Synchronizable Test Sequences of Finite State Machines[*]

August 14, 1995

K. C. Tai and Y. C. Young [#]

Author for correspondence:

Professor K. C. Tai
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695-8206, USA
Tel: (919) 515-7146
Fax: (919) 515-7896
e-mail: kct@csc.ncsu.edu

Abstract

The finite state machine (FSM) model is commonly used for specifying communication protocols and other types of distributed systems. With the use of multiple testers for an FSM, the synchronization between inputs from different testers becomes a problem. A **synchronizable test sequence** of an FSM is a test sequence for which the synchronization problem either does not exist or can be solved by communication between testers. In this paper, we consider two testing strategies for an FSM: **port-based testing**, which does not allow testers for the FSM to communicate with each other, and **group-based testing**, which divides the ports of the FSM into groups and allows the testers for ports in the same group to communicate with each other. For each type of testing, we define a necessary and sufficient condition under which a test sequence of an FSM is synchronizable and show how to generate a set of testers according to a given test sequence. Also, we discuss the issues of test sequence generation and fault detection, and present the results of some empirical studies.

Keywords: finite state machines, test sequence generation, synchronizable test sequences

---

[*] This work was supported in part by the US National Science Foundation under grant CCR-9309043.

[#] The author is also with IBM, Raleigh, North Carolina.

1. Introduction

The finite state machine (FSM) model is commonly used for specifying communication protocols. The problem of generating test sequences based on an FSM has been studied for about two decades (Tarnay, 1991) (Sarikaya, 1993). When an implementation of an FSM is tested for conformance, test sequences are derived from the FSM, and testers (or test drivers) for the implementation are constructed according to these test sequences. With the use of multiple testers for a protocol implementation, the synchronization between inputs from different testers becomes a problem. A **synchronizable test sequence** of an FSM is a test sequence for which the synchronization problem either does not exist or can be solved by communication between testers.

In OSI conformance test [Tar91, Sar93], the lower and upper testers are considered for the implementation under test (IUT). The lower (upper) tester represents the lower (upper) layer of the IUT. In the remote test architecture, the lower and upper testers are not allowed to communicate with each other, and in other test architectures, the lower and upper testers are either coordinated by test coordination procedures or allowed to communicate with each other. In [SB84] Sarikaya and Bochmann discussed the synchronization problem in the remote testing of an FSM with two ports (for the lower and upper testers) and defined a type of synchronizable test sequence that does not have the synchronization problem.

Due to the existence of distributed database systems and communication networks, FSMs with multiple ports are needed to specify protocols [LDB93]. Also, the use of multi-port FSMs makes the design of communication protocols flexible. Protocol specification languages such as LOTOS, Estelle, and SDL [Tur93] allow the use of multiple ports. In addition, many concurrent programming languages allow a process to have multiple entries or ports to communicate with other processes [And91].

To test a multi-port FSM, we assume that a tester is generated for each port of the FSM. In this paper, we consider two testing strategies for an FSM: **port-based testing**, which does not allow testers of the FSM to communicate with each other, and **group-based testing**, which divides the ports of the FSM into groups and allows the testers for ports in the same group to communicate with each other. Group-based testing is more general than port-based testing since the latter can be viewed as a special case of the former with each group containing only one port. One reason for group-based testing is that an FSM may be connected to one component of the FSM's environment via two or more ports, which can be considered as one group. (The use of multi-port connection between two components is often more convenient than that of one-port connection.) For example, a multi-port FSM may have two groups of testers, one group for representing the lower layer and the other group for representing the upper layer. Another reason for group-based testing is to allow flexibility in dealing with various testing architectures. For example, if the lower and upper testers for a two-port FSM are allowed to communicate, then testing of the FSM becomes group-based testing, not port-based testing.

This paper is organized as follows. Section 2 provides basic definitions. Section 3 summarizes previous work on synchronizable test sequences of an FSM. Section 4 gives our motivation for extending the definition of a synchronizable test sequence in [SB84]. Section 5 discusses the following issues for port-based testing of an FSM: a necessary and sufficient condition under which a test sequence is synchronizable, the generation of testers according to a given test sequence, and the selection of synchronizable test sequences. Section 6 addresses the same issues for group-based testing of an FSM. Section 7 discusses some issues related to fault detection by synchronizable test sequences. Section 8 shows the results of our empirical studies on synchronizable test sequences. Section 9 concludes this paper.

2. Preliminaries

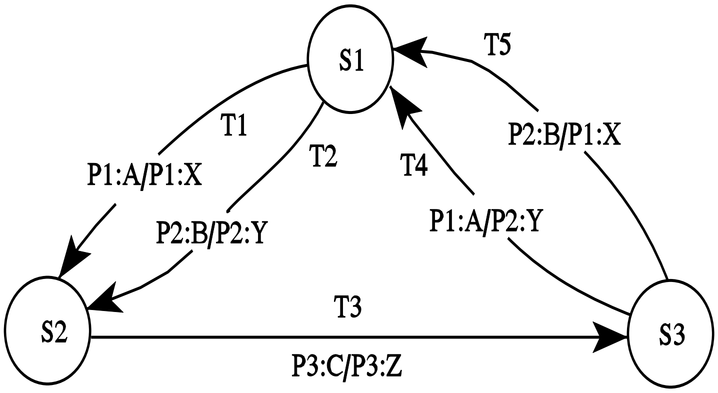
Below we provide a formal definition of a multi-port FSM, which is different from that in [LDB93].

Definition. A **finite state machine** (FSM) M with multiple ports is defined as a 6-tuple M=(S, I, O, T, U, s0), where

- S is the set of states of M
- I is the set of input symbols of M. I includes the empty input symbol, denoted as **null**. Each non-empty input symbol is of the form P:A, where P denotes a port and A an input message.
- O is the set of output symbols of M. Each output symbol is of the form Q:B, where Q denotes a port and B an output message.
- T is the transition function of M, which maps from D to S, where $D \subseteq S \times I$. (Thus, M is **deterministic**.)
- U is the output function of M, which maps from D to $(O1 \times O2 \times \ldots \times Ov) \cup \{ \varepsilon \}$, where each Oi, 1<=i<=v, is a non-empty output symbol, and ε stands for the empty output list. (The output of a transition may contain two or more output symbols with the same port.)
- s0 is the initial state of M.

In the above definition, each non-empty input or output symbol contains two elements: port name and message. The separation of port name and message makes it easier to discuss synchronizable test sequences. If the transition function of an FSM allows a state to have a transition on an input, then this input is said to be **valid** for the state. The output function of an FSM allows zero, one, or more output symbols to be associated with a transition. An FSM communicates with its environment by receiving input symbols and sending output symbols via ports. Each port of an FSM has two unbounded FIFO queues: the input queue, which keeps input messages of the FSM, and the output queue, which keeps output messages of the FSM. It is assumed that the delivery of messages from an FSM to any of its port is FIFO, i.e., messages sent from M to the same port are received in the order sent. For an FSM with two ports, its two testers are commonly referred to as the lower and upper testers. In this paper, the lower and upper testers are referred to as the **L-tester** and the **U-tester**, respectively and the two ports connected to the L- and U-testers are referred to as ports $\mathbf{P_L}$ and $\mathbf{P_U}$, respectively.

Assume that during an execution of an FSM and its environment, the current state of the FSM is S. For a transition T of S, if the first input message at the input port of S is valid for S, then T is said to be **eligible** for S. If S has two or more eligible transitions, then one of them is chosen at random for execution. If S does not have any eligible transitions, then the FSM waits until at least one eligible transition for S becomes available.



A **transition** T from state Si to state Sj that has P:A as the input symbol and (Q1:B1,Q2:B2,...,Qv:Bv) as the list of output symbols is denoted as (Si, Sj, P:A/(Q1:B1,Q2:B2,...,Qv:Bv). Si is referred to as the **head state** of T or **head**(T), Sj the **tail state** of T or **tail**(T), P the input port of T, A the input message of T, Qi , 1<=i<=v, an output port of T, and Bi an output message of T. P:A/(Q1:B1,Q2:B2,...,Qv:Bv) is referred to as the **label** of T or **label**(T). If v = 1, P:A/(Q1:B1) is often referred to as P:A/Q1:B1. If T has no output symbols, it is denoted as (Si, Sj, P:A/ε). If a transition has null as the input symbol, it is referred to as a **timeout transition**.[1] A port is said to be **involved** in a transition if this port is either the input port or an output port of the transition.

Fig. 1. FSM M1 with Ports P1, P2, and P3

For an FSM M, let $D_M = (V_M, E_M)$ denote the **digraph** of M, where V_M is a set of vertices, each representing a state of M, and E_M is a set of directed edges, each representing a transition of M. Fig. 1 shows the digraph of FSM M1, which has three ports P1, P2, and P3, and three states S1, S2 and S3, with S1 being the initial state. A, B and C are input messages of ports P1, P2 and P3, respectively, and X, Y and Z are output messages of ports P1, P2 and P3, respectively. A tabular representation of M1 is given below, which has inputs as rows and states as columns.

	S1	S2	S3
	-----------------	-----------------	-----------------
P1:A	P1:X, S2 (T1)		P2:Y, S1 (T4)
P2:B	P2:Y, S2 (T2)		P1:X, S1 (T5)
P3:C		P3:Z, S3 (T3)	

Each non-empty entry in the above table defines a transition for the corresponding state and input symbol; it contains the output symbol and tail state of this transition, followed by a pair of parentheses enclosing the name of this transition.

Let M denote an FSM. A **transition sequence** of M is a sequence of consecutive transitions in M, and a **test sequence** of M is a transition sequence of M starting from the initial state of M. In this paper, a transition or test sequence is often denoted as a sequence of transition names connected by ".". For example, T1.T3.T5 and T4.T2.T3 are transition sequences of M1 in Fig. 1, and the former is a test sequence of M1, but the latter is not. For a transition sequence, its

---

[1] In some protocols, different timeout signals are used and thus a state may have two or more timeout transitions. In this paper, we consider different timeout signals as different input symbols.

i/o sequence refers to the sequence of labels associated with the transitions in the transition sequence. Since a test sequence of M starts with the initial state of M, it does not need to keep information about the head and tail states of transitions in the test sequence. Thus, a test sequence of M can be denoted by its i/o-sequence. For example, T1.T2.T3 of M can be denoted by its i/o sequence, which is (P1:A/P1:X, P3:C/P3:Z, P2:B/P1:X). Similarly, a transition sequence of M that starts from a state can be denoted by the start state and the i/o sequence of the transition sequence. For a transition sequence E of M, if a transition T occurs (immediately) before a transition T' in E, then T is said to be a (the immediate) **predecessor** of T' in E and T' a (the immediate) **successor** of T in E.

