Verification of Protocol Conformance Test Cases Using Reachability Analysis

Kshirasagar Naik
Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada H3G1M8

Behcet Sarikaya
Bilkent University, Department of Computer Engineering and Information Sciences, Bilkent, Ankara 06533, Turkey

A methodology is presented to verify manually written test cases against the formal specification of a protocol. Initially, a protocol and a test case are modeled as nondeterministic finite state machines and test case verification is viewed as a reachability analysis problem. An existing reachability analysis algorithm, based on the well-known perturbation technique, is modified to take nondeterminism in protocols and special test case features (timeouts and OTHERWISE events) into account. Correctness aspects of the reachability algorithm are proved. The notion of a synchronization error manifesting in a test case due to the nondeterministic nature of a protocol specification is studied. To verify data flow aspects of test cases, we extend our technique by modeling the test case and protocol specification as extended finite state machines. A test case from a proprietary test suite for the transport protocol Class 2 is taken as an example and is shown to contain several design errors.

1. INTRODUCTION

The objective of conformance testing of a communication protocol implementation is to verify whether the implementation conforms with the protocol specification in representative instances of communication [1]. In practice, conformance testing is done by applying a collection of test cases to the implementation under test (IUT) and observing the responses of the IUT.

The behavior of a protocol providing a set of communication functions is marked by sequences of events appearing at its service access points and composed of input events to the protocol and the responses of the protocol in the form of output events. From the conformance point of view, the objective of a test case is to check whether the implementation satisfies the specification requirements with respect to a protocol function. A test case does this checking by applying sequences of input events to the implementation and comparing the responded events with the expected events allowed by the protocol specification. If the behavior of the implementation is allowed by the protocol specification and the test objective is satisfied, then the test case assigns a *Pass* verdict. If the behavior of the implementation is not allowed by the protocol specification, then the test case assigns a *Fail* verdict. However, if the behavior of the implementation is allowed by the protocol but the test objective is not fulfilled, then the test case assigns an *Inconclusive* verdict. Therefore, the correctness of conformance judgement of a test case depends on the correctness of the sequences of events input to the implementation and the expected protocol events stated in the test case. Verification of a test case involves comparing the test case behavior with the protocol behavior and discovering any design errors in the test case.

Large size, nondeterministic events, combinations of many parameters, and communication among the modules of a protocol specification make the behavior of a communication protocol very complex and difficult to understand. Hence, manually designing a set of test cases to test an implementation of a protocol is not a trivial task. Those manually written test cases may contain many design errors. Thus, it is essential to have a methodology to verify test cases against a protocol specification.

Address correspondence to Prof. Behcet Sarikaya, Bilkent University, Dept. of Computer Engineering and Information Sciences, Bilkent, Ankara 06533, Turkey.

0164-1212/92/$5.00

Since the late 1970s, when researchers developed techniques to formally specify communication protocols, the notions of protocol validation and verification have been widely studied [2]. The techniques used in validating a protocol are reachability analysis, duologue-matrix analysis, and symbolic execution [3–6]. Similarly, a number of techniques have been used to verify protocols depending on whether the protocols are specified using finite state machines (FSMs), programming languages, or temporal logic. A list of references to various protocol verification techniques appears in [7] which discusses an algorithmic technique for verifying protocols described using FSMs and propositional temporal logic. In this article, we study the test case verification problem using a reachability analysis technique.

An outline of the verification methodology presented here is as follows. In the test verification system, a test case and a protocol specification are modeled as nondeterministic FSMs, called T-FSM and P-FSM, respectively. Each point of control and observation (PCO) is modeled as two unidirectional channels. The verification process consists of two phases. In the first phase, a perturbation technique is used to generate all the reachable global states in the verification system, where each global state consists of the states of the individual FSMs and the channel states. In the second phase, the global state space is analyzed for various errors, such as unspecified reception, tempoblocking, deadlock, livelock, channel overflow, and synchronization errors.

Real protocols have complicated functionalities with higher expressive power in the form of a set of parameter values associated with each protocol event and a capability to check the appropriateness of the received parameters using predicates. To demonstrate the application of the reachability analysis technique in verifying test cases against real protocols, we model a test case and a protocol specification by extended finite state machines (EFSM).

The rest of the article is organized as follows. In section 2, we model a local single-layer (LS) [1] architecture–based test case and the formal description of a protocol as nondeterministic FSMs and use a perturbation algorithm to generate the global state space, which is then analyzed to detect various test design errors. Synchronization errors occurring in a test case due to nondeterministic behavior of a protocol are studied in section 3. We enhance the verification methodology to extended FSM representations of test cases and protocols in section 4. In section 5, we present an example of verifying a test case using its FSM and EFSM models. Two other test verification approaches are summarized in section 6. Finally, conclusions are stated in section 7.

2. TEST VERIFICATION USING FSM MODELS

In this section, we introduce simple models of protocols and test cases and present a test architecture. Then we define some terms, present a reachability analysis algorithm, and analyze some theoretical properties of the algorithm. Finally, we apply the algorithm to a test case to generate a reachability graph and analyze the graph for various design errors in the test case.

We model a protocol as a nondeterministic FSM asynchronously interacting with the test system. Because of the asynchronous nature of communication, an interaction point between the protocol FSM and the test system is modeled as two unidirectional channels.

A test case must have alternative behavior and timeout mechanisms to test nondeterministic protocols, have loops to repeatedly apply test events if the implementation does not respond in expected time, have an OTHERWISE as an alternative receive event to capture all unintended events, and have a test verdict depending on each alternative outcome of the test purpose. Thus, test cases can be modeled as nondeterministic FSMs.

An abstract testing mechanism must be able to control the IUT and observe its responses at points known as the points of control and observations (PCOs). From a practical point of view, depending on the availability of the PCOs, the test mechanism can be either local or external [1]. The definition of an abstract test architecture requires that the PCOs be distributed over two abstract testing functions, the upper tester (UT) and the lower tester (LT). The LT provides control and observations at the appropriate PCO either below the IUT or remote from the IUT. The UT provides control and observation at the upper service boundary during test execution. In the LS architecture, behavior of a test case can be either separated into LT and UT behavior or specified in an in-line monolithic structure as shown in Figure 1*a*. Distributed single-layer (DS), coordinated single-layer (CS), and remote single-layer (RS) test systems are examples of external test architectures. We assume that the test cases are specified in LS architecture; if not, then they can be converted into this form.

2.1 Notations and Definitions

An external event in an FSM is characterized by three parameters: PCOs, *direction* ("?" denotes an input and "!" denotes an output), and the *message* in the event. However, no channel or direction is associated with an internal event. In a test case FSM the *start*(*timer*), *cancel*(*timer*), and *timeout*(*timer*) con-

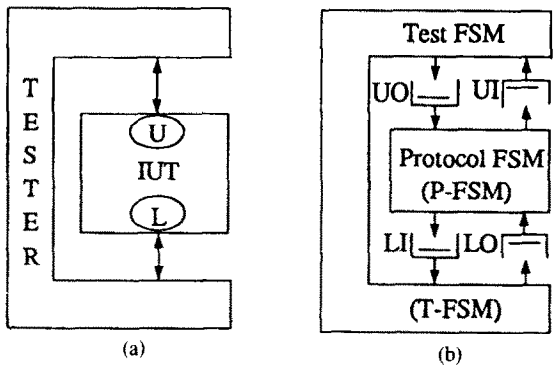

Figure 1. LS architecture-based test system (*a*); verification system for LS architecture (*b*).

stitute the internal events. In a protocol specification, nondeterminism is a way of modeling real-time properties of and the effect of the environment, particularly the service provider, on the communication system. Notationally, an unobservable transition or internal event (*i* event) is used to represent nondeterminism in a protocol model.

A transition in an FSM is characterized by four parameters: *from* and *to* states, *event*, and *priority*. To take care of the semantics of an OTHERWISE, a priority number is associated with each transition in a set of alternative transitions such that priority decreases from top to bottom. In a protocol FSM, all the transitions in a state are assigned the same priority.

We use function notations to access the parameters of events and transitions. For example, *dir*(E) returns the *direction* field in the event E, *pco*(E) returns the PCO of the event E, and *from*(r) returns the *from* state of transition r. The functions *int*(E) and *ext*(E) return a true value if E is an internal event and an external event, respectively. To extract the first message in a channel *channel_id*, we use the notation *head*(*channel_id*).

A communicating FSM consists of a set of states, a set of transitions, an initial state, a set of final states, a set of input channels, and a set of output channels. Verdicts in a test case FSM are stored as tags of the states. The function *verdict*(s) returns the verdict tag of the state s. The present and next states of an FSM M are denoted by *ps*(M) and *ns*(M), respectively. The OTHERWISE is abbreviated as OTH.

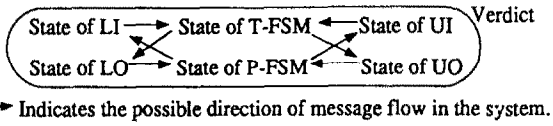

Figure 2. Structure of a global state in the LS architecture-based test verification system shown in Figure 1.

