mathbf{F}_k \cup \mathbf{F}_{k+1}$ otherwise. (4) $\mathbf{F}_k, k \geq 0$ is finite. In fact, $|\mathbf{F}_k| \leq (\prod_{i=1}^{n} |S_i|) * (\prod_{i=1}^{n} |M_{ii\oplus 1}|^k)$. (5) $\mathbf{F}$ is finite if and only if $\exists K : K \geq 0, \mathbf{F}_{K+1} = \emptyset$.

As in reachability analysis, we construct a graph to systematically explore the fair reachability state space of a protocol during validation. Formally, a *fair reachability graph* **FRG** is a directed graph such that each node is labeled with a fair reachable state, and there is a directed edge from a node labeled with S to a node labeled with S' if and only if $S \mapsto_f S'$. In particular, the node labeled with S^0 is called the *initial node* of **FRG**. Therefore, there is a directed path in **FRG** from the node labeled as S to the node labeled as S' if and only if $S \mapsto_f^* S'$. From now on, we will use the term "a fair reachable state" and the term "a node labeled with that state in a **FRG**" interchangeably. We sometimes use $S \in$ **FRG** to denote that S is a fair reachable state. Note that the branching factor for each node in any **FRG** is finite, though **FRG** itself can be infinite.

Even if **FRG** is infinite, it may still be possible to characterize it with invariant properties and prove some results. Of course, we have not done so here. Thus, we can only say, when **FRG** is finite, it provides a useful tool to analyze the protocol. In [8], Gouda et al showed that for $n = 2$, if a (cyclic) protocol has one bounded channel, then its fair reachability graph is finite. The following theorem confirms that this result is also valid for $n > 2$.

Theorem 4.2 Given a protocol $P = (P_1, P_2, \ldots, P_n)$, its fair reachability graph **FRG** is finite if one of the channels is bounded.

In fact, the above result also holds for a protocol with at least one bounded channel. Note that this sufficient condition is weaker than the one presented in [16].

However, the converse of the above theorem is not true. For example, let $P = (P_1, P_2)$ be a protocol such that: in P_1, there is only one state s_1^0 with one sending transition $\tau(s_1^0, -m) = s_1^0$; in P_2, there is only one state s_2^0 with one receiving transition $\tau(s_2^0, +m) = s_2^0$. Clearly, channel C_{12} can grow unbounded. But the fair reachability graph of this protocol is finite with only

one fair reachable state $(s_1^0, s_2^0, \epsilon, \epsilon)$.

Therefore, it would be highly desirable to find a necessary and sufficient condition to completely characterize the class of protocols whose fair reachability graphs are finite. This problem has not been solved in previous studies, even for $n = 2$. In section 5, we present a solution to this important problem.

5 Theory of Fair Reachability Analysis

In this section, we investigate two important theoretical aspects of fair reachability analysis. The first problem has to do with its error detecting capability, while the second one has to do with the termination of the state exploration procedure. Solutions for both problems contribute to the decidability results for cyclic protocols presented in the next section.

5.1 Partial Fair Execution Sequence

Let $S = (s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots, c_{n-1n})$ and $S' = (s'_1, s'_2, \ldots, s'_n, c'_{n1}, c'_{12}, \ldots, c'_{n-1n})$ be two states such that $S \mapsto^* S'$. An *execution sequence* from S to S', denoted as e, is a sequence $X^0 \xrightarrow{\tau^1} X^1 \xrightarrow{\tau^2} \ldots \xrightarrow{\tau^k} X^k, k \geq 0$, such that (1) $\forall j : 0 \leq j \leq k, X^j = (x_1^j, x_2^j, \ldots, x_n^j, c_{n1}^j, c_{12}^j, \ldots, c_{n-1n}^j)$. (2) $X^0 = S$ and $X^k = S'$. (3) $\forall j : 1 \leq j \leq k, X^{j-1} \mapsto X^j$ via the execution of transition $\tau^j = \tau(x_i^{j-1}, \sigma)$ by some process P_i in local state x_i^{j-1} of state X^{j-1}. The length of e, denoted as $|e|$, is defined as the number of transitions in e, i.e., $|e| = k \geq 0$. The corresponding *local execution sequence*, denoted as e_i in process P_i, is a sequence $x_i^0 \xrightarrow{\tau_i^1} x_i^1 \xrightarrow{\tau_i^2}, \ldots, \xrightarrow{\tau_i^{k_i}} x_i^{k_i}$ such that $\forall j : 1 \leq j \leq k_i, \tau_i^j = \tau(x_i^{j-1}, \sigma)$ is the j-th transition of P_i taken in e, and $x_i^j = \tau(x_i^{j-1}, \sigma)$. The length of e_i is defined as the number of local transitions in P_i, denoted as $|e_i|$, i.e., $|e_i| = k_i$. We use notation $e \triangleq \{e_1, e_2, \ldots, e_n\}$ to denote the correspondence among an execution sequence and its local execution sequences.