A digraph D = (V,E) is **strongly connected** if for every pair of vertices Vi and Vj in V, there exists a path from Vi to Vj. An FSM is said to be strongly connected if its digraph is strongly connected. An FSM is said to be **completely specified** if each state of this FSM has a transition for every possible input symbol. For a state S of an FSM, let IO(S) be the set of i/o sequences that start from S. An FSM is said to be **minimal** if for any two states S and S' of the FSM, IO(S) ≠ IO(S'). In this paper, we assume that each FSM is deterministic, strongly connected, minimal, and possibly incompletely specified, unless otherwise specified.

A **postman tour** (or **transition tour**) of a digraph D = (V,E) is a path in D that starts and ends at the same vertex and covers each edge in E at least once. A postman tour of an FSM is a postman tour in the digraph of this FSM. A digraph D has a postman tour if and only if D is strongly connected. The Chinese postman problem, which is to find a minimum-cost postman tour for a digraph, can be solved in polynomial time [EJ73]. The T-method for testing an FSM requires the use of a postman (or transition) tour of the FSM's digraph [NT81].

For a state S in an FSM M, a **unique input/output** (UIO) sequence E is a transition sequence starting from S such that if the sequence of input symbols of E is applied to a state in M other than S, the sequence of corresponding output symbols is different from the sequence of output symbols of E. In other words, an UIO sequence for S can distinguish S from other states in M. The UIO- or U-method for testing an FSM verifies the tail state of a transition of the FSM by using an UIO sequence for the tail state [SD88].

3. Previous Work on Synchronizable Test Sequences of an FSM

In [SB84] Sarikaya and Bochmann considered the synchronization problem for an FSM with two ports (for the L- and U-testers, which do not communicate with each other). Two consecutive transitions are said to have the **synchronization problem** if the input port of the second transition is not involved in the first transition (i.e., the input port of the second transition is neither the input port nor an output port of the first transition.) A test sequence is said to be **synchronizable** if no two consecutive transitions of the test sequence have the synchronization problem. [SB84] also discussed how to extend existing test sequence generation methods in order to generate synchronizable test sequences. In order to distinguish the type of synchronizable test sequence defined in [SB84] from other types of synchronizable test sequences, the former is referred to as a **pair-synchronizable test sequence** in the remainder of this paper.

In [CLC90] Chen, et al. defined a **tightly synchronizable test sequence** of a two-port FSM as a test sequence such that for any two consecutive transitions, the input port of the second transition is the same as the output port of the first transition. (Each transition was assumed to have at most one output symbol.) They showed how to construct a graph, called the **duplex digraph**, of a two-port FSM such that a test sequence of the duplex graph is a tightly synchronizable test sequence of the FSM.

In [BU91] Boyd and Ural investigated complexity issues related to pair-synchronizable test sequences of a two-port FSM. They presented a necessary and sufficient condition for the existence of a pair-synchronizable postman tour of an FSM, and this condition can be determined in polynomial time. Also, they showed that the problem of finding a minimum-length pair-synchronizable postman tour of an FSM is NP-complete.

In [UW93] Ural and Wang considered two-port FSMs satisfying a number of conditions, including the necessary and sufficient condition in [BU91] and the condition that each state in the FSM possess two UIO sequences, one for the L-tester and the other for the U-tester. By modifying the algorithm in [CLC90] for the construction of a duplex digraph and by providing additional algorithms, they showed how to find a pair-synchronizable UIO-based test sequence of an FSM in polynomial time. In [GU95] Guyot and Ural extended the work by showing the construction of a digraph of an FSM M such that all paths in the digraph are pair-synchronizable test sequences of M and by showing that under certain conditions, a pair-synchronizable test sequence of M can distinguish M from any FSM that is not isomorphic to M.

In [CU95] Chen and Ural allowed the L- and U-testers for a two-port FSM to communicate with each other, and they considered the cost of such communication in test sequence generation. They showed how to convert a non-pair-synchronizable test sequence into a synchronizable test sequence by adding communication statements between the L- and U-testers. Such synchronizable test sequences are referred to as **LU-synchronizable test sequences** in this paper. Based on the duplexU digraph of an FSM, a minimum-cost LU-synchronizable test sequence using multiple UIO sequences can be generated. Since the generation of such a test sequence is an NP-complete problem, they proposed a heuristic algorithm that yields an LU-synchronizable UIO-based test sequence with its cost within a bound of the minimum cost.

In [LDB93] Luo, et al. discussed the need for FSMs with multiple ports and extended the definition of a pair-synchronizable test sequence for a multi-port FSM. They also investigated the issue of fault coverage by pair-synchronizable test sequences.

The use of synchronizable test sequences is necessary for conformance testing of an implementation of an FSM, since the implementation is treated as a black box. When the implementation is tested by its developers, it can be modified to perform deterministic testing and thus make every test sequence synchronizable. Details on deterministic testing and debugging of concurrent programs can be found in [TA87, CT91, TCO91, TC95].

4. Motivations for Extending the Definition of a Pair-Synchronizable Test Sequence

In this section, we show that the definition of a pair-synchronizable test sequence is more restrictive than necessary and that some non-pair-synchronizable test sequences are actually synchronizable. Below we first give a general definition of a synchronizable test sequence.

Definition. A test sequence for an FSM M is said to be **synchronizable** if any execution of M and the testers generated according to the test sequence is deterministic (i.e., at any time during an execution of M and these testers, the current state of M has at most one eligible transition).
Notes:
- In this paper, the issue of fault detection is not discussed. Therefore, we consider executions of M and its testers, not executions of an implementation of M and its testers.
- Whether a test sequence of M is synchronizable depends on the test sequence, M, and the construction of testers for the test sequence.

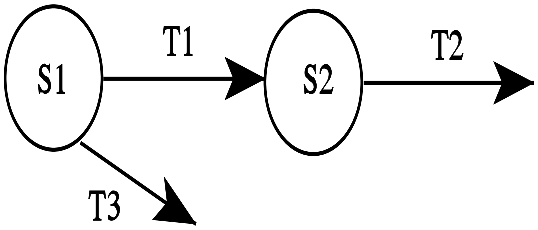
According to [SB84], two consecutive transitions are said to have the **synchronization problem** if the input port of the second transition is not involved in the first transition. In the remainder of this paper, this synchronization problem is referred to as the **pair-based synchronization problem**. To illustrate this problem for a two-port FSM, we first show how the L- and U-testers work for two consecutive transitions T1 and T2. Assume that the input port of T2 is port P_L. The following are three possible relationships between port P_L and T1:

(a) Port P_L is an output port of T1. In this case, the L-tester sends the input message of T2 to port P_L immediately after receiving the output message of T1 from port P_L. No synchronization problem exists.

(b) Port P_L is not an output port of T1, but is the input port of T1. In this case, the L-tester sends the input message of T2 to port P_L immediately after sending the input message of T1 to port P_L. Thus, the input message of T2 will be received after the completion of T1. No synchronization problem exists.

(c) Port P_L is neither the input port nor an output port of T1 (i.e., only port P_U is involved in T1). In this case, the L-tester sends the input message of T2 to port P_L without any dependency on T1. According to [SB84], this case creates the pair-based synchronization problem.

Now we examine the effect of case (c). Let S1 and S2 be the head and tail states of T1, respectively. Assume that S1 is the current state, the input message of T2 is the first input message at port P_L, and the input message of T1 is the first input message at port P_U. Also, we assume that S1 contains a transition, say T3, that has the same input port and message as T2 (see below).



Since the first input messages at both ports P_L and P_U are valid for S1, both transitions T1 and T3 are eligible for S1. If the first input message at port P_L is accepted (i.e., transition T3 is executed), then a synchronization problem occurs, since T1, not T3, is expected to be executed. To prevent this problem to occur, the definition of a pair-synchronizable test sequence in [SB84] does not allow case (c). As a result, at any time during an execution of a two-port FSM by using a pair-synchronizable test sequence, at most one of ports P_L and P_U contains input messages.

Now we assume that transition T3 does not exist (i.e., state S1 does not contain any transition with the same input symbol as T2). When S1 is the current state, the arrival of the input message of T2 at port P_L does not create the pair-based synchronization problem since the input symbol of T2 is invalid for S1. Therefore, we can allow both ports P_L and P_U to contain input messages as long as only one eligible transition exists for the current state.

Another motivation for extending the definition of a pair-synchronizable test sequence is to allow timeout transitions in a synchronizable test sequence. Since a timeout transition does not have an input port, a pair-synchronizable test sequence does not contain any timeout transitions.[2] Assume that when an FSM M is tested by using a test sequence E, the current state S of M has a timeout transition T. If T is the next transition in E and the only eligible transition for S, then T does not create the synchronization problem. On the other hand, if the next transition in E, say T', is a non-timeout transition and T' is the only eligible non-timeout transition for S, there is no guarantee that T' is chosen for execution, since T is also eligible for S.

Following the above discussion, the definition of a pair-synchronizable test sequence is more restrictive than necessary, and some non-pair-synchronizable test sequences are actually synchronizable. Note that the definition of a pair-synchronizable test sequence is based solely on the test sequence. Our new concept of a synchronizable test sequence of an FSM is based on not only the test sequence, but also the transitions of the states in the FSM that are passed through by the test sequence.

Since the multi-port FSM model becomes more commonly used and is more general than the two-port FSM model, we define new synchronous test sequences for a multi-port FSM. As mentioned earlier, we consider two testing strategies for an FSM: **port-based testing**, which does not allow testers for the FSM to communicate with each other, and **group-based testing**, which divides the ports of the FSM into groups and allows the testers for ports in the same group to communicate with each other. Since each port corresponds to a unique tester, group-based testing of an FSM can also be viewed as dividing the testers for the FSM into groups and allowing testers in the same group to synchronize with each other. (One alternative strategy for group-based testing of M is to have one tester for each group of ports of M. In Section 7, we show that "one tester for each port of M" is more effective than "one tester for each group of ports of M" for fault detection.) The next two sections address several issues on port and group-based testing, respectively.