Definition 1: Test verification system. A test verification system (TVS) is defined to be a 3-tuple, $TVS = \langle T, P, C \rangle$, where T is the test case FSM, P is the protocol FSM, and C is the set of channels connecting T and P.

Example. The LS architecture and the corresponding test verification system are shown in Figure 1. A test case is modeled as a T-FSM, the protocol specification is modeled as a P-FSM, and each of the PCOs (L and U) is modeled by two unidirectional channels.

Definition 2: Global state. The global state g of a test verification system $TVS = \langle T, P, C \rangle$ is defined as

$$g \in G \subseteq \{states(T) \times states(P) \times states(LI) \times states(LO) \times states(UI) \times states(UO) \times Verdicts\}.$$

Example. The structure of the global states for the LS architecture-based test verification system in Figure 1 is shown in Figure 2.

The function *new-verdict*(*old verdict*, *present verdict*), abbreviated as *nv*(*ov*, *pv*) is used to compute the new verdict as follows:

```
function nv(ov, pv) {
if (ov = ="none" then return(pv)
else {
    if (ov = = Fail) or (pv = = Fail) then return(Fail)
    else if (ov = = Pass) and (pv = = Inconclusive) then return(Inconclusive)
    else if (ov = = Inconc.) and (pv = = Inconc.) then return(Inconclusive)
    else if (ov = = Pass) and (pv = = Pass) then return(Pass)
    }
end
```

Definition 3: Executable transitions. The set of executable transitions ET occurring in the present global state $g_p = \langle ps(T), ps(P), ps(LI), ps(LO), ps(UI), ps(UO), v \rangle$ in a test verification sys-

tem $TVS = \langle T, P, C \rangle$ is expressed as $ET(g_p) = ETP(g_p) \cup ETT(g_p)$, where

$$
\begin{aligned}
ETP(g_p) = \{r = \langle From, To, E, Priority \rangle | (&from(r) = = ps(P)) \wedge \\
&((int(E) \vee \\
&(ext(E) \wedge (dir(E) = = !)) \vee \\
&(ext(E) \wedge (dir(E) = = ?) \wedge (pco(E) = = L) \wedge (message(E) = = head(LO))) \vee \\
&(ext(E) \wedge (dir(E) = = ?) \wedge (pco(E) = = U) \wedge (message(E) = = head(UO))) \\
&) \\
&\} \textit{ and}
\end{aligned}
$$

$ETT(g_p)$ is computed as follows. Initially, $ETT(g_p) = \phi$, $R = \{r | (from(r) = = ps(T))\}$, and $init\_priority = 0$, $L\_flag = False$, $U\_flag = False$.

While $R \neq \phi$ begin {
init_priority = init_priority + 1
for $r \in R$(priority(r) = = init_priority) begin {
if ($int(E) \vee (ext(E) \wedge (dir(E) = = !))$) then $\{ETT(g_p) = ETT(g_p) \cup \{r\}, R = R - r\}$
if ($ext(E) \wedge (pco(E) = = L) \wedge (dir(E) = = ?) \wedge (message(E) = = head(LI)) \wedge (L\_flag = = False)$) then $\{ETT(g_p) = ETT(g_p) \cup \{r\}, R = R - r, L\_flag = True\}$
if ($(message(E) = = OTH) \wedge (pco(E) = = L) \wedge (content(LI) \neq \phi) \wedge (L\_flag = False)$) then $\{ETT(g_p) = ETT(g_p) \cup \{r\}, R = R - r, L\_flag =$ True}
if ($ext(E) \wedge (pco(E) = = U) \wedge (dir(E) = = ?) \wedge (message(E) = = head(UI)) \wedge (U\_flag = = False)$) then $\{ETT(g_p) = ETT(g_p) \cup \{r\}, R = R - r, U\_flag = True\}$
if
$((message(E) = = OTH) \wedge (pco(E) = = U) \wedge (content(UI) \neq \phi) \wedge U\_flag = False)$) then $\{ETT(g_p) = ETT(g_p) \cup \{r\}, R = R - r, U\_flag = True\}$
}
end
}

Definition 4: Perturbation. Let $g_p = \langle ps(T), ps(P), ps(LI), ps(LO), ps(UI), ps(UO), v \rangle$ be the present global state of a test verification system $TVS = \langle T, P, C \rangle$. Then, the perturbation of g_p by an executable transition $r = \langle From, To, E, Priority \rangle \in ET(g_p)$ is written as $perturbation(g_p, r) = g_n$ such that if r is a transition in T and $verdict(To) = pv$, then

$$
\begin{aligned}
g_n = &\langle To, ps(P), ps(LI), ps(LO), ps(UI), ps(UO), nv(v, pv) \rangle \textit{ if } int(E) \textit{ OR} \\
&\langle To, ps(P), ns(LI), ps(LO), ps(UI), ps(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = ?) \wedge \\
&(pco(E) = = L) \wedge ((message(E) = head(LI)) \vee (message(E) = = OTH) \wedge (LI \neq \phi)) \textit{ OR} \\
&\langle To, ps(P), ps(LI), ns(LO), ps(UI), ps(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = !) \wedge (pco(E) = = L) \textit{ OR} \\
&\langle To, ps(P), ps(LI), ps(LO), ns(UI), ps(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = ?) \wedge \\
&(pco(E) = = U) \wedge (message(E) = head(UI) \vee (message(E) = = OTH) \wedge (UI \neq \phi)) \textit{ OR} \\
&\langle To, ps(P), ps(LI), ps(LO), ps(UI), ns(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = !) \wedge (pco(E) = = U).
\end{aligned}
$$

Else if r is a transition in P, then

$$
\begin{aligned}
g_n = &\langle ps(T), To, ps(LI), ps(LO), ps(UI), ps(UO), nv(v, pv) \rangle \textit{ if } int(E) \textit{ OR} \\
&\langle ps(T), To, ns(LI), ps(LO), ps(UI), ps(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = !) \wedge \\
&(pco(E) = = L); \textit{ OR} \\
&\langle ps(T), To, ps(LI), ns(LO), ps(UI), ps(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = ?) \wedge \\
&(pco(E) = = L) \wedge (message(E) = head(LO)); \textit{ OR} \\
&\langle ps(T), To, ps(LI), ps(LO), ns(UI), ps(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = !) \wedge \\
&(pco(E) = = U) \textit{ OR} \\
&\langle ps(T), To, ps(LI), ps(LO), ps(UI), ns(UO), nv(v, pv) \rangle \textit{ if } (dir(E) = = ?) \wedge \\
&(pco(E) = = U) \wedge (message(E) = = head(UO)).
\end{aligned}
$$

In the following definitions, $C = \{LI, LO, UI, UO\}$, S_{Tf} is the set of final states in the T-FSM, S_{Pf} is the set of final states in the P-FSM, G_f is the set of final states of the global state space G.

2.2 Reachability Analysis Algorithm

We modify the reachability analysis algorithm for protocol validation [3], so that the modified algorithm can handle the following features of the test verification model: (*1*) internal events, priorities in a set of alternative events, OTHERWISE events, and verdicts in T-FSM; and (*2*) internal events in P-FSM. It is assumed that each channel is bounded. Perturbation of a global state is stopped if a channel overflow error, defined below, is detected.

Definition 5: Channel overflow error. An overflow error in a global state $g = \langle ps(T), ps(P), ps(LI), ps(LO), ps(UI), ps(UO), v \rangle$ is defined as $channeloverflow(g) = ((content(LI) > capacity(LI)) \vee (content(LO) > capacity(LO)) \vee (content\ (UI) > capacity(UI)) \vee (content(UO) > capacity(UO)))$, where $capacity(Q)$ is a parameter representing the maximum number of messages in the FIFO queue Q.

Algorithm 1.

Input: a T-FSM, a P-FSM, and the communication channels and their capacities.

Output: a global state space.

Procedure: the algorithm consists of the following steps:

S1. Define a set of global states G and a set of global transitions R. Initially, G contains only the initial global state g_1 and $R = \phi$.

S2. Find a member $g_p \in G$ of the set of global states whose perturbations have not been determined. If no such member exists, then terminate.

S3. Calculate the set of executable transitions $ET(g_p)$ in state g_p using Definition 3.

S4. Compute G_p, a set of global states, by perturbing g_p. Initially $G_p = \phi$.

$\forall r \in ET(g_p)\{G_p = G_p \cup perturbation(g_p, r) \wedge$
$R = R \cup \{\langle g_p, perturbation(g_p, r),$
$event(r), priority(r)\rangle\}$
}.

S5. If G_p is an empty set, report g_p as a terminal state in the global state space.

S6. $\forall g \in G_p$ *begin*{
channeloverflow$(g) \rightarrow$ *mark* g "*perturbed*" *and* $G \leftarrow G \cup \{g\}$
elseif $g \notin G \rightarrow$ *mark* g "*unperturbed*" *and* $G \leftarrow G \cup \{g\}$
}.