When $S = S^0$, e is an execution sequence for reachable state S'. In this case, it can be rewritten as $S^0 \xrightarrow{\tau^1} S^1 \xrightarrow{\tau^2} \ldots \xrightarrow{\tau^k} S^k$ with $S^k = S'$.

Similarly, if S' is fair reachable from S, then there is a sequence $X^0 \xrightarrow{\vec{\nu}_1} X^1 \xrightarrow{\vec{\nu}_2} \ldots \xrightarrow{\vec{\nu}_k} X^k, k \geq 0$, such that (1) $\forall j : 0 \leq j \leq k, X^j = (x_1^j, x_2^j, \ldots, x_n^j, c_{n1}^j, c_{12}^j, \ldots, c_{n-1n}^j)$. (2) $X^0 = S$ and $X^k = S'$. (3) $\forall j : 1 \leq j \leq k$, X^j is fair reachable from X^{j-1} via the execution of fair progress vector $\vec{\nu}_j$ in state X^{j-1}. Such a sequence is called a *fair execution sequence* from S to S', denoted as $fs(S, S')$. The length of $fs(S, S')$, denoted as $|fs(S, S')|$, is defined the number of fair progress vectors in the sequence, i.e., $|fs(S, S')| = k$. The corresponding local execution sequence in P_i is also denoted as e_i, i.e., $fs(S, S') \triangleq \{e_1, e_2, \ldots, e_n\}$. Note that if S is fair reachable, then $\forall j : 0 \leq j \leq k, X^j$ is fair reachable. In this case, S' is a fair reachable state.

When $S = S^0$, $fs(S, S')$ is simplified to $fs(S)$, and is rewritten as $S^0 \xrightarrow{\vec{\nu}_1} S^1 \xrightarrow{\vec{\nu}_2} \ldots \xrightarrow{\vec{\nu}_k} S^k, k \geq 0$. In this case, $fs(S')$ is called a fair execution sequence of fair reachable state S'.

By definition, for each reachable state, there exists at least one execution sequence, but such a sequence might not be unique. However, some of these execution sequences may have the same set of local execution sequences. Let $e \triangleq \{e_1, e_2, \ldots, e_n\}$ and $e' \triangleq \{e'_1, e'_2, \ldots, e'_n\}$ be two execution sequences for a reachable state S. We define a relation $\equiv$ over the set of execution sequences for S as follows: $e \equiv e'$ if and only if $\forall i \in [1..n] : e_i = e'_i$. It is straightforward that $\equiv$ is an equivalence relation over the set of execution sequences for S. Therefore, for any reachable state S, each such local execution sequence set characterizes a set of execution sequences for S. For state exploration, it is sufficient to examine these local execution sequence sets for each reachable state.

Formally, a local execution sequence set $\{e_1, e_2, \ldots, e_n\}$ is *schedulable* for a state S if and only if there is an execution sequence e for S such that the corresponding set of local execution sequences in e is $\{e_1, e_2, \ldots, e_n\}$.

Similarly, a local execution sequence set $\{e_1, e_2, \ldots, e_n\}$ is *fair schedulable* for a state S if and only if there is a fair execution sequence $fs(S)$ for S such that the corresponding set of local execution sequences in e is $\{e_1, e_2, \ldots, e_n\}$.