---

[2] In [Sar93] timeout transitions are allowed in a synchronizable test sequence, but the discussion is incomplete.

5. Port-Based Testing of an FSM

This section discusses several issues on port-based testing of an FSM, including the generation of port-based testers according to a given test sequence, the necessary and sufficient condition under which a test sequence is port-synchronizable and the selection of port-synchronizable test sequences.

5.1. Port-Based Testers and Synchronization Problem

In this section, we first show the generation of a set of testers from a test sequence, according to port-based testing. Such testers are referred to as **port-based testers**. We then define the port-based synchronization problem, according to port-based testers..

The port-based tester for port P of an FSM M according to a test sequence E of M, denoted as Tester(P,E,M), contains the following two types of statements:

- send A to P;
- receive B from P;

where A is an input message of M, B is an output message of M, a send operation is non-blocking, and a receive operation is blocking. For the sake of simplicity, Tester(P,E,M) is referred to as Tester(P) if E and M are either implied or immaterial, and as Tester(P,E) if M is implied. When "receive B from P" is executed by Tester(P), if the first output message of M at port P is not B, then Tester(P) has an abnormal termination. It is assumed that the delivery of messages from Tester(P) to port P is FIFO, i.e., messages sent from Tester(P) to port P are received in the order sent. In section 2, the FSM model assumes that each transition is associated with an input symbol and zero, one, or more output symbols. Thus, a transition in an FSM can be viewed as a receive statement, followed by zero, one, or more send statement. For the sake of simplicity, a tester is defined as a sequence of send and receive statements, not as a sequence of transitions. If necessary, a sequence of send and receive statements can be converted into a sequence of transitions.

Algorithm Port_Tester_Gen

```
Input:   a test sequence E of an FSM M
Output:  Tester(E), which is {Tester(P,E) | P is a port involved in E}
(1)  For each port P involved in E, let Tester(P,E) be empty.
(2)  For each transition T in E (starting from the first transition in E),
       (a) if the input symbol of T is P:A, then add "send A to P" to the end of Tester(P,E).
       (b) if the list of output symbols of T is (Q1:B1,Q2:B2,...,Qv:Bv), then
             for each i from 1 to v, add "receive Bi from Qi" to the end of Tester(Qi,E).
end of algorithm.
```

Consider test sequence R1 = T1.T3.T4 of M1 , which is shown in Fig. 1. By applying the above algorithm to T1.T3.T4, Tester(R1) contains the following testers for ports P1, P2, and P3:

Tester(P1,R1)	Tester(P2,R1)	Tester(P3,R1)
(T1) send A to P1	(T4) receive Y from P2	(T3) send C to P3
(T1) receive X from P1		(T3) receive Z from P3
(T4) send A to P1		

(To improve readability, each send or receive statement in a tester is preceded with the name of the transition from which the statement is derived.) R1 is not pair-synchronizable since the input port of T3, which is P3, is not involved in T1 (i.e., P3 is neither the input port nor an output port of T1), and the input port of T4, which is P2, is not involved in T3. When an execution of M1 and the above testers starts, state S1 is the current state. When a transition of S1 is to be chosen, it is possible that messages A and C are available at ports P1 and P3, respectively. However, only P1:A is valid for S1 and thus only transition T1 is eligible for S1. After the execution of T1, state S2 becomes the current state of M1. When a transition of S2 is to be chosen, it is possible that messages A and C are available at ports P1 and P3, respectively. However, only P3:C is valid for S2 and thus only transition T3 is eligible for S2. Although R1 is not pair-synchronizable, it does not have the problem of synchronizing inputs form different testers. Below we formally define the synchronization problem based on the use of port-based testers.

Definition. Let E be a test sequence of an FSM M and let Tester(E) be the set of testers generated by algorithm Port_Tester_Gen according to E. During an execution of M and Tester(E), the **port-based synchronization problem** occurs when a state of M has two or more eligible transitions. (As a result of this problem, the execution of M and Tester(E) is nondeterministic and may result in an abnormal termination.)

5.2 Definition of a Port-Synchronizable Test Sequence of an FSM

Definition. For a transition T in a transition sequence E of an FSM, its **port-predecessor transition**, denoted as **PPT**(T,E), is defined as follows:

- If T is a timeout transition, then PPT(T,E) is null.
- If T is not a timeout transition, PPT(T,E) is the closest predecessor of T in E that involves the input port of T. If such a predecessor of T in E does not exist, then PPT(T,E) is null.

Let T be a non-timeout transition in a test sequence E of an FSM M and let P:A be the input symbol of T. According to algorithm Port_Tester_Gen, the statement "send A to port P" appears in Tester(P,E). Consider the following two cases:

- PPT(T,E) is null. In this case, "send A to port P" is the first statement in Tester(P,E). During an execution of M and Tester(E), the statement "send A to port P" causes the port-based synchronization problem to occur if and only if P:A is a valid input for one of the states entered before the head state of T.
- PPT(T,E) is not null. In this case, "send A to port P" in Tester(P,E) appears immediately after the statements generated for PPT(T,E). Thus, T's input symbol arrives at M only after the completion of PPT(T,E). During an execution of M and Tester(E), the statement "send A to port P" causes the port-based synchronization problem to occur if and only if P:A is a valid input for one of the states entered after the head state of PPT(T,E) and before the head state of T.

Definition. For a transition T in a transition sequence E of an FSM, its **port-predecessor interval**, denoted as PPI(T,E), is defined as follows:

- If T is a timeout transition, then PPI(T,E) is ε, where ε denotes the empty sequence..
- If T is not timeout transition and PPT(T,E) is null, then PPI(T,E) is the sequence of transitions in E before T.

- If T is not timeout transition and PPT(T,E) is not null, then PPI(T,E) is the sequence of transitions in E after PPT(T,E) and before T.

Notes:

- If T is the first transition in E, then PPT(T,E) = null and PPI(T,E) = ε.
- If PPT(T,E) is the immediate predecessor of T, then PPI(T,E) = ε.

Definition. Let T be a transition in a transition sequence E of an FSM M. T is said to be **port-synchronizable** for (E,M), if the following two conditions hold:

(a) either PPI(T,E) is ε, or PPI(T,E) is not ε and the input symbol of T is invalid for the head state of any transition in PPI(T,E).
(b) if T is not a timeout transition, then head(T) does not have a timeout transition.

Notes:

- If T is a timeout transition, then T is port-synchronizable for (E,M). The reason is that when head(T) becomes the current state and there are no input symbols available, then T is the only eligible transition.
- If T is a non-timeout transition and head(T) has a timeout transition, then T is not port-synchronizable for (E,M). The reason is that when head(T) becomes the current state, the timeout transition instead of T may be chosen for execution. However, this situation cannot occur if the timeout transition of a state is chosen only if the state does not have eligible non-timeout transitions. This property, referred to as "**timeout-last**", holds under the following assumptions:
  - In a single-processor environment, when the current state of an FSM waits on a timeout transition, the FSM has a context switch and later chooses the timeout transition only if the current state does not have eligible non-timeout transitions.
  - In a network environment, the network delay is negligible and thus the current state of an FSM chooses a timeout transition only if the current state does not have eligible non-timeout transitions.
- Assume that T is a non-timeout transition and is port-synchronizable for (E,M). The arrival of T's input symbol before head(T) becomes the current state of M does not create an additional eligible transition for the head state of any transition in PPI(T,E).

Definition. Let E be a transition sequence of an FSM M. E is said to be **port-synchronizable** for M if every transition in E is port-synchronizable for (E,M).

Consider the test sequence R2 = T1.T3.T4.T2.T3.T5 of M1. R2 is a postman tour of M1. Below we show the PPT and PPI of each transition in R2. Since T3 has two occurrences in R2, T3_1 and T3_2 denote the first and second occurrences of T3, respectively.

Transition T	T1	T3_1	T4	T2	T3_2	T5
PPT(T,R2)	null	null	T1	T4	T3_1	T2
PPI(T,R2)	ε	T1	T3_1	ε	T4.T2	T3_1

R2 is not pair-synchronizable, but it is port-synchronizable for M1. (Note that no pair-synchronizable postman tours of M1 exist.) By applying algorithm Port_Tester_Gen to R2, we obtain the following testers for ports P1, P2, and P3:

Tester(P1,R2)	Tester(P2,R2)	Tester(P3,R2)
----------------------------	----------------------------	---------------------------

(T1) send A to P1
(T1) receive X from P1
(T4) send A to P1
(T5) receive X from P1

(T4) receive Y from P2
(T2) send B to P2
(T2) receive Y from P2
(T5) send B to P2

(T3_1) send C to P3
(T3_1) receive Z from P3
(T3_2) send C to P3
(T3_2) receive Z from P3

Consider another test sequence R3 = T1.T3.T5.T2.T3.T4 of M1. R3 is also a postman tour of M1. Below we show the PPT and PPI of each transition in R3.

Transition T	T1	T3_1	T5	T2	T3_2	T4
PPT(T,R3)	null	null	null	T5	T3_1	T5
PPI(T,R3)	ε	T1	T1.T3_1	ε	T5.T2	T2.T3_2

R3 is not port-synchronizable for M1 since

- PPI(T5,R3) = T1.T3_1 and the input symbol of T5 is valid for head(T1).
- PPI(T4,R3) = T2.T3_2 and the input symbol of T4 is valid for head(T2).

Let a **timeout-free test sequence** of an FSM refers to a test sequence such that the head state of each transition in the sequence does not have a timeout transition. Below we show some properties of pair- and port-synchronizable tests sequences of an FSM.

Theorem 5.1. Let E be a timeout-free test sequence for an FSM M.

(a) E is pair-synchronizable if and only if for each transition T in E, PPI(T,E) is ε.

(b) If E is pair-synchronizable, then E is port-synchronizable for M. But the converse is not necessarily true.

(c) If M is completely specified, then E is port-synchronizable if and only if E is pair-synchronizable for M.

Proof. This theorem follows the definitions of pair- and port-synchronizable test sequences of an FSM. Q.E.D.

Theorem 5.2. Let E be a port-synchronizable test sequence of an FSM M and let Tester(E) be the set of testers generated by algorithm Port_Tester_Gen according to E.

(a) Any execution of M and Tester(E) is deterministic and successful. Thus, the port-based synchronization problem never occurs during any execution of M and Tester(E).

(b) At any time during an execution of M and Tester(E), two or more ports of M may contain input messages.