S7. Go to step S2.

In the following, we establish the termination and soundness properties and analyze the computational complexity of the algorithm.

Theorem 1. Algorithm 1 terminates.

Proof. The proof is done by showing that no infinite branch can be generated by the algorithm. Proof by contradiction is used here. Assume that the algorithm generates an infinite branch in the global state space. Referring to step S2, the algorithm does not perturb an already perturbed global state. Therefore, all the states in the infinite branch must be distinct. However, referring to the definition of a global state (Definition 2), since the FSMs are finite and the channels are bounded, no infinite number of distinct global states can be generated. Hence, the generation of an infinite branch is not possible. □

Theorem 2. All the global states generated by the algorithm are reachable from the initial global state.

Proof. The proof is done by induction on the generated global states as follows.

Basis: the initial system state g_1 consists of all the initial states of the constituent FSMs and empty channels. Hence, the initial global state is reachable.

Hypothesis: assume that a global state g_p is reachable from the initial state g_1.

Induction: let G_p be the set of global states generated by perturbing G_p. Step S4 of Algorithm 1 generates a state $s \in G_p$ by perturbing g_p, where perturbation of g_p is defined as the firing of a single executable transition in one FSM in the verification system. Hence all members of G_p are reachable from g_p. □

The size of the state space is proportional to the product of the state spaces of the T-FSM and the P-FSM, the number of nondeterministic loops in the FSMs, and the channel capacities. Explosion of the global state space is limited by the fact that a T-FSM is much smaller in size than a P-FSM, as one test case does not test all the protocol functions.

2.3 Analysis of Global State Space and Test Case Verification

We define the types of errors expected to be present in the global state space, in addition to channel

overflow errors defined in subsection 2.2, and discuss the issues involved in test verdict analysis.

2.3.1 Types of errors in the global state space.

Definition 6. Unspecified reception error. An unspecified reception error (URE) in a global state g is defined as $URE(g) = (g$ is not a terminating state$) \wedge (\exists c \in C | (\neg empty(c)) \wedge (c \notin C_l))$, where $C_l = \{channel(r) | dir(r) = = ?) \wedge ((r$ is a transition in state $g) \vee (r$ is reachable from g after a sequence of internal transitions))$\}$.

Definition 7: Deadlock error. A deadlock in a global state $g = \langle ps(T), ps(P), ps(LI), ps(LO), ps(UI), ps(UO), v \rangle$ is defined as $deadlock(g) = (g \in G_f) \wedge (ps(T) \notin S_{Tf}) \wedge (ps(P) \notin S_{Pf}) \wedge (\forall c \in C(empty(c)))$.

Definition 8: Blocking reception error. A blocking reception in a global state $g = \langle ps(T), ps(P), ps(LI), ps(LO), ps(UI), ps(UO), v \rangle$ is defined as follows: $blocking(g) = (g \in G_f) \wedge (\exists c \in C | \neg empty(c))$.

Definition 9: Tempo-blocking error (livelock). A livelock in a global state space G is defined as $livelock(G) = \exists$ a cycle in G.

2.3.2 Test verdict analysis. The final global states can be classified into two categories: one category of final states, called erroneous states, represents deadlock, blocking reception, and channel overflow errors; the other category of final states are error free.

Theorem 3. Assuming that the P-FSM does not contain any transition for exception handling, an error-free terminating global state must have a *Pass* or an *Inconclusive* verdict.

Proof. The proof is done by contradiction. Assume that an error-free terminating global state has a *Fail* verdict. An error-free terminating global state means both the T-FSM and the P-FSM are in their final states and the channels are empty. That is, the test behavior leading to the error-free terminating global state is allowed by the P-FSM behavior. Hence, the error-free terminating global state cannot end in a *Fail* verdict. Thus, it must end in a *Pass* or an *Inconclusive* verdict. □

Corollary 1. A *Pass/Inconclusive* verdict in a T-FSM is incorrect iff the verdict is attached to an erroneous state.

Corollary 2. A *Fail* verdict in a T-FSM is correct iff the verdict does not appear in any of the error-free states.

According to Theorem 3, it is possible to verify a *Fail* verdict. However, to verify a *Pass* or an *Inconclusive* verdict in a test case, the test purpose must be taken into account.

The purpose of a test case can be expressed as a regular expression on the interactions of the P- and T-FSMs. The global state space should accept the regular expression and terminate in an error-free state which has a *Pass* verdict. All other error-free states must have an *Inconclusive* verdict.

3. SYNCHRONIZATION ERRORS IN A TEST CASE

The issue of synchronization in a test case, an event timing problem, was first studied using a deterministic FSM model of a protocol specification [8]. With such a model of a protocol, a test case faces a synchronization problem if, in a sequence of two test events, the second event sends a message to the protocol through the PCO L (U), whereas the first event does not involve PCO L (U).

In this section, we present a general notion of synchronization error and enumerate the protocol specification behavior resulting in a synchronization error in a test case. Referring to Figure 3, let a test event (input or output) occur at PCO U (L) at time t_0, let t_1 be the time instant for a test event to be output at the same PCO U (L), and let S be a sequence of test events occurring in the interval

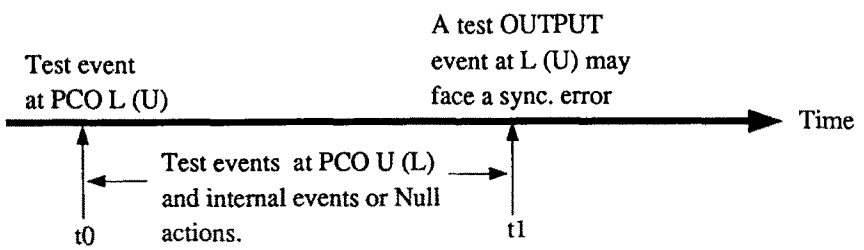

Figure 3. The context of an event facing a synchronization error.

$(t_0$-$t_1)$ such that S includes only internal test events and events occurring at the other PCO L (U). Therefore, conceptually, the output test event at PCO U (L) at time t_1 faces a synchronization problem if the length of the interval $(t_0$-$t_1)$ is indeterminate. In the following, we enumerate three cases of protocol behavior making the interval $(t_0$-$t_1)$ indeterminate.

1. The test case behavior in the interval $(t_0$-$t_1)$ corresponds to the protocol specification behavior $\{i, L!PDU\}$ $(\{i, U!PDU\})$.
2. The test case behavior in the interval $(t_0$-$t_1)$ corresponds to the protocol specification behavior $\{L?PDU1, L!PDU2\}$ $(\{U?PDU1, U!PDU2\})$ or $\{L?PDU1, Null\ action\}$ $(\{U?PDU1, Null\ action\})$.
3. The test case behavior in the interval $(t_0$-$t_1)$ corresponds to the protocol specification behavior $(i, Null\ action)$.

A null action means the protocol specification does not produce any event at the PCOs in response to the preceding input or internal event. Referring to Figure 5, the effect of receiving an acknowledgment PDU in state 10 is a null action taken by the protocol. The first two cases above are applicable to both the nondeterministic FSM and EFSM models of a protocol and the third case is applicable only to an EFSM model of a protocol where a null action means an action updating internal protocol variables and not generating any event at a PCO.

Note that the second case above is the source of a synchronization error in a test case involving a deterministic FSM model of a protocol studied in [8]. Therefore, the notion of a synchronization error studied here is a general one encompassing both deterministic and nondeterministic FSM and EFSM models of protocols.

In the following, a synchronization error is formally defined.

Definition 10: First output successor (fos). In the global state space G, there are two types of transitions: protocol specification transitions (p-transitions) and test case transitions (t-transitions). A t-transition $tr2 = \langle From2, To2, E2, Pr2\rangle$ is the first output successor of another t-transition $tr1 = \langle From1, To1, E1, Pr1\rangle$, written $(tr2 = fos(tr1))$ iff $((tr2$ is the first output t-transition following $tr1) \wedge (pco(E1) = = pco(E2)) \wedge (dir(E2) = = !))$.

Definition 11: Synchronization error. In a global state space G, a synchronization error is defined as $syncerr(G) = \exists\ tr1 = \langle From1, To1, E1, Pr1\rangle$ and $tr2 = \langle From2, To2, E2, Pr2\rangle | ((tr2 = = fos(tr1) \wedge$ (all the global states between the states $from(tr1)$ and $to(tr2)$ are error-free)) $\wedge$ (the sequence of transitions between $tr1$ and $tr2$ contains p-transitions with events $(i, E3)$ or $(E4, E3/null)$, or $(i, null)$, where $((dir(E3) = = !) \wedge ((pco(E3) \neq pco(E2)))$, $((dir(E4) = = ?) \wedge ((pco(E4) \neq pco(E2)))$.

A null transition appears only in the extended FSM model of a protocol and represents actions updating the variables of an extended FSM with no external protocol event taking place.