Given a reachable state S and one of its schedulable local execution sequence sets, $\{e_1, e_2, \ldots, e_n\}$, we want to construct for S from $\{e_1, e_2, \ldots, e_n\}$ a fair execution sequence $fs(S^k) = S^0 \xrightarrow{\vec{v}_1} S^1 \xrightarrow{\vec{v}_2} \cdots \xrightarrow{\vec{v}_k} S^k, k \geq 0$, such that $S^k \mapsto^* S$ and there is no S' such that $S^k \mapsto_f S'$ and $S' \mapsto^* S$ via the remaining local transitions in $\{e_1, e_2, \ldots, e_n\}$ in state S^k. It is not difficult to show that given S and $\{e_1, e_2, \ldots, e_n\}$, $fs(S^k)$, and thus S^k, is *unique*. Hence, $fs(S^k)$ and S^k are called the *partial fair execution sequence* and the *fair precursor* for S with respect to $\{e_1, e_2, \ldots, e_n\}$, respectively, denoted as $pfs(S, \{e_1, e_2, \ldots, e_n\})$ and $fp(S, \{e_1, e_2, \ldots, e_n\})$. When $\{e_1, e_2, \ldots, e_n\}$ is given and no confusion arises, they are denoted as $pfs(S)$ and $fp(S)$ for short. Note that in state $fp(S)$, at least one of the local execution sequences is in its tail state, i.e., $\exists i \in [1..n]$, the local state of P_i in $fp(S)$ is equal to s_i, the local state of P_i in S.

The construction of $pfs(S)$ and $fp(S)$ for S with respect to $\{e_1, e_2, \ldots, e_n\}$ is carried out by the following algorithm:

Step 1: Initially, set $X = S^0$, and $seq = S^0$.

Step 2: Construct $\vec{t}$ in state X as follows: $\forall i \in [1..n] : t_i$ is set to the transition in e_i in state X if x_i is not the tail state in e_i; $t_i = \lambda$ otherwise.

Step 3: Compute $\vec{v}$ from $\vec{t}$. If no $\vec{v}$ can be derived from $\vec{t}$, goto step 5.

Step 4: Let X' be the state resulting from the execution of $\vec{v}$ in X. Set $seq = seq \cdot \vec{v} \cdot X$ and $X = X'$. Goto step 2.

Step 5: Output $pfs(S)$ as seq and $fp(S)$ as X. End of procedure.

The correctness of above algorithm can be argued informally as follows. Let k be the number of iterations from step 2 through step 4 in the algorithm. Denote S^k as X at the time the algorithm terminates. First, observe that during each iteration, if a fair progress vector is formed, then at least two local execution sequences e_i and $e_{i\oplus 1}$ are involved. As a result, the number of transitions remained in e_i and $e_{i\oplus 1}$ are decreased by 1, respectively. Since the number of transitions in each e_j is finite, termination of the algorithm is guaranteed. Second, it is straightforward that at the time the algorithm terminates, seq is the fair execution sequence for S^k with respect to $\{e_1, e_2, \ldots, e_n\}$. Note that at this point, no fair progress vector can be derived from S^k with respect to the remaining transitions in $\{e_1, e_2, \ldots, e_n\}$. Therefore, at the time the algorithm terminates, seq and X are indeed the partial fair execution sequence and fair precursor for S with respect to $\{e_1, e_2, \ldots, e_n\}$, respectively.

In section 4, we have shown that for a cyclic protocol, any fair reachable state is a reachable state with equal channel length. Now, with the partial fair execution sequence construction algorithm, we are able to show that the converse is also true.

Theorem 5.1 Any reachable state with equal channel length is fair reachable.

Thus, we obtain an equivalence between the set of fair reachable states and the set of reachable states with equal channel length. In other words, we now have a completely characterization for the fair reachability state space.

Theorem 5.2 The fair reachability state space is exactly the set of reachable states with equal channel length.

An important implication of this theorem is that the notion of fair reachability is *consistent*

with the notion of fair execution sequence in the sense stated in the following theorem.

Theorem 5.3 Let $\{e_1, e_2, \ldots, e_n\}$ be a schedulable local execution sequence set for S. If $\{e_1, e_2, \ldots, e_n\}$ is fair schedulable for S, then any other schedulable local execution sequence set $\{e'_1, e'_2, \ldots, e'_n\}$ for S is also fair schedulable for S. In other words, if S is fair reachable, then it is fair reachable via any execution sequence for S.