(c) If E is pair-synchronizable, then at any time during an execution of M and Tester(E), at most one port of M contains input messages.

(d) Assume that E does not contain timeout transitions. M does not have a port-synchronizable test sequence F such that $F \neq E$ and Tester(F)=Tester(E).

Proof.

(a) Our proof is based on induction on $|E|$, which is the number of transitions in E. First, we show that (a) is true for $|E| = 1$. When E contains only one transition, this transition is the only eligible transition for the initial state of M. Thus, any execution of M and Tester(E) is deterministic and successful. Now we assume that (a) is true for $|E| = k$. Let E be E'.T, where E' is a test sequence with $|E'| = k$. Consider the following two cases for T:

- T is a timeout transition. Since T has no input symbol, it has no effect on any execution of M and Tester(E) before M reaches the end of E'. After M has reached the end of E', head(T) becomes the current state of M and T is the only eligible transition.

- T is not a timeout transition. Let the input symbol of T be P:A. Consider the following two cases for PPT(T,E):
  - PPT(T,E) is null. According to algorithm Port_Tester_Gen, "send A to P" is the first statement in Tester(P,E). Since P:A is invalid for any of the states entered by M before the head state of T, message A at port P has no effect on any execution of M and Tester(E) before M enters the head state of T. When head(T) becomes the current state of M, message A at port T is the only valid input for head(T).
  - PPT(T,E) is not null. According to algorithm Port_Tester_Gen, Tester(P,E) contains "send A to P" for T, which immediately follows the send and/or receive statements for PPT(T,E). One of the following two cases holds for PPT(T,E):
    - PPT(T,E) has port P as an output port. In this case, "send A to P" for T is executed after the completion of PPT(T,E).
    - PPT(T,E) has port P as the input port and not as an output port. In this case, "send A to P" is executed after the "send ... to P" for PPT(T,E).

    Thus, "send A to P" for T has no effect on any execution of M and Tester(E) from the beginning of E to PPT(T,E). After the completion of PPT(T,E), input message A of T becomes the first input message at port P. Since P:A is invalid for any of the head states of transitions between PPT(T,E) and T, message A at port P has no effect any execution of M and Tester(E) before M enters the head state of T. When head(T) becomes the current state of M, message A at port T is the only valid input for head(T).

Following the above discussion, any execution of M and Tester(E) is deterministic and thus successful. Therefore, (a) is true for any port-synchronizable test sequence of M.

(b) Below we describe a situation that causes two or more ports of M to contain input messages during an execution of M and Tester(E). Let T be a transition in E such that PPT(T,E) is not null, PPI(T,E) is not ε, and the input symbol of T is P:A. Assume that during an execution of M and Tester(E), input message A of T arrives at port P before or immediately after the completion of PPT(T,E). Then the input queue of port P is non-empty during the completion of transitions in PPI(T,E). The transitions in PPI(T,E) have ports other than P as input ports. Thus, during the execution of transitions in PPI(T,E), the situation that two or more ports of M contain input messages occurs at least once.

(c) Assume that E is pair-synchronizable. For any transition T in E, PPI(T,E) is ε. Thus, for any transition T in E that is not the first transition, the immediate predecessor of T is PPT(T,E). During any execution of M and Tester(E),

- before the execution of the first transition in E, only the input port of the first transition may contain input messages.
- after the execution of a transition T in E and before the execution of T's immediate successor in E, only the input port of T's immediate successor may contain input messages.

Thus, at any time during this execution, at most one port of M contains input messages

(d) Assume that F is a port-synchronizable test sequence of M such that F≠E and Tester(F)=Tester(E). For i>0, let Ei and Fi be the ith transitions of E and F, respectively. Assume that the initial state of M is s0 and that the input symbols of E1 and F1 are P:A and Q:B, respectively. Consider each of the following two cases:

- P=Q. In this case, Tester(P,E) = Tester(P,F) and thus A=B. Because M is deterministic, we have E1=F1.
- P≠Q. In this case, "send A to P" is the first statement in Tester(P,F) and "send B to Q" the first statement in Tester(Q,F). Let Fj, j>1, be the first transition in F with P:A as the input symbol. Since "send A to P" is the first transition in Tester(P,F), PPT(Fj,F) is null and state s0 does not contain any transition with P:A as the input symbol. However, according to E, T1 is a transition of s0 and has P:A as the input symbol. Due to this contradiction, P≠Q can never hold.

Following the above discussion, we have E1=F1. By applying the same argument, we have Ei=Fi for any i>0, and consequently E=F. However, this is a contradiction to our assumption that F≠E. Thus, M does not contain a port-synchronizable test sequence F such that F≠E and Tester(F)=Tester(E). Q.E.D.

One interesting question is whether there exists a test sequence E of an FSM M such that E is not port-synchronizable for M, but any execution of M and Tester(E) is deterministic (and thus successful). The following theorem says such a test sequence of M does not exist. Based on this theorem, the definition of a port-synchronizable test sequence of M provides a necessary and sufficient condition under which a test sequence of M does not have the synchronization problem. To prove this theorem, we consider **the delayed execution** of M and its testers, which means that during an execution of M and its testers, the selection of a transition of the current state of M is delayed until (a) each tester for M either has completed or is blocking on a receive, and (b) all messages sent by these testers have arrived at ports of M. Obviously, the delayed execution of M and Tester(E) is deterministic if and only if any execution of M and Tester(E) is deterministic.

Theorem 5.3. Let E be a test sequence of an FSM M and let Tester(E) be the set of testers generated by algorithm Port_Tester_Gen according to E. E is port-synchronizable for M if and only if any execution of M and Tester(E) is deterministic.
Proof. Following Theorem 6.2(a), if E is port-synchronizable for M, then any execution of M and Tester(E) is deterministic. Assume that E is not port-synchronizable for M. There exists at least one transition, say T, in E such that T is not port-synchronizable for (E,M). Thus, PPI(T,E) is not ε and the input symbol of T is valid for the head state of a transition, say U, in PPI(T,E). During the delayed execution of M and Tester(E), when the current state of M is head(U) and a transition of head(U) is chosen after the input symbols of both T and U are available. Thus, the delayed execution of M and Tester(E) is nondeterministic. Q.E.D.

5.3. Generation of Port-Synchronizable Test Sequences of an FSM

In this section, we briefly discuss possible approaches to extending a test sequence generation method for an FSM in order to generate port-synchronizable test sequences. One approach is to apply the concept of **backtracking**. Assume that during the application of a test sequence generation method to an FSM M, E is the partial test sequence generated so far and transition T is allowed to follow E. We need to determine whether T is port-synchronizable for E.T with respect to M. If the answer is yes, then we append T to the end of E and continue the original test sequence generation procedure. If the answer is no, then we try to find another transition, say T’, that is allowed to follow E, and then we determine whether T’ is port-synchronizable for E.T’ with respect to M. We repeat this process until such a transition is found. If no such transition exists for E, then we replace the last transition in E with another transition and repeat the same process. Eventually either we find one port-synchronizable test sequence of M, or we conclude that no port-synchronizable test sequence of M exists, according to the original test sequence generation method. The concept of backtracking was also used in the generation of pair-synchronizable test sequences [SB84, LDB93].

Some test sequence generation methods involve the generation of two types of transition sequences for a state S of an FSM M:

- transfer sequences for S, which start from the initial state of M and reach at S.
- state identification or validation sequences for S, which start from S.

Assume that for a transition T of M, a test sequence generation method has been extended to generate

- a set FS of port-synchronizable transfer sequences for head(T), and
- a set DS of port-synchronizable identification or validation sequences for tail(T).

We need to find one element of FS, say F, and one element of DS, say D, such that F.T.D is port-synchronizable for M. To search for such a port-synchronizable test sequence, we have the following observations:

(a) For each element D of DS, let Null(D) be the set of transitions in D with their PPT being null. To determine whether F.T.D is synchronizable for M, we only need to determine whether T and transitions in Null(D) are synchronizable for (F.T.D, M). Transitions in D that are not in Null(D) can be ignored, since they can never create the port-base synchronization problem for F.T.D.

(b) Let U be the shortest suffix of F that involves the input ports of T and transitions in Null(D) at least once. To determine whether F.T.D is port-synchronizable for M, we only need to determine whether T and transitions in Null(D) are port-synchronizable for (U.T.D, M). The reason is that T and transitions in Null(D) have their PPT in U.

Based on (b), we have developed algorithms for generating UIO-based port-synchronizable test sequences of an FSM.

6. Group-Based Testing of an FSM

As mentioned earlier, group-based testing of an FSM divides the ports of this FSM into groups and allows the testers for ports in the same group to communicate with each other in order to synchronize the inputs from different testers. Using group-based testing can make more test sequences of an FSM to be synchronizable. Below we first define a group-based tester. In Section 6.1, we explain how to extend port-based testing to group-based testing and define a group-synchronizable test sequence of an FSM. In Section 6.2, we present an algorithm for generating group-based testers and define the group-based synchronization problem. In Section 6.3, we discuss the generation of group-synchronizable test sequences of an FSM.

A **group-def** for an FSM M defines how the ports of M are divided into groups. We assume that a tester has one port for communication with other ports in the same group. The **group-based tester** for port P of M according to a test sequence E of M and a group-def G for M, denoted as **Tester**(P,E,M,G), contains the following four types of statements:

- send A to P;
- receive B from P;
- send permit to Tester(Q,E,M,G);
- receive permit to Tester(Q,E,M,G);

where A is an input message of M, B an output message of M, "permit" a message used between testers in the same group, and Q another port in P's group in G. For the sake of simplicity, Tester(Q,E,M,G) is referred to as Tester(Q,E,G) if M is either implied or immaterial. Also, we use "Tester(Q,E,G)" instead of "the port of Tester(Q,E,G)" in the above send and receive statements involving message permit. Like a port-based tester, a group-based tester contains a sequence of non-blocking send and blocking receive statements.

6.1 Definition of a Group-Synchronizable Test Sequence of an FSM

We first use an example to illustrate our concept of group-based testing. Consider test sequence R3 = T1.T3.T5.T2.T3.T4 of M1. As mentioned earlier, R3 is not port-synchronizable for M1 since

- PPI(T5,R3) = T1.T3_1 and the input symbol of T5 is valid for head(T1).
- PPI(T4,R3) = T2.T3_2 and the input symbol of T4 is valid for head(T2).