Though a synchronization error appears in a test case, it is not always possible to detect such an error by analyzing only the test case FSM, because the errors are caused by some inherent behavior of the protocol specification. Moreover, in some instances, while analyzing a test FSM, a sequence of events seems to represent a synchronization error, but a comparison of the behavior of the test case with the protocol FSM indicates that the event sequence causes an unspecified reception error and not a synchronization error. Therefore, for detecting synchronization errors and separating them from other types of errors, it is necessary to analyze the global state space of the test verification system.

4. TEST VERIFICATION USING EFSM MODELS OF TEST CASES AND PROTOCOLS

An EFSM model of a protocol is distinct from an FSM model in two respects: *(1)* an event exchanged between two EFSM entities is composed of a set of protocol parameters such as addressing, sequencing, flow control, and error detection information, in addition to the actual user data to be sent or received, and *(2)* in the EFSM model, individual component parameters in a receive event can be checked for the desired operation of the protocol.

A communicating EFSM is a eight-tuple, $EX = \langle S, V, R, s_0, S_f, h_0, I_C, O_C\rangle$, where S is a finite set of states, V is a finite set of identifiers—analogous to the variables and constants in a programming language—to hold values, s_0 is the initial state of EX, $S_f \subset S$ is a set of final states of EX, h_0 is a set of initialization functions, I_C is the set of input channels from which EX receives messages, and O_C is the set of output channels into which EX puts messages for communicating with other EFSMs.

An external event in an EFSM is a four-tuple, $E = \langle channel, direction, message\ identifier, list\ of\ parameters\rangle$. For example, $L?CR(Psrc\_ref, Pdst\_ref, Poption1, PRcredit)$ is an event, where L is the channel name, ? is the input direction, CR is the PDU name, and $(Psrc\_ref, Pdst\_ref, Poption1, PR$-

credit) is the list of parameter values received in the *CR* PDU.

A transition in an EFSM is represented by a six-tuple, $r = \langle From, To, Event, Predicate, Assignments, Priority \rangle$, where the transition takes the EFSM from the *From* state to the *To* state if *Predicate* ($Predicate : V \rightarrow \{True, False\}$) evaluates to true; then a set of value assignments is done in the *Assignment* clause, which is a set of functions ($f_n : V \rightarrow V$). *We denote a protocol EFSM and a test case EFSM as P-EFSM and T-EFSM, respectively.*

In the context of an EFSM, the definition of the set of executable transitions (Definition 3) in a global state is extended to account for the predicates in the EFSM's transitions.

Definition 12: Executable transitions. The set of executable transitions $XET(g_p)$ occurring in the present global state g_p is given by the following expression: $XET(g_p) = \{r | r \in ET(g_p) \wedge predicate(r) = True\}$, where $ET(g_p)$ is computed using Definition 3.

4.1 Modeling a Protocol as an EFSM

A P-EFSM can be obtained from the P-FSM model used in section 2. The FSM model is extended with the addition of a list of parameters for each ASP (Abstract Service Primitive) and PDU. A number of identifiers and constraint values are also defined. FSM transitions are modified in the following manner to obtain EFSM transitions. If the *Event* in a transition is a receive event then, first, a predicate denoting the enabling condition is added as a component of the EFSM transition. Next, the values of the EFSM's variables are updated by assigning values of the received parameters to the variables of the EFSM. These assignments are added to the *Assignment* component of the P-EFSM transition. If the *Event* is a send event, then the event parameters are assigned variable or constant values. An enabling condition can also be associated with an EFSM transition containing a send event.

4.2 Modeling a Test Case as an EFSM

In section 2, a test case designed to test a nondeterministic protocol was modeled as a nondeterministic FSM. As protocol events are parameterized and mechanisms to check values of those parameters are introduced in the form of transition predicates resulting in an EFSM description of a protocol, an event in a test case must also be parameterized with mechanisms to assign values to those parameters, and provisions must be there to put constraints on events. The structure of a complete event line in a test case is shown below.

L!CR Assignments Boolean_condition
Label Constraint Verdict

A constraint on a send event is interpreted as a set of assignments of values to the parameters of the event being sent; a constraint on a receive event is interpreted as a set of boolean conditions on the values of the parameters received in the event. The above send event line can be modeled as an EFSM transition: $\langle From, To, L!CR, boolean\_condition, (Assignment\ and\ Constraint), Priority \rangle$, where *From*, *To*, and *Priority* are dependent on the context of the event line in the test case. Applying this process to all event lines, we obtain the T-EFSM [9].

4.3 Reachability Analysis of EFSMs

Algorithm 1 was designed to generate the global state space of the combined behavior of a test case and a protocol modeled as FSMs. Using Definition 3, step S3 of the algorithm, computes the set of executable transitions $ET(g_p)$ in the present global state g_p and step S4 perturbs g_p by using $ET(g_p)$. The same algorithm can be used to generate the global state space of a test case and a protocol modeled as EFSMs by using Definition 12 of the set of executable transitions in the global state g_p.

The global state space generated from the EFSM models of a protocol and a test case differs from that generated from the FSM models in that it contains two additional fields of information—assignment of values to parameters and transition predicate. These additional fields are kept as parts of the transitions as explained in Figure 4.

5. VERIFICATION EXAMPLE

In this section, we demonstrate the application of the test case verification methodology. In section 5.1, we take the FSM models of a simplified transport protocol and a test case and verify the test case by

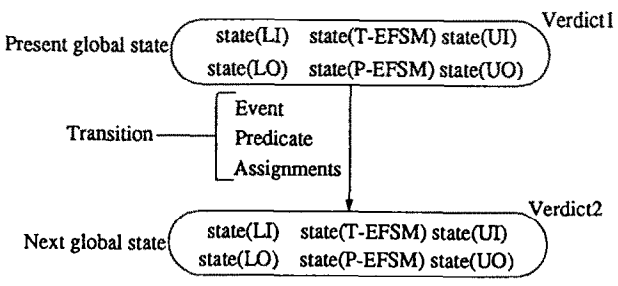

Figure 4. Global state and transition structures in the global state space of EFSM models.

using the methodology discussed in section 2. In section 5.2, we consider the EFSM models of the same protocol specification and test case and show how reachability analysis can be used to verify a test case described as an EFSM.

5.1 Reachability Analysis Using FSM Models

To show an application of the verification process, an FSM description of a simplified class 2 transport protocol, a test case, the global state space, and analysis of the global state space are presented below.

5.1.1 A simplified transport protocol. A nondeterministic FSM model of a class 2 transport protocol [10], called P-FSM, is shown in Figure 5; it can handle one transport connection. State 1 is both the initial and the final state of the FSM and represents a closed connection. State 10 corresponds to an open connection state. If the connection establishment procedure is initiated by the user of the transport entity, then the FSM moves from state 1 to 10 through the states {1, 2, 3, 4, 10} and if the connection is established by the peer entity, then the FSM moves from state 1 to 10 through {1, 5, 6, 7, 10}. There are two internal transitions in the FSM. One internal transition from state 10 to 11 models the effect of the environment on the protocol specification leading to a disconnection of the transport connection along the state sequence {10, 11, 12, 9, 1}. The other internal transition from state 10 to 18 models the acknowledgement (AK) transmission policies, including timeouts.

5.1.2 A test case for basic interconnection test. For verification purposes, we choose a test case for basic interconnection test from a real test suite developed at the National Computing Center (NCC) [11]. Since the NCC test suite is in the CS architecture, we rewrite the test case from CS to LS architecture using informations in the test management protocol used by the test suite. The architectural transformation of the test case is done manually and is shown in a TTCN-like notation in Figure 6. In the test case, a sequence of test events is represented one line after the other, each new event indented once from left to right, as time is assumed to progress. Test events at the same level of indentation and belonging to the same predecessor event represent the possible alternative events occurring at that time. Based on these notions of sequence and alternative events, the FSM description of the test case, called the T-FSM, is shown in Figure 7.

The test case contains many possible alternative test scenarios depending on how the implementation behaves. However, informally, the objective of the test case is to establish a connection, send a *DT* PDU, receive an *AK* for the sent DT, and close the connection. In terms of test events, the test purpose is represented by the regular expression {*L!CR. U?TCONind. U!TCONresp. L?CC. L!DT. U?TDATAind. L?AK. L!DR. U?TDISind. L?DC*}. This expression will be used while analyzing test verdicts.

5.1.3 Generation and analysis of global state space. In this example, all the channel capacities are assumed to be two. The initial global state is ⟨1, 1, *E*, *E*, *E*, *E*, *Null*⟩, where the first 1 is the initial state of T-FSM, the second 1 is the initial state of P-FSM, *E* represents the empty states of the channels, and *Null* represents that no verdict is associated with the initial global state. Applying Algorithm 1, we obtain the global state space shown in the Appendix. The initial global state is perturbed by the transition *L!CR* in the T-FSM generating global state 2. Then, global state 2 is perturbed by transition *Start*(*A*) in the T-FSM and *L?CR* in the P-FSM generating global states 3 and 4, respectively. This way, possible perturbations of the global states continue.