5.2 Finite Fair Reachability Graph

Fair reachability analysis for a cyclic protocol P depends on the construction of the fair reachability graph **FRG** for P. For fair reachability analysis to be useful, **FRG** thus constructed must be finite. However, no necessary and sufficient condition has been established so far to exactly characterize the class of cyclic protocols amendable for fair reachability analysis. Without such a condition, the class of cyclic protocols whose **FRG**'s are finite cannot be completely described.

In this section, we solve this problem in two steps. First, we investigate the class of cyclic protocols without a sending cycle, i.e., no P_i has a cycle in which all transitions are sending transitions. Through the study, we discover the concept of *simultaneous unboundedness*, which is more fundamental in causing a cyclic protocol to have an infinite fair reachability graph than is the notion of a sending cycle. Then, we go on to show that the *lack of simultaneous unboundedness* is indeed a necessary and sufficient condition for a cyclic protocol to have a finite fair reachability graph. For completeness, we also show the undecidability of whether a cyclic protocol has a finite fair reachability graph.

For ease of presentation, we formalize the concept before the result.

Definition 5.1 A cyclic protocol $P = (P_1, P_2, \ldots, P_n)$ is *simultaneously unbounded* if for any constant $K \geq 0$, there exists a reachable state $S = (s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots, c_{n-1n})$ such that $\forall i \in [1..n] : |c_{ii\oplus 1}| > K$; otherwise, it is *not* simultaneously unbounded.

First, we notice that for a cyclic protocol without sending cycles, the notion of unboundedness is equivalent to simultaneous unboundedness.

Lemma 5.1 Given a cyclic protocol $P = \{P_1, P_2, \ldots, P_n\}$ without sending cycles. If one of its channels is unbounded, then all the other channels are unbounded.

Second, we show that for a simultaneously unbounded cyclic protocol, we can find a fair reachable state whose channels are simultaneously unbounded.

Lemma 5.2 Given a cyclic protocol $P = (P_1, P_2, \ldots, P_n)$, if there is a reachable state $S = (s_1, s_2, \ldots, s_n, c_{n1}, c_{12}, \ldots, c_{n-1n})$ such that $\forall i \in [1..n] : |c_{ii\oplus 1}| \geq K$ for some constant $K \geq 0$, then there exists a fair reachable state $S' = (s'_1, s'_2, \ldots, s'_n, c'_{n1}, c'_{12}, \ldots, c'_{n-1n})$ such that $\forall i \in [1..n] : |c'_{ii\oplus 1}| \geq K$.

With these two lemmas, we can establish an equivalence between the finiteness of reachability graph and finiteness of fair reachability graph for the class of cyclic protocols without sending cycles.

Theorem 5.4 Given a cyclic protocol $P = \{P_1, P_2, \ldots, P_n\}$ without sending cycles, its fair reachability graph is finite if and only if its reachability graph is finite.

In fact, we can derive a stronger result based on the preceding proof.

Theorem 5.5 Given a cyclic protocol $P = \{P_1, P_2, \ldots, P_n\}$ without reachable sending cycles, its fair reachability graph is finite if and only if its reachability graph is finite.

From this theorem, we can see that simultaneous channel unboundedness is another factor, and probably a more fundamental factor than sending cycle in causing a fair reachability graph

to become infinite, as is confirmed by the following theorem.

Theorem 5.6 Given a cyclic protocol $P = (P_1, P_2, \ldots, P_n)$, P has a finite fair reachability graph if and only if P is not simultaneously unbounded.

The next theorem says that if a cyclic protocol has a finite **FRG**, then we will be able to find the least upper bound $K \geq 0$ such that each reachable state has at least one channel whose length is bounded by K. In fact, K takes on the value that is the longest channel length any $S \in \mathbf{F}$ can have.

Theorem 5.7 Given a cyclic protocol P with a finite **FRG**, we can determine the least upper bound $K \geq 0$ such that each reachable state of P has at least one channel whose length is bounded by K. Specifically, K is exactly the value such that $\mathbf{F}_K \neq \emptyset \wedge \mathbf{F}_{K+1} = \emptyset$, i.e., the longest channel length among all the nodes in **FRG**.