By applying algorithm Port_Tester_Gen to R3, we obtain the following testers for ports P1, P2, and P3:

Tester(P1,R3)	Tester(P2,R3)	Tester(P3,R3)
(T1) send A to P1	(T5) send B to P2	(T3_1) send C to P3
(T1) receive X from P1	(T2) send B to P2	(T3_1) receive Z from P3
(T5) receive X from P1	(T2) receive Y from P2	(T3_2) send C to P3
(T4) send A to P1	(T4) receive Y from P2	(T3_2) receive Z from P3

Now we consider group-based testing of M1 according to group-def G1 for M1, which is defined as {(P1,P2}, {P3}). Since P1 and P2 are in the same group, we can insert statements into Tester(P1,R3) and Tester(P2,R3) to delay the arrivals of inputs of M at ports P1 and P2. The following testers are the above testers modified by insert statements to delay the delivery of the input of T5 until after the completion of T1 and to delay the delivery of the input of T4 until after the completion of T2. By doing so, the port-synchronization problems due to T5 and T4 are eliminated.

Tester(P1,R3,G1)	Tester(P2,R3,G1)	Tester(P3,R3,G1)
(T1) send A to P1	receive permit from Tester(P1,R3,G1)	(T3_1) send C to P3
(T1) receive X from P1	(T5) send B to P2	(T3_1) receive Z from P3
send permit to Tester(P2,R3,G1)	(T2) send B to P2	(T3_2) send C to P3
(T5) receive X from P1	(T2) receive Y from P2	(T3_2) receive Z from P3
receive permit from Tester(P2,R3,G1)	send permit to Tester(P1,R3,G1)	
(T4) send A to P1	(T4) receive Y from P2	

Now we formalize our concept of group-based testing. Let T be a transition in a test sequence E of an FSM M. If T is a timeout transition, then T has no synchronization problem with the input symbols of predecessors of T in E. If T is not a timeout transition, then the input symbol of T may have the synchronization problem with the input symbol of a predecessor of T in E (i.e., the two input symbols have a race condition) only if the head state of this predecessor contains a transition that is not the predecessor and has the same input symbol as T. Below we consider the closest predecessor of T in E that has this property.

Definition. For a transition T in a transition sequence E of an FSM M, **Race**(T,E,M) is defined as follows:

- If T is a timeout transition, then Race(T,E,M) is null.
- If T is not a timeout transition, then Race(T,E,M) is the closest predecessor U of T in E such that head(U) has a transition that is not U and has the same input symbol as T. If no such U exists, Race(T,E,M) is null.

One of the following three cases holds for Race(T,E,M):

(a) Race(T,E,M) is null.
(b) Race(T,E,M) is PPT(T,E) or a predecessor of PPT(T,E).
(c) Race(T,E,M) is a transition in PPI(T,E).

Assume that head(T) has no timeout transitions. T is port-synchronizable for (E,M) if and only if either PPI(T,E) is ε or PPI(T,E) is not ε and the input symbol of T is valid for the head state of some transition in PPI(T,E). In case (a) or (b), T is port-synchronizable for (E,M), and in case (c), T is not port-synchronizable for (E,M). Thus, T is port-synchronizable for (E,M) if and only if Race(T,E,M) is not a transition in PPI(T,E).

Lemma 6.1: Let T be a transition in a transition sequence E of an FSM M. T is not port-synchronizable for (E,M) if and only if

- Race(T,E,M) is a transition in PPI(T,E), or
- T is not a timeout transition and head(T) has a timeout transition.

Note: If head(T) has no timeout transtision or the timeout-last property holds, then T is port-synchronizable for (E,M) if and only if Race(T,E,M) is not a transition in PPI(T,E).

Assume that head(T) has no timeout transitions and that T is not port-synchronizable for (E,M). Based on the above lemma, Race(T,E,M) is a transition in E after PPT(T,E) and before T. Assume that the input port of T is P. For a given group-def G for M, consider the following three cases for Race(T,E,M):

(a) Race(T,E,M) has a port, say Q, in P's group in G as an output port. Since Race(T,E,M) is not PPT(T,E), Q≠ P. In this case, we can insert statements into Tester(P,E,G) and Tester(Q,E,G) to delay the delivery of the input of T until after the receipt of output messages of Race(T,E,M) at port Q. This is accomplished by inserting
    - "receive permit from Tester(Q,E,G)" in Tester(P,E,G) immediately before "send ... to P" for T, and
    - "send permit to Tester(P,E,G)" in Tester(Q,E,G) immediately after "receive .... from Q" for Race(T,E,M).

(b) Race(T,E,M) does not have any port in P's group in G as an output port and there exists a transition U in E after Race(T,E,M) and before T that has a port, say Q, in P's group in G as an output port. Since U is not PPT(T,E), Q≠ P. The following diagram shows the ordering of T, PPT(T,E), Race(T,E,M), and U in E.

PPT(T,E)   Race(T,E,M)   U   T

... ------------> ... --------------> .... ---> ... ---> ...

In this case, we can insert statements into Tester(P,E,G) and Tester(Q,E,G) to delay the delivery of the input of T until after the receipt of output messages of U at port Q.

(c) Otherwise (i.e., neither Race(T,E,M) nor any transition in E after Race(T,E,M) and before T has a port in P's group in G as an output port). In this case, the race condition between the input symbols of T and Race(T,E,M) cannot be eliminated by synchronization among testers for ports in P's group in G.

In port-based testing, the PPT and PPI of a transition T in a transition sequence E of an FSM M are used to determine whether T is port-synchronizable for (E,M). Based on the above discussion, we extend the concepts of PPT and PPI for group-based testing.

Definition. Let E be a transition sequence of an FSM M and let G be a group-def for M. For a transition T in E with port P as the input port, its **group-predecessor transition** according to G, denoted as **GPT**(T,E,G), is defined as follows:

- If T is a timeout transition, then GPT(T,E,G) is null.
- If T is not timeout transition, then GPT(T,E,G) is the closest predecessor of T in E that either has P as the input port or has any port in P's group in G as an output port. If such a predecessor of T in E does not exist, then GPT(T,E,G) is null.

Notes:

- If GPT(T,E,G) is null, then PPT(T,E) is null. But the converse is not true.
- If PPT(T,E) is not null, then GPT(T,E) is either PPT(T,E) or a successor of PPT(T,E) in E.
- If GPT(T,E,G) ≠ PPT(T,E), then GPT(T,E,G) is not null and has a port Q in P's group in G, Q ≠ P, as an output port.

Definition. Let E be a transition sequence of an FSM M and let G be a group-def for M. For a transition T in E, its **group-predecessor interval** according to G, denoted as **GPI**(T,E,G), is defined as follows:

- If T is a timeout transition, then GPI(T,E,G) is ε.
- If T is not timeout transition and GPT(T,E,G) is null, then GPI(T,E,G) is the sequence of transitions in E before T.
- If T is not timeout transition and GPT(T,E,G) is not null, then GPI(T,E) is the sequence of transitions in E after GPT(T,E,G) and before T.

Notes:

- If T is the first transition in E, then GPT(T,E,G) = null and GPI(T,E,G) = ε.
- If GPT(T,E,G) is the immediately predecessor of T, then GPI(T,E,G) = ε.
- GPI(T,E,G) is a suffix of PPI(T,E).
- If PPI(T,E) = ε, then GPT(T,E,G) = PPT(T,E).

Definition. Let T be a transition in a transition sequence E of an FSM M and let G be a group-def for M. T is said to be **group-synchronizable** for (E, M, G), or **G-synchronizable** for (E,M), if the following two conditions hold:

(a) GPI(T,E,G) is ε, or GPI(T,E,G) is not ε and the input symbol of T is invalid for the head state of any transition in GPI(T,E,G).

(b) If T is not a timeout transition, then head(T) does not have a timeout transition.

Notes:

- If T is a timeout transition, then T is G-synchronizable for (E,M).
- If T is a non-timeout transition and head(T) has a timeout transition, then T is not G-synchronizable for (E,M).
- Assume that either head(T) has no timeout transition or the timeout-last property holds.
  - T is G-synchronizable for (E,M) if and only if Race(T,E,M) is not a transition in GPI(T,E,G).
  - If G makes all ports of M as elements of the same group, then T is G-synchronizable for (E,M).

Definition. Let E be a transition sequence of an FSM M and let G be a group-def for M. E is said to be **group-synchronizable** for (M,G), or **G-synchronizable** for M, if every transition in E is G-synchronizable for (E,M).

Earlier we used test sequence R3 = T1.T3.T5.T2.T3.T4 of M1 with group-def G1 = {(P1,P2}, {P3}) to illustrate the concept of group-based testing. Below we show the PPT, PPI, Race, GPT and GPI of each transition in test sequence R3 of M1 with group-def G1.

Transition T	T1	T3_1	T5	T2	T3_2	T4
PPT(T,R3)	null	null	null	T5	T3_1	T5
PPI(T,R3)	ε	T1	T1.T3_1	ε	T5.T2	T2.T3_2
Race(T,R3,M1)	null	null	T1	null	null	T2
GPT(T,R3,G1)	null	null	T1	T5	T3_1	T2
GPI(T,R3,G1)	ε	T1	T3_1	ε	T5.T2	T3_2

For each transition T in R3, Race(T,E,M1) is either null or a transition not in GPI(T,R3,G1). So R3 is G1-synchronizable for M1. Following the above discussion, we have the following theorem.

Theorem 6.2. Let E be a timeout-free test sequence of an FSM M and let G be a group-def for M.

(a) If E is pair-synchronizable or port-synchronizable for M, then E is G-synchronizable for M. But the converse is not true.

(b) If G makes each port of M as the only element of a unique group, then E is port-synchronizable for M if and only if E is G-synchronizable for M.

(c) If G makes all ports of M as elements of the same group, then E is G-synchronizable for M and thus the problem of synchronizing inputs from different testers does not exist.

(d) E is G-synchronizable for M if and only if for each transition T in E, Race(T,E,M) is not a transition in GPI(T,E,G).

(e) If M is completely specified, then E is G-synchronizable for M if and only if for each transition T in E, GPI(T,E,G) is ε.