Three types of errors—unspecified reception error, blocking reception error, and channel overflow error—are found in the global state space shown in the Appendix. The occurrences of these errors together with their implications on the test case are discussed below.

Unspecified reception error. Global state 52 corresponds to T-FSM state 6, P-FSM state 12, *LI* con-

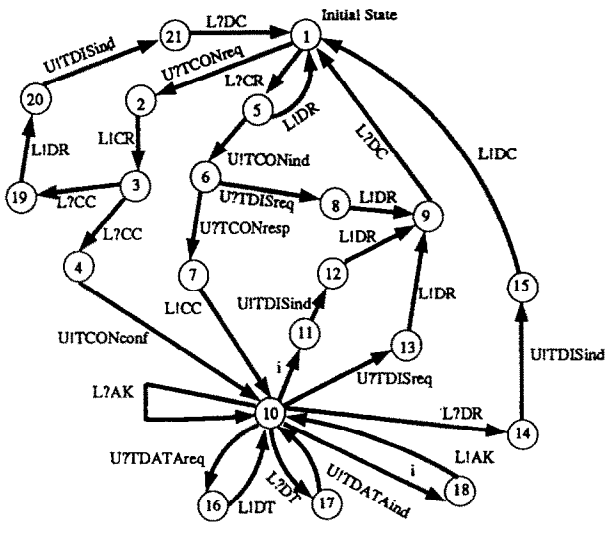

Figure 5. FSM description of a class 2 transport protocol.

```
L!CR
  Start(A)
    U?TCONind
    | U!TCONresp
    |   L?CC
    |   | Cancel(A)
    |   |  L!DT
    |   |    U?TDATAind
    |   |    | L?AK
    |   |    | | L!DR
    |   |    | |   U?TDISind
    |   |    | |   | L?DC              Pass
    |   |    | |   | L?OTHERWISE       Fail
    |   |    | |   U?OTHERWISE         Fail
    |   |    | L?OTHERWISE             Fail
    |   |    U?OTHERWISE               Fail
    |   ?Timeout                       Fail
    |   L?OTHERWISE                    Fail
    L?DR                               Inconclusive
    U?OTHERWISE                        Fail
```

Figure 6. A test case for basic interconnection test.

taining *CC*, *UI* containing *TDISind*, and other channels empty. That is, there are two messages—*CC* and *TDISind*—in the input queues of T-FSM, but T-FSM has no transition in state 6 to receive the *TDISind* from channel *UI*. Therefore, global state 52 indicates that there is an unspecified reception error in state 6 of the T-FSM. An analysis of the situation yields that the T-FSM faces an unspecified reception error because of the P-FSM's nondeterministic transition from state 10 to 11, i.e., nondeterministic initiation of disconnection by the protocol entity.

Blocking reception error. Global state 68 corresponds to T-FSM state 12, P-FSM state 9, *LI* containing *DR*, *UI* containing *TDISind*, and *UO* containing a *DT*. Since there are nonempty communication channels and no transitions are possible from state 68, it represents a blocking reception error. A

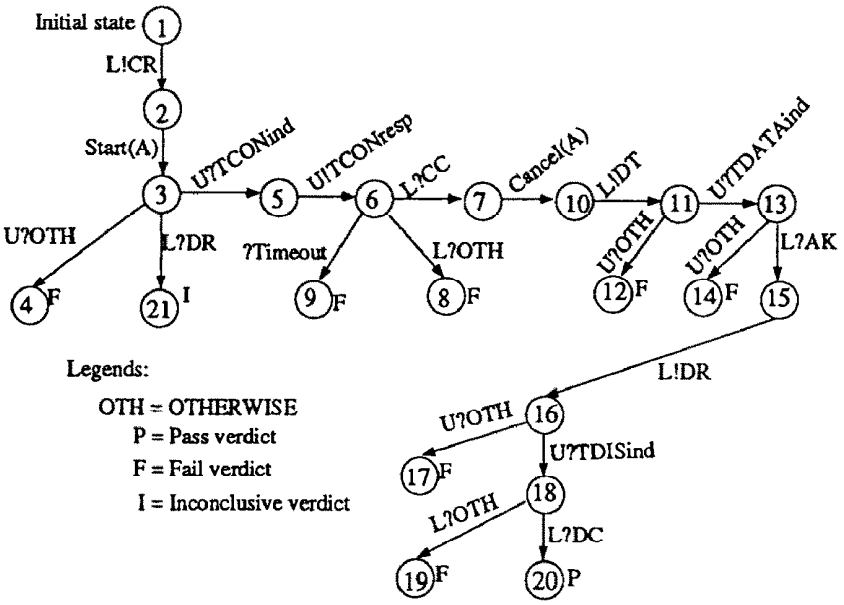

Figure 7. FSM description of test case in Figure 6.

blocking reception error in global state 68 is more complex than the unspecified reception error in state 52 in that to detect the cause of the error we have to go backward from state 68. The error in state 68 is due to two unspecified reception errors in states 20 and 61. That is, the errors in states 20 and 61 propagate forward in the state space and eventually represent a blocking reception error in state 68. Other global states representing blocking reception errors are 36, 50, 57, 87, and 100.

Channel overflow error. Any channel with more than two messages represents a channel overflow error. Global states 76, 77, 81, 102, and 105 represent channel overflow error. In all these instances, the error is in channel *LI* and is caused by the nondeterministic generation of *AK* PDUs in the P-FSM. Because of a channel overflow error, a test FSM can not be judged to be erroneous, because a communication channel neither reflects the behavior of a test case nor is controlled by a test case.

All the errors appearing due to nondeterministic generation of events by the protocol entity can be eliminated by using the notion of a background default tree [12] facility in the test case to capture nondeterministically generated PDUs that can neither be forced nor ignored.

Test purpose and verdict analysis. In the global state space, there are two error-free states, 42 and 108. Both the test behavior {*L!CR*, *Start*(*A*), *U?TCONind*, *U!TCONresp*, *L?CC*, *Cancel*(*A*), *L!DT*, *U?TDATAind*, *L?AK*, *L!DR*, *U?TDISind*, *L?DC*} and {*L!CR*, *Start*(*A*), *L?DR*} leading to global states 42 and 108, respectively, are accepted by the protocol specification. However, state 42 ends in a *Pass* verdict and state 108 ends in an *Inconclusive* verdict. To verify if the verdicts are correct, the test purpose regular expression is taken into account. Since the test behavior leading to global state 42 satisfies the test purpose regular expression, the assignment of a *Pass* verdict is correct. Since the test behavior leading to global state 108 does not satisfy the test purpose regular expression, assignment of an *Inconclusive* verdict is also correct.

5.2 Reachability Analysis Using EFSM Models

5.2.1 EFSM model of the protocol. We extend the P-FSM given in Figure 5 in three steps. In step 1, ASPs used to interact with a transport layer and the PDUs used in the transport layer are specified with parameters, as shown below.

TCONreq(*dst_addr*,*proposed_options*), *TCONind*(*src_addr*, *proposed_options*),

TCONresp(*dst_addr*, *proposed_options*), *TCONconf*(*src_addr*, *Accepted_options*),

TDISreq(*dst_ref*), *TDISind*(*src_addr*, *reason*), *TDATAreq*(*dst_ref*, *user_data*, *EOT*),

TDATAind(*src_ref*, *user_data*, *EOT*),

CR(*src_ref*, *dst_ref*, *credit*, *options*), *CC*(*src_ref*, *dst_ref*, *credit*, *options*),

AK(*src_ref*, *dst_ref*, *credit*, *exp_seqno*), *DC*(*src_ref*, *dst_ref*),

DT(*src_ref*, *dst_ref*, *tpdu_number*, *EOT*, *user_data*), *DR*(*src_ref*, *dst_ref*, *reason*).

In step 2, the following identifiers are defined to save variable and constant values.

Poptions: The set of options implemented in the protocol,

*Poption*1: options received in a *TCONreq*,

*Poption*2: options received in a *CR*,

*Poption*3: options received in a *CC*,

*Poption*4: options received in a *TCONresp*,

PTRseq: Expected sequence number in a received *DT*,

PTSseq: Sequence number in a *DT* to be sent,

PRcredit: Credit received from the peer,

PScredit: Credit to be sent to the peer.

Finally, in step 3 the transitions are extended by adding predicates and assignments. The following are some of the transitions in the P-EFSM.

Initialization: Psrc_ref = "spec," Pdst_ref = "tester", PRcredit = 1.