The discovery of this necessary and sufficient condition is significant in that we are now able to exactly describe the class of cyclic protocols with finite fair reachability graphs from the protocol operational semantics viewpoint. To the best of our knowledge, this condition is the first necessary and sufficient condition for a cyclic protocol to have a finite fair reachability graph. However, as expected, the decidability aspect of this condition is negative, as is stated in the following theorem. The proof of the theorem is based on showing it is true for $n = 2$, an easy reduction by using the decidability result of boundedness detection established in [10].

Theorem 5.8 Given a cyclic protocol $P = (P_1, P_2, \ldots, P_n)$, it is undecidable whether P has a finite fair reachability graph.

6 Applying Fair Reachability Analysis

To demonstrate the power of our generalized fair reachability analysis technique, we show in this section that both deadlock detection and livelock detection are decidable for cyclic protocols with finite reachability graphs. The decidability of deadlock detection is a direct result from the theory of generalized fair reachability presented in section 5, while the decidability of livelock detection is an easy extension to $n > 2$ from the one established in [8].

6.1 Deadlock Detection

Let $P = (P_1, P_2, \ldots, P_n)$ be a cyclic protocol. From the discussion in section 5, we know that the fair reachable state space $\mathbf{F}$ for P is exactly the set of reachable states with equal channel length. Hence, the set of deadlock states is included in $\mathbf{F}_0$. If $\mathbf{F}$ is finite, then deadlocks in P are detectable by constructing the finite fair reachability graph **FRG** for P. In addition, we know that the class of cyclic protocols that are not simultaneously unbounded is exactly the class of cyclic protocols whose **FRG**'s are finite. As a result, we obtain the decidability of deadlock detection for this class of cyclic protocols, as stated in the following theorem.

Theorem 6.1 Given a cyclic protocol P whose fair reachability graph is **FRG**, P has a deadlock state if and only if there is a deadlock node in **FRG**. Hence, deadlock detection is decidable for the class of cyclic protocols whose fair reachability graphs are finite.

6.2 Livelock Detection

A livelock occurs in a protocol when each communicating entity is busy exchanging messages but doing nothing "useful". In [8, 9], livelock is modeled by introducing a marking function into the communicating finite state machine model. For $n = 2$, they showed that livelock detection is undecidable for general protocols [9], but is decidable for protocols with finite fair reachability graphs [8].

In this subsection, we are going to generalize these results to cyclic protocols. In modeling livelock within the communicating finite state machine model, we adopt and generalize the definitions in [8].

A *marked cyclic protocol* is a tuple (P, φ), where $P = (P_1, P_2, \ldots, P_n)$ is a cyclic protocol, and φ is a function, called the *marking* of the protocol, that assigns to each edge in P_i a value in $\{0, 1\}$. Let ε be an edge in P_i, if $\varphi(\varepsilon) = 1$, it is called a *progress edge*; if $\varphi(\varepsilon) = 0$, it is called a *nonprogress edge*.

Definition 6.1 Let (P, φ) be a marked cyclic protocol, and let $\mathcal{C}_i$, $1 \leq i \leq n$, be a cycle in P_i. The tuple $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n)$ is called a *livelock* in (P, φ) if and only if the following three conditions are satisfied: (1) Each cycle $\mathcal{C}_i$ has at least one edge (transition), denoted as $cs_i^0 \xrightarrow{\varepsilon_1^i} cs_i^1 \xrightarrow{\varepsilon_2^i} \cdots \xrightarrow{\varepsilon_{q_i}^i} cs_i^{q_i}$, where $cs_i^0 = cs_i^{q_i}$ and $q_i > 0$. (2) All edges of cycle $\mathcal{C}_i$ are nonprogress. (3) There exists an infinite execution sequence $e \triangleq \{e_1, e_2, \ldots, e_n\}$, where

$$e_i = s_i^0 \xrightarrow{\tau_i^1} s_i^1 \xrightarrow{\tau_i^2} \cdots \xrightarrow{\tau_i^{n_i}} cs_i^0 \xrightarrow{\varepsilon_i^1} cs_i^1 \xrightarrow{\varepsilon_i^2} \cdots \xrightarrow{\varepsilon_i^{q_i}} cs_i^0 \xrightarrow{\varepsilon_i^1} cs_i^1 \xrightarrow{\varepsilon_i^2} \cdots \xrightarrow{\varepsilon_i^{q_i}} cs_i^0 \xrightarrow{\varepsilon_i^1} \cdots\cdots$$

and $n_i \geq 0$. In other words, P has a livelock if and only if P can reach a state from which P can loop indefinitely through a nonprogress cycle whose local execution cycles correspond to $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n)$.