6.2 Group-Based Testers and Synchronization Problem

As mentioned earlier, a group-based tester for a test sequence of an FSM is a modification of the port-based tester for the test sequence by inserting statements to delay the arrivals of inputs from testers, if necessary, in order to synchronize the inputs from testers in the same group. To reduce overhead for synchronization, we keep the inserted synchronization statements to a minimum. (In Section 7, we show that reducing the number of inserted synchronization statements also provides better fault detection.) Let T be a transition in a timeout-free test sequence E of an FSM M and let G be a group-def for M. Assume that the input symbol of T is P:A. To delay the arrival of T's input symbol, consider the following cases for GPT(T,E,G).

(a) GPT(T,E,G) is null. In this case, the delay of "send A to P" for T is impossible. Note that in this case, PPT(T,E) is also null and thus GPT(T,E,G) = PPT(T,E).

(b) GPT(T,E,G) is not null and GPT(T,E,G) = PPT(T,E). In this case, the delay of "send A to P" for T is unnecessary, since the input message of T becomes the input message at port P only after the completion of PPT(T,E).

(c) GPT(T,E,G) is not null and GPT(T,E,G) ≠ PPT(T,E) (i.e., GPT(T,E,G) is a transition in PPI(T,E)). In this case, "send A to P" for T can be delayed until after the completion of GPT(T,E,G). Note that GPT(T,E,G) ≠ PPT(T,E) implies that GPT(T,E,G) is not null and has a port Q in P's group in G, Q ≠ P, as an output port.

Thus, the delay of "send A to P" for T is necessary and possible only if GPT(T,E,G) ≠ PPT(T,E). Also, the delay of "send A to P" for T is necessary only if T is not port-synchronizable for M, i.e., only if Race(T,E,M) is a transition in PPI(T,E). As an example, consider transition T3_1 in R3 of M1. GPT(T3_1,E,G1) is T1 and PPT(T3_1,E,G1) is null. Since Race(T3_1,R3,M1) is null, there is no need to delay the input symbol of T3_1.

Assume that GPT(T,E,G) ≠ PPT(T,E) and Race(T,E,M) is a transition in PPI(T,E). Consider the following relationships between Race(T,E,M) and GPT(T,E,G):

- Race(T,E,M) is GPT(T,E,G). In this case, T is G-synchronizable for (E,M) and "send A to P" for T is delayed until after the completion of GPT(T,E,G).
- Race(T,E,M) is a successor of GPT(T,E,G). In this case, T is not G-synchronizable for (E,M). We assume that this does not stop the construction of group-based testers for M. Thus, "send A to P" for T is delayed until after the completion of GPT(T,E,G).
- Race(T,E,M) is a predecessor of GPT(T,E,G). In this case, "send A to P" for T could be delayed until after the completion of GPT(T,E,G). However, it is possible to allow earlier arrival of the input symbol of T and still avoid the race condition between the inputs of Race(T,E,M) and T. (In Section 7, we show that earlier arrivals of inputs from testers provides better fault detection, if the earlier arrivals of inputs do not create race conditions.) Consider the following cases:
    - Race(T,E,M) has an output port in P's group in G. In this case, "send A to P" for T is delayed until after the completion of Race(T,E,M). As an example, consider transition T5 in R3 of M1. GPT(T5,E,G1) is T3_1 and Race (T5,E,G1) is T1, which is a predecessor of T5. The delivery of the input symbol of T5 is delayed until after the completion of T1, not T3_1.
    - Race(T,E,M) does not have an output port in P's group in G. In this case, we search for the first transition in E after Race(T,E,M) that has a port in P's group in G as an output port. Let this transition be U. U must exist, since GPT(T,E,M,G) has a port in P's group in G as an output port. (Thus, U is either GPT(T,E,G) or a predecessor of GPT(T,E,G) in E.) In this case, "send A to P" for T is delayed until after the completion of U.

Following the above discussion, we have the following definition.

Definition. Let T be a transition in a transition sequence E of an FSM M and let G be a group-def for M. Assume that the input port of T is P. **Delay**(T,E,M,G) is defined as follows:

(a) if GPT(T,E,G) $\neq$ PPT(T,E) and Race(T,E,M) is a transition in PPI(T,E), then
  (a.1) if Race(T,E,M) is GPT(T,E,G) or a successor of GPT(T,E,G), then Delay(T,E,M,G) is GPT(T,E),
  (a.2) otherwise (i.e., Race(T,E,M) is a predecessor of GPT(T,E,G)),
    (a.2.2) if Race(T,E,M) has an output port in P's group in G, then Delay(T,E,M,G) is Race(T,E,M),
    (a.2.2) otherwise, Delay(T,E,M,G) is the first transition in E after Race(T,E,M) that has a port in P's group in G as an output port.
(b) otherwise, Delay(T,E,M,G) is null.

Notes:

- If T is a timeout transition, both PPT(T,E) and GPT(T,E,G) are null and thus Delay(T,E,M,G) is null.
- Delay(T,E,M,G) is not null if and if GPT(T,E,G) $\neq$ PPT(T,E) and Race(T,E,M) is a transition in PPI(T,E).
- If Delay(T,E,M,G) is not null, then it is either Race(T,E,M,G), GPT(T,E,G), or a transition in E between them.
- If T is port-synchronizable for (E,M), then Delay(T,E,M,G) is null. If Delay(T,E,M,G) is null and T is G-synchronizable for M, then T is port-synchronizable for (E,M).
- Assume that T is G-synchronizable for (E,M). Delay(T,E,M,G) is not null if and only if the delay of the input symbol of T is necessary and possible.

Algorithm Group_Tester_Gen

```
Input:   a test sequence E of an FSM M and a group-def G for M.
Output:  Tester(E,G), which is {Tester(P,E,G) | P is a port involved in E}
(1)  For each port P involved in E, let Tester(P,E,G) be empty.
(2)  For each transition T in E, let Syn(T) be {(R,W) | T = Delay(U,E,M,G) for some transition
     U in E, W is the input port of U, and R is the first output port of T that is in W's group
     in G}. Initially, Syn(T) is set to be the empty set.
(3)  For each transition T in E  (starting from the first transition in E),
     (a)  compute Delay(T,E,M,G).
     (b)  If Delay(T,E,M,G) is not null,
          (b.1)  assume that the input port of T is P.
          (b.2)  let Q be the first output port of Delay(T,E,M,G) that is in P's group in G.
          (b.3)  Syn(Delay(T,E,M,G)) := Syn(Delay(T,E,M,G)) ∪ {(Q,P)}.
 (3) For each transition T in E (starting from the first transition in E) ,
     (a)  if the input symbol of T is P:A, then add "send A to P" to the end of Tester(P,E,G).
     (b)  if the list of output symbols of T is (Q1:B1,,Q2:B2,...,Qv:Bv) then
          for each i from 1 to v, add "receive Bi from Qi" to the end of Tester(Qi,E,G).
     (c)  if Syn(T) is not empty, then for each (R,W) in Syn(T),
          (d.1)  add "send permit to Tester(W,E,G)" to the end of Tester(R,E,G).
          (d.2)  add "receive permit from Tester(R,E,G)" to the end of Tester(W,E,G).
end of algorithm.
```

Earlier we showed the PPT, PPI, Race, GPT and GPI of each transition in test sequence R3 of

M1 with group-def G1. All transitions in R3 except T5 and T4 have their Delay for G1 being null. Delay(T5,R3,M1,G1) = T1, and Delay(T4,R3,M1,G1) = T2. Tester(P1,R3,G1), Tester(P2,R3,G1), and Tester(P3,R3,G1) shown in Section 6.1 are exactly the testers generated by algorithm Group_Tester_Gen for M1 according to R3 and G1.

Now we consider group-based testers for R3 of M1 with group-def G2 = {(P1,P2,P3)}. Since G2 makes all ports of M1 in the same group, R3 is G2-synchronizable for M1. Below we show the PPT, PPI, Race, GPT, GPI, and Delay of each transition in test sequence R3 of M1 with group-def G2.

Transition T	T1	T3_1	T5	T2	T3_2	T4
PPT(T,R3)	null	null	null	T5	T3_1	T5
PPI(T,R3)	ε	T1	T1.T3_1	ε	T5.T2	T2.T3_2
Race(T,R3,M1)	null	null	T1	null	null	T2
GPT(T,R3,G2)	null	T1	T3_1	T5	T2	T3_2
GPI(T,R3,G2)	ε	ε	ε	ε	ε	ε
Delay(T,R3,M1,G2)	null	null	T1	null	null	T2

All transitions in R3 except T5 and T4 have their Delay for G2 being null. For each of T5 and T4, it has the same Delay for both G1 and G2. Thus, according to algorithm Group_Tester_Gen, Tester(P1,R3,G1), Tester(P2,R3,G1), and Tester(P3,R3,G1) are the same as Tester(P1,R3,G2), Tester(P2,R3,G2), and Tester(P3,R3,G2), respectively.

Let E be a transition sequence of an FSM M and let G and G' be two group-def for M such that each group in G is a subset of a group in G'. Assume that E is G-synchronizable for M. Let T be a transition in E. Since T is G-synchronizable for (E,M), Race(T,E,M) is not a transition in GPI(T,E,G). Assume that the input port of T is in group R of G and in group R' of G'. Since R' is a superset of R, GPI(T,E,G') is a suffix of GPI(T,E,G) and thus Race(T,E,M) is not a transition in GPI(T,E,G'). Hence, T is G'-synchronizable for (E,M). Consider the following cases for Delay(T,E,M,G):

- Delay(T,E,M,G) is not null. In this case, GPT(T,E,G) ≠ PPT(T,E) and Race(T,E,M) is a transition in PPI(T,E). Since GPI(T,E,G') is a suffix of GPI(T,E,G), GPT(T,E,G') ≠ PPT(T,E) and thus Delay(T,E,M,G') is not null. Furthermore, because Delay(T,E,M,G) is either Race(T,E,M,G), GPT(T,E,G), or a transition in E between them, Delay(T,E,G) is also Delay(T,E,G').
- Delay(T,E,M,G) is null. Since T is G-synchronizable for M, then T is port-synchronizable for (E,M). Therefore, Delay(T,E,M,G') is null.

Following the above discussion, we have the following theorem.

Theorem 6.3. Let E be a transition sequence of an FSM M and let G and G' be two group-def for M such that each group in G is a subset of a group in G'. Assume that E is G-synchronizable for M. Then

- E is G'-synchronizable for M.
- For each transition T in E, Delay(T,E,M,G) = Delay(T,E,M,G').
- Tester(E,G) = Tester(E,G').