⟨1, 5, *L?CR*(*src_ref*, *dst_ref*, *Poption*2, *PScredit*), [*Pdst_ref* = = *src_ref* & *Psrc_ref* = = *dst_ref*], □, 1⟩

⟨5, 6, *U!TCONind*(*Pdst_ref*, *Poption*2), *T*, □, 1⟩

⟨6, 7, *U*?*TCONresp*(*dst_ref*, *Poption*4), [*dst_ref* = = *Pdst_ref*], [*PTRseq* = *PTSseq* = 0], 1⟩

⟨7, 10, *L*!*CC*(*Psrc_ref*, *Pdst_ref*, *Poption*4, *PRcredit*), *T*, □, 1⟩

⟨10, 11, *i*, *T*, [*Reason* = *yy*], 1⟩

⟨11, 12, *U*!*TDISind*(*Reason*), *T*, □, 1⟩

⟨12, 9, *L*!*DR*(*Psrc_ref*, *Pdst_ref*, *Reason*), *T*, □, 1⟩

⟨13, 9, *L*!*DR*(*Psrc_ref*, *Pdst_ref*, *Reason*1), *T*, □, 1⟩

⟨9, 1, *L*?*DC*(*src_ref*, *dst_ref*), [*src_ref* = = *Pdst_ref* & *dst_ref* = = *Psrc_ref*], *T*, □, 1⟩

⟨10, 14, *L*?*DR*(*src_ref*, *dst_ref*, *Reason*2), *T*, □, 1⟩

⟨14, 15, *U*!*TDISind*(*Reason*2), *T*, □, 1⟩

⟨15, 1, *L*!*DC*(*Psrc_ref*, *Pdst_ref*), *T*, □, 1⟩

⟨10, 18, *i*, *T*, □, 1⟩

⟨18, 10, *L*!*AK*(*Psrc_ref*, *Pdst_ref*, *PTRseq*, *PRcredit*), *T*, □, 1⟩

⟨10, 17, *L*?*DT*(*src_ref*, *dst_ref*, *Seq*, *EOT*, *Data*), [*src_ref* = = *Pdst_ref* & *dst_ref* = = *Psrc_ref*
AND PRcredit⟨ ⟩0 & *Seq* = = *PTRseq*], 1⟩

⟨17, 10, *U*!*TDATAind*(*Data*, *EOT*), *T*, [*PTRseq* = (*PTRseq* + 1) *mod* 128, *PRcredit* = *PRcredit* − 1], 1⟩

5.2.2 EFSM model of the test case. All the transitions of the T-EFSM corresponding to the test case in Figure 7 are given below. To check if an analysis of the global state space generated from EFSM models of a protocol and a test case is useful in detecting parameter errors in test cases, we have intentionally introduced an error in the test case EFSM: according to the P-EFSM description, the first *DT* received by it must have a sequence number equal to 0, but in the T-EFSM, the first *DT* sent has a sequence number 1.

Initialization: *Tsrc_ref* = "*tester*", *Tdst_ref* = "*spec*", *TRcredit* = 0

⟨1, 2, *L*!*CR*(*Tsrc_ref*, *Tdst_ref*, *Toption*1, *TRcredit*), *T*, □, 1⟩

⟨2, 3, *Start*(*A*), *T*, □, 1⟩

⟨3, 4, *U*?OTHERWISE, *T*, □, 2⟩

⟨3, 5, *U*?*TCONind*(*src_ref*, *Toption*2), [*src_ref* = = *Tsrc_ref*], □, 1⟩

⟨5, 6, *U*!*TCONresp*(*dst_ref*, *Toption*2), *T*, □, 1⟩

⟨6, 9, ?*Timeout*, *T*, □, 2⟩

⟨6, 8, *L*?OTHERWISE, *T*, □, 3⟩

⟨6, 7, *L*?*CC*(*src_ref*, *dst_ref*, *Toption*3, *TScredit*), [*src_ref* = = *Tdst_ref* & *dst_ref* = = *Tsrc_ref*], [*TTRseq* = 1], 1⟩

⟨7, 10, *Cancel*(*A*), *T*, □, 1⟩

⟨10, 11, *L*!*DT*(*Tsrc_ref*, *Tdst_ref*, *TTRseq*, *T*, "*abcd*"), *T*, □, 1⟩

⟨11, 12, *U*?OTHERWISE, *T*, □, 2⟩

⟨11, 13, *U*?*TDATAind*(*src_ref*, *Data*, *EOT*), [*Data* = = "*abcd*" & *EOT* = = *T*], □, 1⟩

⟨13, 14, *U*?OTHERWISE, *T*, □, 2⟩

⟨13, 15, *L*?*AK*(*src_ref*, *dst_ref*, *Seqno*, *Credit*), [*Seqno* = = *TTR_seq* + 1], □, 1⟩

⟨15, 16, *L*!*DR*(*Tsrc_ref*, *Tdst_ref*, *some_reason*), *T*, □, 1⟩

⟨16, 17, *U*?OTHERWISE, *T*, □, 1⟩

⟨16, 18, *U*?*TDISind*(*src_ref*, *some_reason*), *T*, □, 1⟩

⟨18, 19, *L*?OTHERWISE, *T*, □, 2⟩

⟨18, 20, *L*?*DC*(*src_ref*, *dst_ref*), [*src_ref* = = *Tdst_ref AND dst_ref* = = *Tsrc_ref*], □, 1⟩

5.2.3 Generation and analysis of global state space. Using Definition 12 of executable transitions in a global state, Algorithm 1 was run on the P-EFSM and the T-EFSM, described above, to generate a global state space. Because of space limitations, we have shown only a part of the global state space in Figure 8. Values of the variables updated by a transition are shown near the transition. Also, the predicate associated with a transition is shown near the transition as a label.

After successfully opening a connection, the T-EFSM sends a *DT* PDU with the sequence number 1 to the P-FSM by putting the message in the channel *LO*. The predicate associated with the *L?DT* transition in state 10 of the P-EFSM is [(*src_ref* = = *Pdst_ref*) AND (*dst_ref* = = *Psrc_ref*) AND (*PRcredit*⟨ ⟩0) AND (*Seq* = = *PTRseq*)], where *PRcredit* is the credit value sent to the T-EFSM, *Seq* is the sequence number in the received *DT*, and *PTRseq* is the expected sequence number. Since *PRcredit* = 1, the condition (*PRcredit*⟨ ⟩0) evaluates to true. However, *Seq* = 1 and *PTRseq* = 0. Therefore, (*Seq* = = *PTRseq*) evaluates to false and, consequently, the entire predicate evaluates to false. Thus, the transition *L?DT* is rendered unexecutable in state 10 of the P-EFSM and the global state 12

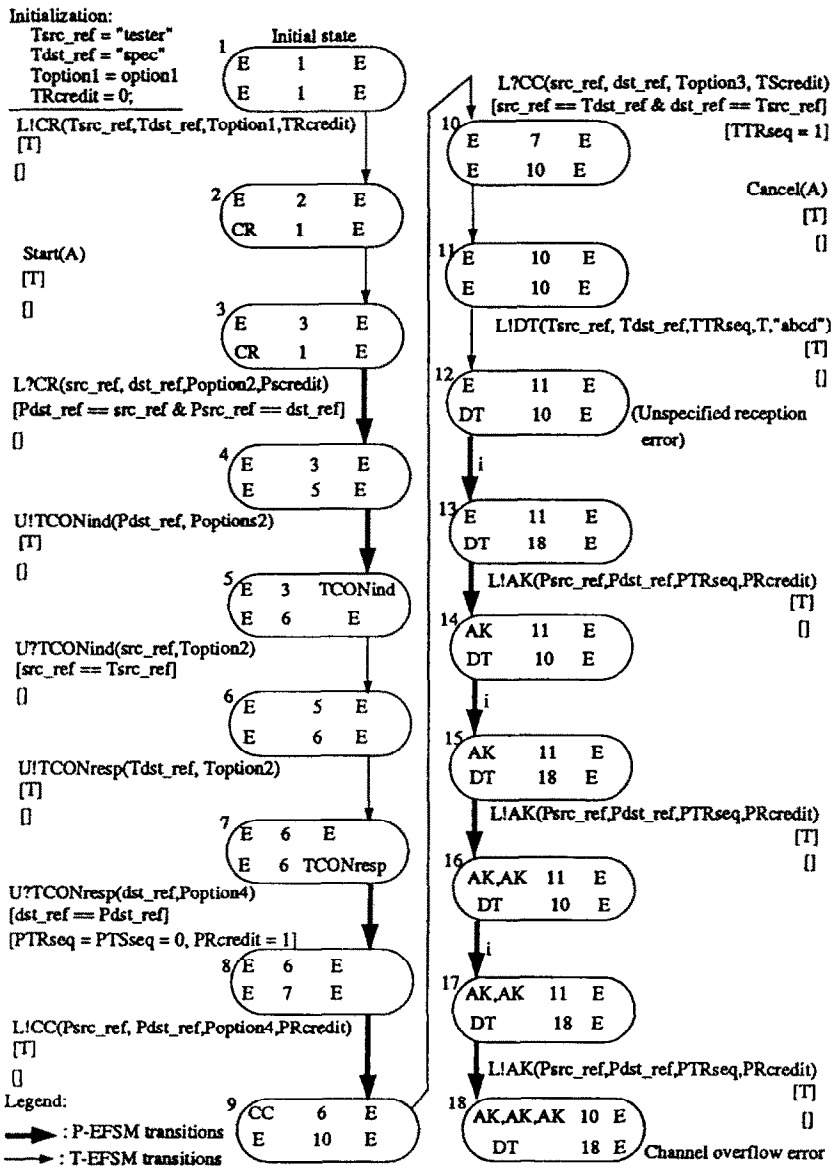

Figure 8. Partial global state space generated from the EFSM descriptions of the test case and protocol specification.

cannot be perturbed using the same transition. Then, the P-EFSM nondeterministically outputs a sequence of *AK* PDUs in state 10 to the channel *LO*, which eventually overflows as seen in the global state 18.