By the undecidability of livelock detection for $n = 2$ machines shown in [9], we have the following:

Theorem 6.2 Livelock detection is undecidable for cyclic protocols.

Next, we show that livelock detection can be solved for cyclic protocols with finite fair reachability graphs [6]. The proof is a straightforward generalization from the one in [8].

Theorem 6.3 Given a marked cyclic protocol P whose fair reachability graph **FRG** is finite, P has a livelock if and only if there is a fair execution cycle in **FRG** such that each of its corresponding local execution cycle is nonempty and is marked nonprogress. Therefore, livelock detection is decidable for the class of cyclic protocols with finite fair reachability graphs.

7 Conclusion

In this paper, we generalized the fair reachability analysis technique to cyclic protocols with $n \geq 2$ communicating finite state machines. We established an equivalence between the set of all fair reachable states and the set of all reachable states with equal channel length, and discovered a necessary and sufficient condition for the class of cyclic protocols whose fair reachability graph are finite. The effectiveness of generalized fair reachability analysis is demonstrated by showing both deadlock detection and livelock detection are decidable for the class of cyclic protocols with finite fair reachability graphs. The strength of our approach lies in the natural generalization of existing fair reachability technique and its simple, straightforward, and efficient decision procedure, which were missing in both [16, 17] and [13, 14].

Fair reachability analysis was originally proposed as a technique to reduce state explosion during reachability analysis [18]. The same argument also applies to our work reported in this paper. By forcing the system to progress through a fair execution sequence, we have cut down the redundancy of state exploration due to equivalent execution sequences. Thus, our generalized fair reachability technique also significantly reduces the complexity of protocol verification.

In [7], we study the detection of other logical errors for the class of cyclic protocols whose fair reachability graphs are finite. Therefore, for the class of cyclic protocols that are not simul-

taneously unbounded, logical correctness can be validated algorithmically using our generalized fair reachability analysis technique. However, finite extensions of a fair reachability graph are needed in order to detect logical errors other than deadlocks, as was the case for boundedness detection for $n = 2$ in [10]. This phenomenon shows that *pure* fair reachability analysis is not sufficient to handle all the logical errors for the class of cyclic protocols with finite fair reachability graphs. However, for this class of cyclic protocols, the finite fair reachable state space does serve well as a basis from which other logical errors can be detected.

It is possible to incorporate internal transitions into our fair progress vector formulation to allow our generalized fair reachability technique to handle cyclic protocols with internal transitions and still achieve good state reduction in the analysis. We are currently working on this issue.

During the write-up of this paper, we were informed of the independent work by Peng on extending fair reachability to a model called "single-link communicating finite state machines" [15]. In this model, each process can have multiple output channels but has only one common input channel to store messages from other processes. Although cyclic protocols are included in this model, the notion of fair reachability in this model is quite different from ours in that only two machines are allowed to make progress at one time restricted by the so-called "weight-balance" constraint in [15]. It is not clear, however, what class of protocols in his model is amendable for his analysis technique. For cyclic protocols, our fair reachability formulation has the following advantages: (1) Our fair reachability state space maintains the same nice equal channel length property as for $n = 2$ [18, 10]. (2) Both concurrency and synchronization vectors in our fair reachability notion allow more than two machines to progress at the time. As a result, for most cyclic protocols, our analysis achieves greater reduction in state generation than the one in [15]. (3) Aside from deadlock, our approach can also detect livelocks and other logical errors, which are not covered in [15].