Definition. Let E be a test sequence of an FSM M and let G be a group-def for M. Let Tester(E,G) be the set of testers generated by algorithm Group_Tester_Gen according to E and G. During an execution of M and Tester(E,G), the **group-based synchronization problem** occurs when a state of M has two or more eligible transitions. (As a result of this problem, this execution of M and Tester(E,G) is nondeterministic and may result in an abnormal termination.)

Theorem 6.4. Let E be a test sequence of an FSM M, G a group-def for M, and Tester(E,G) the set of testers generated by algorithm Group_Tester_Gen according to E and G. Assume that E is G-synchronizable for M.

(a) If E is port-synchronizable, then Tester(E,G) = Tester(E), which is the set of testers generated by algorithm Port_Tester_Gen according to E.

(b) Any execution of M and Tester(E,G) is deterministic and successful. Thus, the group-based synchronization problem never occurs during any execution of M and Tester(E,G).

(c) The number of send and receive statements involving message permit in Tester(E,G) is minimum. That is, there does not exist a set S of group-based testers for M with group-def G such that
- any execution of M and S is deterministic and successful, and
- S contains less number of send and receive messages involving message permit.

(d) At any time during an execution of M and Tester(E,G), two or more ports of M may contain input messages.

(e) If G makes all ports of M as elements of the same group, then at any time during an execution of M and Tester(E,G), at most one port of M contains input messages.

(f) Assume that E does not contain timeout transitions. M does not have a timeout-free G-synchronizable test sequence such that F≠E and Tester(F,G)=Tester(E,G).

Proof.

(a) Assume that E is port-synchronizable for M. For each transition T in E, Delay(T,E,M,G) is null. Thus, Tester(E,G) does not contain send and receive statements involving message permit.

(b) The proof is similar to as that for Theorem 5.2(a) except that for transition T in E=E'.T, we consider the following three cases for Delay(T,E,M,G) and PPT(T,E):
- Delay(T,E,M,G) is not null.
- Delay(T,E,M,G) is null and PPT(T,E) is not null.
- Both Delay(T,E,M,G) and PPT(T,E) are null.

In each of the above cases, according to algorithm Group_Tester_Gen, the input symbol of T has no effect on any execution of M and Tester(E,G) from the beginning to the end of E'. After M enters the head state of T, the input symbol of T is the only valid input for this state. Thus, any execution of M and Tester(E,G) is deterministic and successful

(c) For each transition T in E, if the insertion of send and receive statements involving message permit is needed, then "receive permit from ..." for T must be inserted immediately before "send ... to ..." for T, and "send permit to ..." for T must be inserted immediately after "receive ... from ..." for either Race(T,E,M), GPT(T,E,G), or a transition after Race(T,E,M) and before GPT(T,E,M). The inserted synchronization statements for T cannot be used as inserted synchronization statements for another transition in E. According to algorithm Group_Tester_Gen, Delay(T,E,M,G) is not null if and only if the insertion of send and receive statements involving message permit for T is necessary. Therefore, the number of send and receive statements involving message permit in Tester(E,G) is minimum.

(d) It follows Theorem 5.2(b).

(e) The proof is the same as that for Theorem 5.2(c) except that for any transition T in E, GPT(T,E,G) is ε. Thus, for any transition T in E that is not the first transition, the immediate predecessor of T is either PPT(T,E) or Delay(T,E,M,G).

(f) The proof is similar to that for Theorem 5.2(d) and is omitted. Q.E.D.

One interesting question is that for an FSM M and a group-def G for M, whether there exists a test sequence E of M such that E is not G-synchronizable for M, but any execution of M and Tester(E,G) is deterministic. The following theorem answers this question. To prove this theorem, we consider **the delayed execution** of M and Tester(E,G). Obviously, the delayed execution of M and Tester(E,G) is deterministic if and only if any execution of M and Tester(E,G) is deterministic.

Theorem 6.5. Let E be a test sequence of an FSM M, G a group-def for M, and Tester(E,G) the set of testers generated by algorithm Group_Tester_Gen according to E and G. E is G-synchronizable for M if and only if any execution of M and Tester(E,G) is deterministic.
Proof. Following Theorem 6.3(b), if E is G-synchronizable for M, then any execution of M and Tester(E,G) is deterministic. Assume that E is not G-synchronizable for M. Following Theorem 6.2(d), there exists at least one transition, say T, in E such that Race(T,E,M,G) is a transition in GPI(T,E,G). Let Race(T,E,M,G) be transition U. During the delayed execution of M and Tester(E,G), when the current state of M is head(U) and a transition of head(U) is to be chosen, the input symbols of both T and U are available and valid. Thus, the delayed execution of M and Tester(E,G) is nondeterministic. Q.E.D.

6.3 Generation of Group-Synchronizable Test sequences of an FSM

In Section 5.3, we briefly discussed possible approaches to extending a test sequence generation method for an FSM in order to generate port-synchronizable test sequences. These approaches can also be applied to extend a test sequence generation method for an FSM to generate group-synchronizable test sequences. For a transition T in a test sequence E of an FSM M, T is port-synchronizable for (E,M) if and only if (a) Race(T,E,M) is not a transition in PPI(T,E) and (b) if T is not a timeout transition, then head(T) does not have a timeout transition. For a given group-def G for M, T is G-synchronizable for (E,M) if and only if (a) Race(T,E,M) is not a transition in GPI(T,E,G) and (b) if T is not a timeout transition, then head(T) does not have a timeout transition. Since GPI(T,E,G) is a suffix of PPI(T,E). the determination of whether T is G-synchronizable for (E,M) does not require any information about the transitions in E before PPI(T,E). In other words, the determination of whether T is G-synchronizable for (E,M) does not require any information about the transitions in E that are not needed for determination of whether T is port-synchronizable for (E,M).

7. Fault Detection by Synchronizable Test Sequences of an FSM

In earlier discussion, we made the following claims regarding fault detection of synchronizable tests sequences of an FSM M:

(a) “One tester for each port of M” is more effective than “one tester for all ports of M” for fault detection.

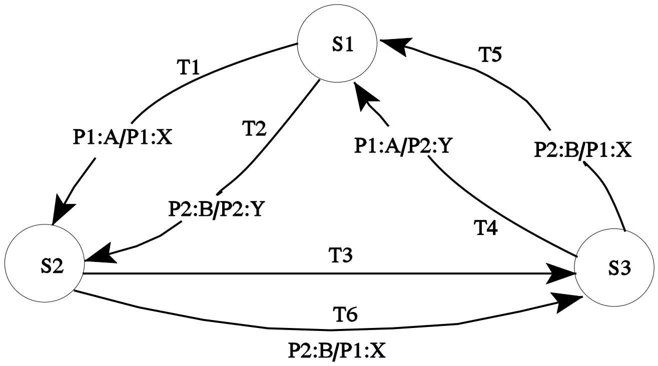
(b) "One tester for each port of M" is more effective than "one tester for each group of ports of M" for fault detection.

(c) Reducing the number of unnecessary send and receive statements involving message permit in testers for M provides better fault detection.

(d) Earlier arrivals of inputs from testers for M provides better fault detection, if the earlier arrivals of inputs do not create race conditions.

In this section, we give examples to illustrate the above claims. In fact, claims (a), (b), and (c) are based on claim (d). Other issues on fault detection of synchronizable tests sequences of an FSM are not considered in this paper.

Assume that FSM M1 in Fig. 1 has been implemented incorrectly as the FSM M2 in Fig. 2. M2 is the same as M1 except that M2 contains an extra transition T6, which is from state S2 to state S3 and has the same input and output symbols as T5.



Consider test sequence R4 = T1.T3.T5 of M1. R4 is not port-synchronizable for M1. Consider the following three sets of testers for R4:

- By using one tester for all ports of M1, the following is Tester(R4) for R4.

  (T1) send A to P1
  (T1) receive X from P1
  (T3) send C to P3
  (T3) receive Z from P3
  (T5) send B to P2
  (T5) receive X from P1

- Tester(R4,G2), according to algorithm Group_Tester_Gen, contains the following testers:

Tester(P1,R4,G2)	Tester(P2,R4,G2)	Tester(P3,R4,G2)
(T1) send A to P1	receive permit from Tester(P1,R4,G2)	(T3) send C to P3
(T1) receive X from P1	(T5) send B to P2	(T3) receive Z from P3
send permit to Tester(P2,R4,G2)		
(T5) receive X from P1		

- Tester(R4,G2)' is Tester(R4,G2) modified by inserting additional send and receive statements involving permit so that the input symbol of a transition is sent by a tester only when the transition is the next that is expected to be executed.

```
        Tester(P1,R4,G2)'
--------------------------------------
(T1) send A to P1
(T1) receive X from P1
send permit to Tester(P3,R4,G2)
(T5) receive X from P1
```

```
        Tester(P2,R4,G2)'
---------------------------------------------
receive permit from Tester(P3,R4,G2)
(T5) send B to P2
```

```
        Tester(P3,R4,G2)'
---------------------------------------------
receive permit from Tester(P1,R4,G2)
(T3) send C to P3
(T3) receive Z from P3
send permit to Tester(P2,R4,G2)
```

Note that both Tester(R4) and Tester(R4,G2)' send the input symbol of a transition in R4 only when the transition is the next that is expected to be executed. Any execution of M2 with either Tester(R4) or Tester(R4,G2)' is deterministic and successful. Thus, Tester(R4) and Tester(R4,G2)' can never detect the fault in M2. During an execution of M2 and Tester(R4,G2), when S2 is the current state, assume that both P3:C (for T3 in M1) and P2:B (for T5 in M1) are available. Thus, both T3 and T6 are eligible for S2 in M2. If T3 is chosen, then the execution is successful. However if T6 is chosen, then the execution is unsuccessful, since the next state S3 has no valid input. The reason for possible detection of the fault in M2 is that Tester(R4,G2) allows earlier arrival of the input symbol of T5.

8. Results of Empirical Studies

In this section, we describe the results of our empirical studies on pair- and port-synchronizable test sequences. We used a number of FSM-based protocols in our empirical studies. For each FSM-based protocol M, we applied the following steps:

(a) used **tsg**, a test sequence generation tool, to generate a set S of test sequences of M.
(b) for each test sequence E in S, determined whether E is pair-synchronizable, not pair-synchronizable, but port-synchronizable for M, or not port-synchronizable for M.