Analysis of the global state space reveals that there is an unspecified reception error in global state 12. As expected, this reception error occurs because transition *L?DT* in state 10 of the P-EFSM cannot be fired because the predicate evaluates to false. This unspecified reception error is due to the parameter error introduced in the test case.

6. RELATED RESEARCH

Two other approaches to test case verification have been reported in which the protocol and the test case are assumed to be specified in the formal description technique LOTOS [13] and TTCN [1], respectively.

In the first approach [14], first both the LOTOS specification of a protocol and the TTCN specification of a test case are translated into a common semantic model, a chart [15] with FIFO queues modeling the communication mechanism between the test case and the protocol specification. Next, an interleaved symbolic execution mechanism is used to compare the behavior of the test chart with the protocol chart. A limitation of this approach is that only static aspects of a test case can be compared with the behavior of a protocol specification and dynamic aspects due to timeouts cannot be verified.

In the second approach [16], the verification process consists of two steps. First, the TTCN test case is translated into a LOTOS specification. Second, a test and trace analysis tool called TETRA, based on a LOTOS interpreter [17], takes the LOTOS descriptions of the test case and the protocol specification as inputs and computes their parallel composition by tracing the executable paths in the two specifications. The concerns expressed about the tool are its high space requirement and long verification time.

7. CONCLUDING REMARKS

We have developed a simple yet powerful methodology to verify a test case against a protocol specification by using nondeterministic FSM models. It was observed that even a simple test case for a basic interconnection test in the LS architecture contains a few design errors. These were classified as reception, blocking reception, and channel overflow errors. A general notion of a synchronization error in the context of deterministic and nondeterministic FSM and EFSM models of protocols was studied. To detect such an error and distinguish it from other types of errors, it is necessary to analyze the global state space. Finally, we modeled a test case and a protocol specification as EFSMs and used the same algorithm to generate the global state space. The only change required was to define an executable transition to take the transition predicates into account.

In protocol validation systems, state explosion can be controlled by using the notion of a canonical sequence when two processes communicate through two channels [18]. However, when two processes communicate through more than two channels, it has been stated that either a canonical sequence does not exist or is very difficult to find. Hence, the notion of canonical sequence does not seem to be useful in our verification model to control the state explosion. It will be interesting to explore other ways to control the state explosion problem.

We have modeled test purposes as regular expressions. It would be interesting to develop models for error types, possibly using temporal logic formulas. Such an approach could lead to a semantic model for test correctness requirements.

ACKNOWLEDGMENTS

This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Canadian Commonwealth Fellowship Agency.

REFERENCES

1. International Organization for Standardization (ISO), Information processing systems—Open systems interconnection. 9646 (1991), Conformance testing methodology and framework. Part 1: General concepts, ISO 9646, (1991).
2. G. J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall Software Series, New York, 1991.
3. C. H. West, General Techniques for Communications Protocol Validation, *IBM J. Res. Dev.* 22, 393–404 (1978).
4. P. Zafiropulo, C. H. West, H. Rudin, D. D. Cowan, D. Brand, Towards Analyzing and Synthesizing Protocols, *IEEE Trans. Commun.* COM-28, 651–661 (1980).
5. S. T. Vuong, D. D. Hui, and D. D. Cowan, VALIRA —A tool for protocol validation via reachability analysis, in *Proceedings of the IFIP WG6.1 6th International Workshop on Protocl Specification, Testing, and Verification, VI* B. Sarikaya and G. von Bochmann, eds.), North Holland, Amsterdam, 1987, pp. 35–41.
6. P. Zafiropulo, Protocol Validation by Duologue-Matrix Analysis, *IEEE Trans. Commun.* COM-26, 1187–1194 (1978).
7. K. Sabnani, An Algorithmic Technique for Protocol

Verification, *IEEE Trans. Commun.* COM-36, 924–931 (1988).
8. B. Sarikaya and G. von Bochmann, Synchronization and Specification Issues in Protocol Specification, *IEEE Trans. Commun.* COMM-32, 389–395 (1984).
9. K. Naik and B. Sarikaya, EFSM semantics for TTCN, in *Proceedings of the 15th Biennial Conference on Communication*, Queen's University, Kingston, Ontario, Canada, 1990.
10. G. von Bochmann, Specification of a Simplified Transport Protocol Using Different Formal Description Techniques, *Comp. Net. ISDN Syst.* 18, 335–377 (1989/90).
11. *Abstract Test Suite for Transport Protocol Class 2*, The National Computing Center Limited, Manchester, UK, 1988.
12. Proposed Draft Amendment 1 to ISO 9646—Part 3: TTCN Extensions, ISO/IEC 9646-3/PDAM 1, January 1992, 21p.
13. International Organization for Standardization (ISO), Information processing systems—Open systems interconnection—LOTOS—A formal description technique based on the temporal ordering of observational behavior, ISO 8807 (1988).
14. K. Naik and B. Sarikaya, Static validation of TTCN test cases, in *Proceedings of the 3rd International Workshop on Protocol Test Systems*, North-Holland, Amsterdam, 149–170, 1990.
15. G. Karjoth, Implementing process algebra specifications by state machines, in *Proceedings of the IFIP WG6.1 International Workshop on Protocol Specification, Testing and Verification, VIII*, North-Holland, Amsterdam, 47–60, 1988.
16. M. Dubuc, G. von Bochmann, O. Bellal, F. Saba, Translation from TTCN to LOTOS and the validation of test cases, in *Proceedings of the Formal Description Technique '90 Conference (FORTE '90)*, North-Holland, Amsterdam, 1990.
17. L. Logrippo, A. Obaid, J. P. Briand, M. C. Fehri, An Interpreter for LOTOS, A Specification Language for Distributed Systems, *Software Pract. Exp.* 18, 365–385 (1988).
18. J. Rubin and C. H. West, An Improved Protocol Validation Technique, *Comp. Net. ISDN Syst.* 6, 65–73 (1982).

APPENDIX

System State Space of the Validation for the T-FSM and P-FSM shown in Figures 7 and 5, respectively.

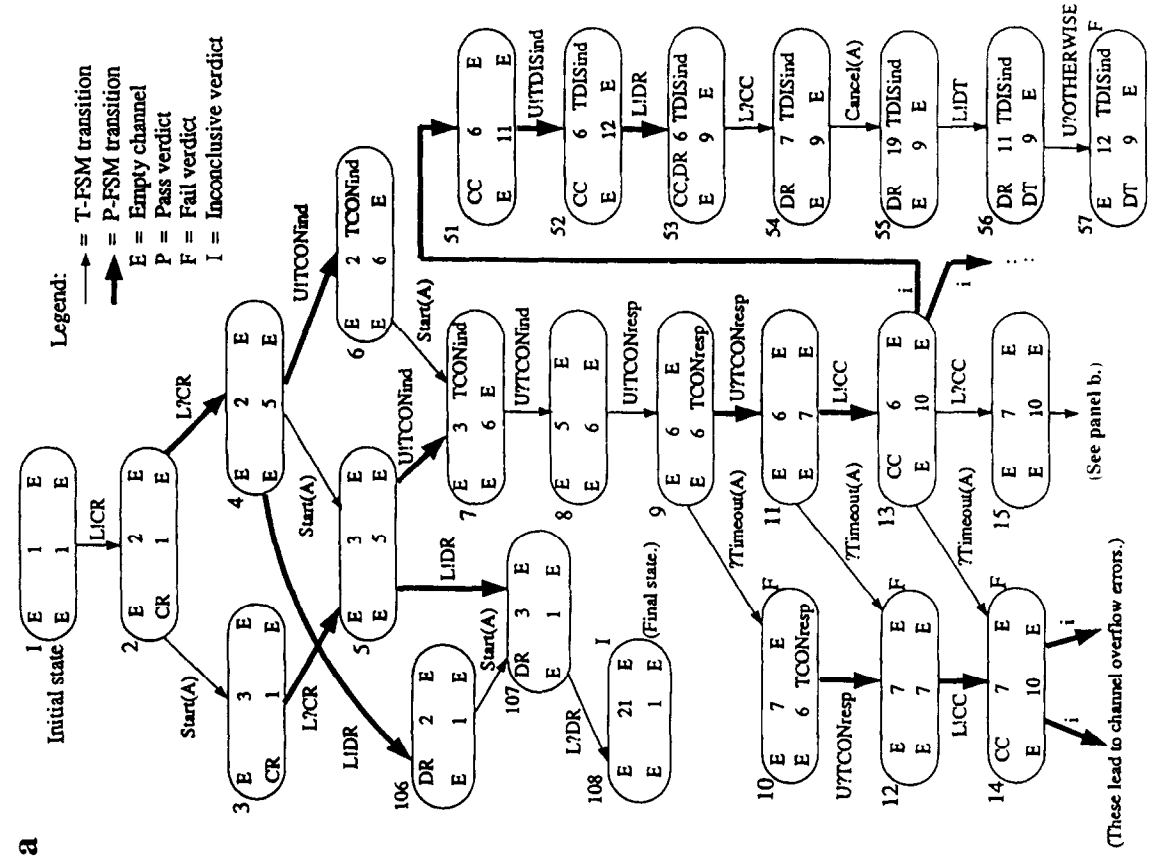

Figure a.1.