Many open problems remain concerning our approach. First, although we have found a necessary and sufficient condition for the class of cyclic protocols whose logical correctness is decidable, we are not sure how general it is in terms of tightening the boundary of cyclic protocols whose logical correctness is decidable. Further investigation of this aspect is necessary in order to fully evaluate its role in the decidability hierarchy. Second, a cyclic protocol is still simple in topology. It would be beneficial to look into the possibility of generalizing our work to protocols with more complicated and yet regular network topologies. Third, fair reachability analysis is only one type of improved reachability analysis techniques studied in the two machine case. The result of our work here should encourage more research on extending other techniques to the analysis of protocols with more than two machines. In [7], the collective power of both fair progress and *maximal progress* [11, 13] state exploration is illustrated in the finite extension process, and has produced encouraging results. But more work along this line is necessary. Finally, it would be interesting to investigate the possibility of carrying the fair reachability analysis technique over to other specification models, such as the extended finite state machine model.

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] L. Cacciari and O. Rafiq, "On Improving Reduced Reachability Analysis," Proc. Fifth International Conference on Formal Description Techniques for Distributed Systems and Communications Proto-

cols – FORTE'92, Perros-Guirec, France, October 13-16, 1992, M. Daiz and R. Groz (Ed.), Elsevier Science Publishers B.V. (North-Holland), 1992, pp. 137–152.

[3] L. Cacciari and O. Rafiq, "Decidability Issues in Reduced Reachability Analysis," Proc. 1993 International Conference on Network Protocols, San Francisco, CA, October 19–22, 1993, pp. 158–165.

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

[5] H. Liu and R.E. Miller, "Deadlock Detection for Cyclic Protocols Using Generalized Fair Reachability Analysis," Technical Report CS-TR-3135, Dept. of Computer Science, Univ. of Maryland at College Park, September 1993.

[6] H. Liu and R.E. Miller, "Generalized Fair Reachability Analysis for Cyclic Protocols: Part 1," Technical Report CS-TR-3204, Dept. of Computer Science, Univ. of Maryland at College Park, January 1994.

[7] H. Liu and R.E. Miller, "Generalized Fair Reachability Analysis for Cyclic Protocols: Decidability for Logical Correctness Problems," March 1994, *submitted for publication.*

[8] M.G. Gouda, C.H. Chow, and S.S. Lam, "Livelock Detection in Networks of Communicating Finite State Machines," Technical Report, TR-84-10, Dept. of Computer Science, Univ. of Texas at Austin, April 1984.

[9] M.G. Gouda, C.H. Chow, and S.S. Lam, "On the Decidability of Livelock Detection in Networks of Communicating Finite State Machines," Proc. Protocol Specification, Testing and Verification, IV, Y. Yemini, R. Strom, and S. Yemini (Ed.), Elsevier Science Publishers B.V. (North-Holland), 1985, pp. 47–56.

[10] 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.

[11] 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.

[12] K. Okumura, "Protocol Analysis from Language Structure," Proc. Protocol Specification, Testing and Verification, VIII, S. Aggarwal and K. Sabnani (Ed.), Elsevier Science Publishers B.V. (North-Holland), 1988, pp. 113–124.

[13] J. Pachl, "Reachability Problems for Communicating Finite State Machines," Research Report, CS-82-12, Dept. of Computer Science, Univ. of Waterloo, May, 1982

[14] J. Pachl, "Protocol Description and Analysis Based on a State Transition Model with Channel Expressions," Proc. Protocol Specification, Testing and Verification, VII, J. Rubin and C.H. West (Ed.), Elsevier Science Publishers B.V. (North-Holland), 1987, pp. 207–219.

[15] W. Peng, "Single-Link Communicating Finite State Machines," Dept. of Computer Science, Southwest Texas State Univ., October, 1993, *private communication.*

[16] W. Peng and S. Purushothaman, "A Unified Approach to the Deadlock Detection Problem in Networks of Communicating Finite State Machines," Proc. of 2nd International Conference, CAV'90, New Brunswick, N.J., June, 1990, E.M. Clarke and R.P.Kurshan (Ed.), Lecture Notes in Computer Science, Vol. 531, pp. 243–252.

[17] W. Peng and S. Purushothaman, "Data Flow Analysis of Communicating Finite State Machines," ACM Transactions on Programming Languages and Systems, Vol. 13, No. 3, 1991, pp. 399–442.

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

[19] D. Sidhu, A. Chung, and T.P. Blumer, "Experience with Formal Methods in Protocol Development," ACM SIGCOMM, Computer Communication Review, Vol. 21, No. 2, April, 1991, pp. 81–101.