The tsg (*t*est *s*equence *g*eneration) tool was obtained from the Department of Computer Science at University of British Columbia. For a given FSM, tsg can generate test sequences according to the D-method [Koh78], UIO-method [SD88], and W-method [Cho78]. For the input FSM, the tool inserts a reset transition for each state. If the input FSM is not completely specified, the tool inserts a self-loop transition for each unspecified input of a state. Each of the inserted transitions has no output.

We applied tsg to generate UIO-based test sequences for a number of FSM-based protocols. For each transition T in a given FSM, tsg generated one or more UIO-based test sequences, each starting with a reset transition, followed by u.T.v, where u is a transition sequence from the initial state to the head state of T and v is a UIO sequence for the tail state of T. After a set S of UIO-based test sequences for an FSM was generated by tsg, S was modified as follows:

(a) The test sequences in S for inserted transitions (i.e., reset transitions and the self-loop transitions for unspecified inputs) were deleted.
(b) For each of the remaining test sequences in S, the inserted transitions in this test sequence, if they existed, were deleted, and if the resulting test sequence contained zero or one transition, then this test sequence was deleted.
(c) After step (b), if two or more test sequences in S were identical, then only one of them was kept in S.

The following algorithm was used to determine whether a test sequence is pair-synchronizable, and if not, whether it is port-synchronizable for a given FSM M.

```
let E = E1.E2.....En be a test sequence of M;
pair_flag := true;        /* to indicate whether E is pair-synchronizable */
port_flag := true;    /* to indicate whether E is port-synchronizable for M */
for i = 2, 3, ..., n until port_flag = false do begin
    visit transition Ei;
    if Ei is a timeout transition (i.e., the input symbol of Ei is null) then
        begin    pair_flag := false;    port_flag := true;        cycle;    end;
        /* the cycle statement completes the current iteration of the for loop */
    if the head state of Ei has a timeout transition then
        begin    pair_flag := false;    port_flag := false;    cycle;    end;
    let the input symbol of Ei is P:A;
    if port P is not involved in E(i-1) then begin
        pair_flag := false;
        ppt_found := false;  /* to indicate whether PPT(Ei,E) has been found */
        for j = i-1, i-2, ..., 1 until port_flag = false or ppt_found = true do begin
            visit transition Ej;
            if port P is involved in Ej
            then  ppt_found := true;
            else  if P:A is a valid input for head(Ej) then port_flag := false;
        end;      /* end of the loop for j */
    end;
end;              /* end of the loop for i */
```

After the completion of the above algorithm, if pair_flag is true, then E is pair-synchronizable. Otherwise, if port_flag is true, then E is not pair-synchronizable, but port-synchronizable for M. If port_flag is false, then E is not port-synchronizable for M. Note that if the timeout-last property holds, then the statement “if the head state of Ei has a timeout transition then ...” in the above algorithm is deleted.

The following table shows some of the results of our empirical studies.

protocol name test_seq_# pair_# port_# non_port_#

T-class-0	55	45 (82%)	52 (95%)	3 (5%)
T-class-4	107	78 (73%)	83 (78%)	24 (22%)
Q931	50	20 (40%)	48 (96%)	2 (4%)
T-timeout	25	10 (40%)	12 (48%)	13 (52%)

where

- test_seq_# is the total number of UIO-based test sequences generated by tsg for the corresponding protocol.
- pair_# is the number of pair-synchronizable test sequences.
- port_# is the number of port-synchronizable test sequences.
- non_port_# is the number of non-port-synchronizable test sequences.
- T-class-0 refers to the transport class 0 protocol [SB84]. This protocol has 4 states and 21 transitions.
- T-class-4 refers to the transport class 4 protocol [SL89], which has 15 states and 60 transitions.
- Q931 refers to the ISDN Q931 protocol [ZC94], which has 8 states and 31 transitions.
- Both T-class-4 and Q931 contain multiple types of timeout signals, which were treated as inputs from a unique port in our empirical studies.
- T-timeout refers to the transport protocol in Fig. 2.9 of [Sar93] modified by deleting one timeout transition, since the head state of this timeout transition has another timeout transition. The T-timeout protocol has 6 states, 15 non-timeout transitions and one timeout transition. If the timeout-last property holds, then the number (percentage) of port-synchronizable test sequences for this protocol increases from 12 (48%) to 23 (92%).

The above table show that for the set of UIO-based test sequences generated by tsg for a protocol, the percentage of pair-synchronizable test sequences ranges from 40% to 82% and that the percentage of port-synchronizable test sequences ranges from 78% to 96%. The increase of percentage of synchronizable test sequences due to the definition of a port-synchronizable test sequence ranges from 5% to 56%. These results indicate that the use of port-synchronizable test sequences may significantly increase the number of synchronizable test sequences of an FSM.

9. Conclusions

In this paper, we have discussed several issues on port- and group-based testing of an FSM with multiple ports, including the definitions of port- and group-synchronizable test sequence, the generation of port- and group-based testers, and approaches to generating port- and group-synchronizable test sequences. Port-synchronizable test sequences of an FSM have fewer restrictions than pair-synchronizable test sequences. Our empirical studies show that an FSM may contain many port-synchronizable test sequences that are not pair-synchronizable. Group-based testing is more general than port-based testing and makes more test sequences to become synchronizable.

According to Theorem 5.3, the set of port-synchronizable test sequences of an FSM M is exactly the set of test sequences of M that do not have the synchronization problem in port-based testing of M. Similarly, according to Theorem 6.4, the set of group-synchronizable test sequences of an FSM M with a group-def G is exactly the set of test sequences of M that do not have the synchronization problem in group-based testing of M with group-def G. Furthermore, according to Theorem 6.3(c), the number of inserted send and receive statements for synchronization in the group-based testers for M is minimum. Therefore, we have provided very interesting results about synchronizable test sequences of FSMs.

Our definitions of port- and group-synchronizable test sequences of an FSM are based on executions of the FSM and its testers, not executions of an implementation of the FSM and its testers. In this paper, we have briefly addressed some issues related to fault detection by synchronizable test sequences of an FSM. We are investigating additional issues related to fault detection. In this paper, we have also discussed several approaches to generating port- and group-synchronizable test sequences of an FSM. Currently we are developing algorithms for the generating synchronizable UIO test sequences.

Acknowledgments: The authors wish to thank Dan Duvarney for his effort in carrying out empirical studies. Also, the authors are grateful to Pramod Koppol for his comments and to Prof. S. T. Chanson and J. Zhu for providing the tsg tool.

References

[And91] G. Andrews, *Concurrent Programming: Principles and Practice*, Benjamin Cummings Pub. Co., Inc., Redwood, CA, 1991.

[BU91] S. Boyd and H. Ural, "The synchronization problem in protocol testing and its complexity," Information Processing Letters, Vol. 40, Nov. 1991, 131-136.

[CT91] R. Carver and K. C. Tai, "Replay and Testing for Concurrent Programs," IEEE Software, Vol. 8. No. 2, March 1991, 66-74.

[CLC90] W. H. Chen, C. S. Lu, L. Chen, and J.T . Tang, "Synchronizable protocol test generation via the duplex technique," Proc. IEEE INFOCOM '90, 561-563.

[CU95] W. H. Chen and H. Ural, "Synchronizable test sequences based on multiple UIO sequences," *IEEE/ACM Trans. Network*, Vol.3, No. 2, April 1995 152-7.

[CTU93] W. H. Chen, C. Y. Tang, and H. Ural, "Minimum-cost synchronizable test sequence generation via the duplexU digraph," Proc. IEEE INFOCOM '93, 128-135.

[EJ73] J. Edmons and E. L. Johnson, "Matching, Euler tours, and the Chinese postman," Math. Programming 5 (1973), 88-124.

[GU95] S. Guyot and H. Ural, "Synchronizable checking sequences based on UIO sequences," to appear in Proc. Protocol Test Systems VIII, 1995.

[Koh78] Z. Kohavi, Switching and Finite Automata Theory. 2nd edition, McGraw-Hill, 1978.

[LDB93] G. Luo, R. Dssouli, G. v. Bochmann, P. Venkataram, and A. Ghedamsi, "Generating synchronizable test sequences based on finite state machine with distributed ports," Proc. Protocol Test Systems VI, 1993, 139-153.

[NT81] S. Naito, and M. Tsunoyama, "Fault detection for sequential machines by transition tours," Proc. IEEE Fault Tolerant Computing Symp., 1981, 238-243.

[Sar93] B. Sarikaya, *Principles of Protocol Engineering and Conformance Testing*, Ellis Horwood Limited, 1993.

[SB84] B. Sarikaya and G. v. Bochmann, "Synchronization and specification issues in protocol testing," IEEE Trans. on Communications, Vol. 32, No.4, Apr. 1984, 389-395.

[SD88] K. Sabnani and A. Dahbura, "A protocol test generation procedure," Computer Networks and ISDN Systems, Vol. 15, no. 4, 1988, 285-297.

[SL89] D. P. Sidhu and T-K Leung, "Formal Methods for protocol testing: A detailed study," IEEE Trans. on Software Engineering., Vol. 15, no. 4, April 1989, 413-426.

[TA87] K. C. Tai, and S. Ahuja, "Reproducible testing of communication software", Proc. IEEE Inter. Conf. on Computer Software and Applications (COMPSAC) '87, Oct. 1987, 331-337.

[TCO91] K. C. Tai, R. H. Carver, and E. E. Obaid, "Debugging concurrent Ada programs by deterministic execution," IEEE Trans. Soft. Eng., Vol. 17, No. 1, Jan., 1991, 45-63.

[TC95] Tai, K. C., and Carver, R. H., "Testing of Distributed Programs", a chapter in *Handbook of Parallel and Distributed Computing*, ed. A. Zoyama, to be published by McGraw-Hill, 1995.

[Tar91] K. Tarnay, *Protocol Specification and Testing*, Plenum Press, 1991.

[Tur93] K J., Turner, ed., *Using Formal Description Techniques: An Introduction to Estelle, Lotos, and SDL*, Wiley, 1993.

[UW93] H. Ural and Z. Wang, "Synchronizable test sequence generation using UIO sequences," Computer Communications, Vol. 16, No. 10, Oct. 1993, 653-661.

[ZC94] J. Zhu and S. T. Chanson, "Toward evaluating fault coverage of protocol test sequences," Proc. Protocol Specification, Testing, and Verification, XIV, 1994 130-144.