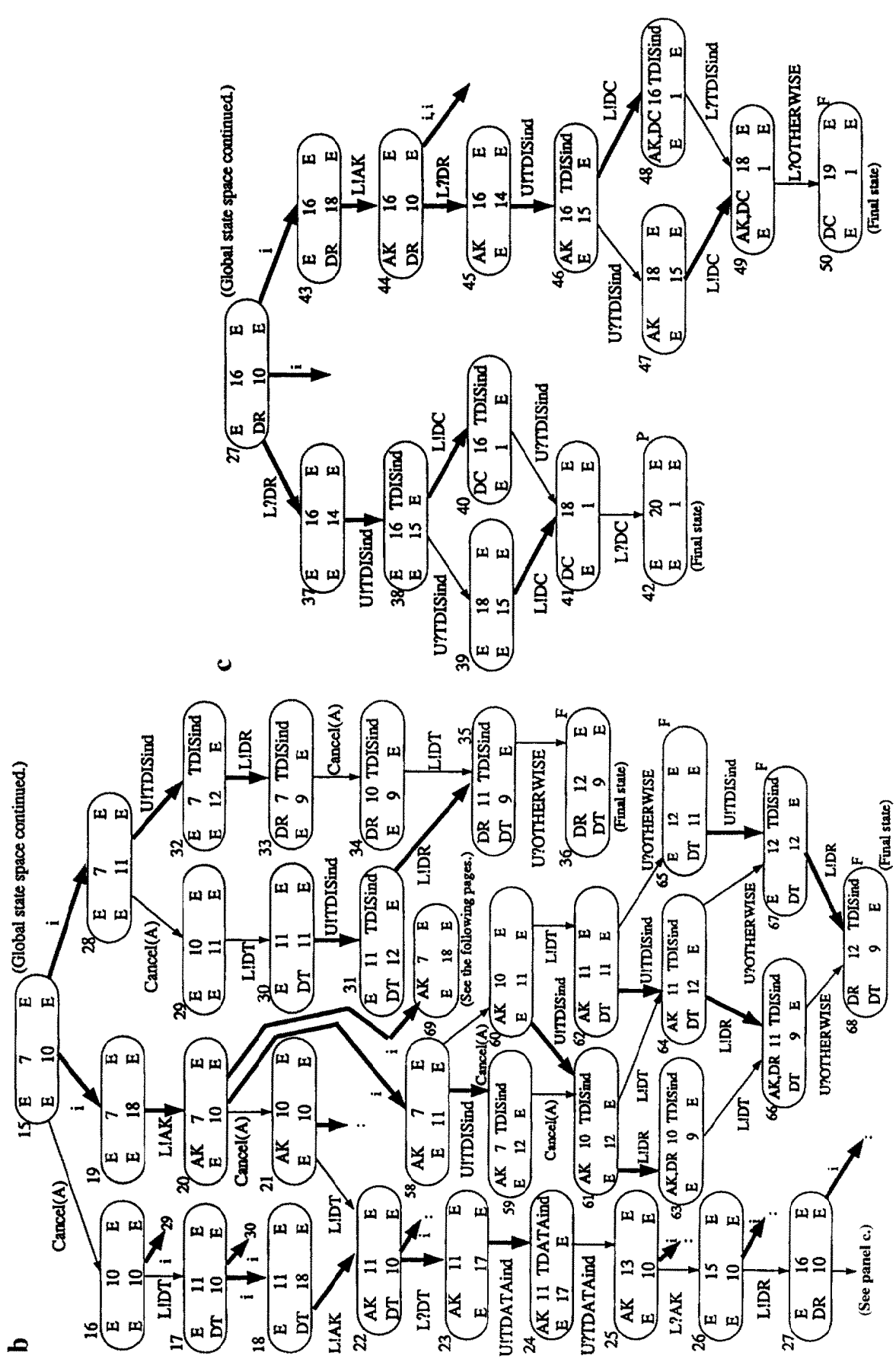

Figure a.2.

Figure a.3.

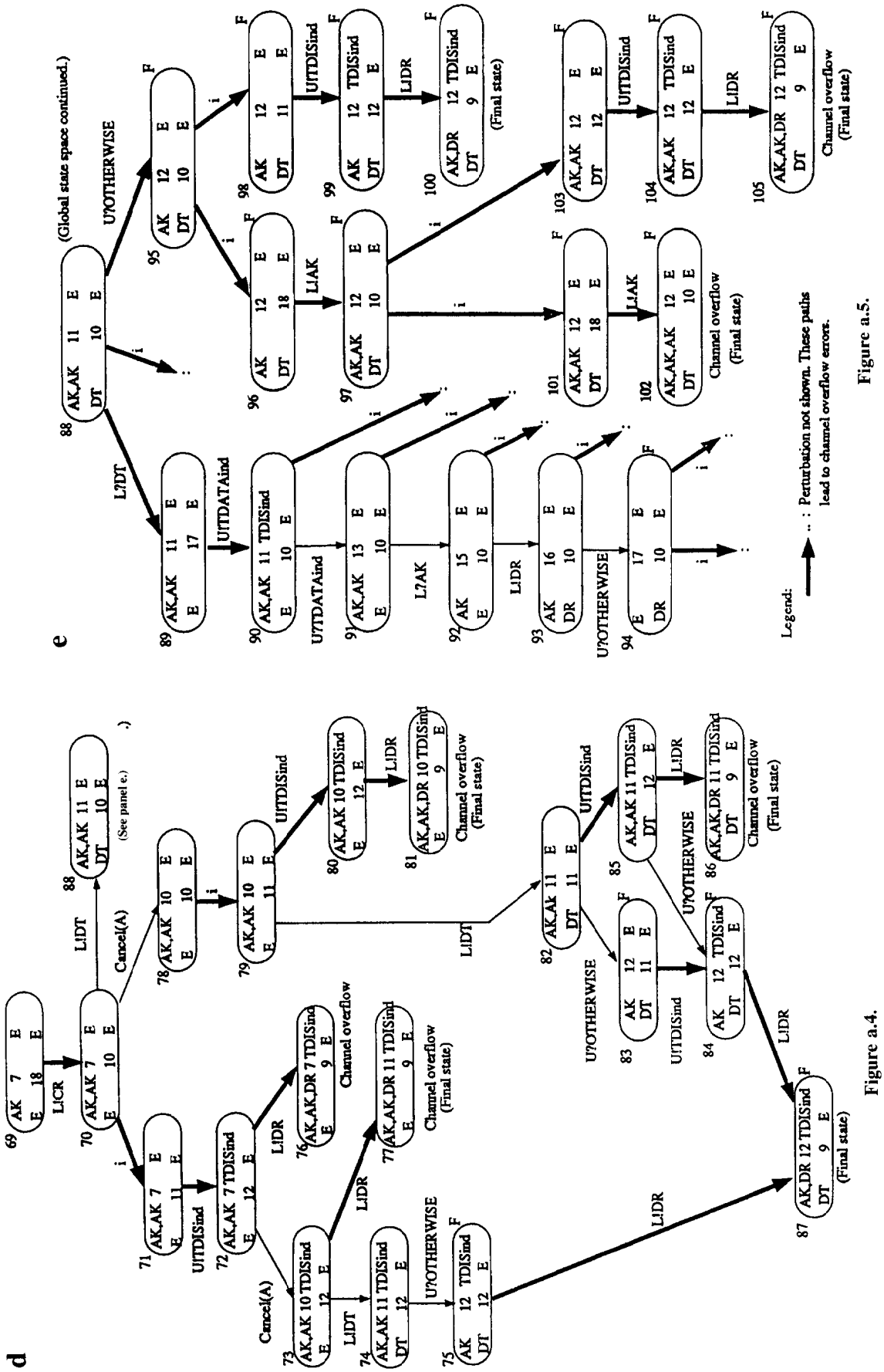

Figure a.4.

Figure a.5.