A Discrete Event Systems Approach to Network Fault Management: Detection & Diagnosis of Faults

S. Bhattacharyya, Z. Huang
Electrical & Computer Engineering
University of Kentucky
Lexington, KY 40506

V. Chandra
Department of Technology
Eastern Kentucky University
Richmond, KY 40475

R. Kumar
Electrical & Computer Engineering
Iowa State University
Ames, IA 50011

***Abstract*— An important aspect of network management is fault management, i.e., determining, locating, isolating, and correcting faults in the network. We study modeling of communication network protocols, and their fault detection, fault identification and fault location in the discrete event systems diagnosis framework. Our approach provides a systematic way of performing fault diagnosis in network fault management, and is illustrated through a network fault diagnosis example.**

I. Introduction

A fault is regarded as a physical condition that causes a device, a component or, an element to fail to perform in a required manner. A typical diagnosis system uses the observations of the system to detect the failure, isolate (locate) the source of failure, and diagnose the root cause. Discrete Event System (DES) approach to failure diagnosis [2], [3], [4], [5], [9], [10], [1] is a systematic method for timely and accurate diagnosis of failures in event-driven systems, such as telecommunication networks. The system behavior is modeled as a a finite state machine. The failure events form a part of the event set. An occurrence of fault can be detected/isolated/diagnosed by observing the event-trace the system executes. The challenge is to detect the occurrence of failures within bounded delay. We present a DES framework to represent network protocols, and address the problem of their fault management.

II. Notation and preliminaries

A Discrete Event System (DES) [6] is a dynamic system that evolves in accordance with the abrupt occurrence of events, at irregular intervals. Let Σ denotes finite set of events. A concatenation of events is called an event $trace$. A $language$ is a collection of traces. Σ^* is the set of all finite traces of events of Σ including the empty string ϵ. A language is thus a subset of $\Sigma^{*\cdot}$. For a language H, the notation $\overline{H}$ called the prefix closure of H, and it consists of the set of all prefixes of traces in H. H is said to be prefix closed if $H = \overline{H}$. A discrete event system can be represented by a state machine or automaton consisting of a four tuple, $G = (X, \Sigma, \delta, x_0)$, where, X is the set of states, Σ is the set of events, $\delta : X \times (\Sigma \bigcup \{\epsilon\}) \rightarrow 2^X$ is the state transition function, and $x_0 \in X$ is the initial state. G is said to be deterministic if $\delta(., \epsilon) = \emptyset$ and $|\delta(., \sigma)| \leq 1$ for all $\sigma \in \Sigma$. Otherwise it is nondeterministic. The event behavior of the discrete event system modeled by G is described by its *generated* language: $L(G) := \{s \in \Sigma^* \mid \delta(x_0, s) \neq \emptyset\}$. *Synchronous composition* [6] of state machines is used to represent the concurrent behavior of two interacting discrete event systems. The synchronous composition of G_1 and G_2 is denoted $G_1 \| G_2$.

This work was supported in part by the National Science Foundation under the grants NSF-ECS-0218207, NSF-ECS-0244732 and NSF-EPNES-0323379, and a DoD-EPSCoR grant from the Office of Naval Research under the grant N000140110621.

III. Network Fault Management Approach

A communication network is an event-driven system, in which events occur at each node, at unknown irregular intervals of time. Accordingly a communication network can be regarded as a DES, with each node in the network represented as a deterministic finite state machine (FSM) [7], [8]. The normal behavior of the system is modeled by state machine G_0, and the faulty behavior of the system is modeled by state machines $\{G_i | i > 0\}$, where i represents a system with i^{th} kind of fault. Each G_i contains at most one fault, reflecting the underlying assumption that at most one fault can exist at any given time. We can think of the overall model of the system to be G whose behavior is the union of the behaviors of $G_{i \geq 0}$.

The state behavior of the system is given by state-traces specifying the sequence of states visited, whereas event behavior is given by the event traces that can be executed. For a state trace Π, we use $tr(\Pi)$ to denote the associated event trace. Two state-traces Π_1, and Π_2 with the same event trace $tr(\Pi_1) = tr(\Pi_2)$ are considered indistinguishable. Henceforth, we represent the state behavior of G_i by L_i, and event behavior by $L(G_i)$ then $\bigcup_{i \geq 0} L_i$ is the set of all possible state traces, and $\bigcup_{i \geq 0} L(G_i)$ is the set of all possible event traces that can be executed by a system.

We consider the simplified version of the network layer of X.25 protocol taken from [8] to show the models of different systems $\{G_i | i \geq 0\}$ under the representation method we have discussed. X.25 is a protocol standard for WAN communications that defines how connections

between user devices and network devices are established and maintained.

Example 1: A model G_0 of the non-faulty behavior for a part of the X.25 protocol is shown in Figure 1. In G_0, there exist four states S_1, S_2, S_3, and a "dump state" D. The

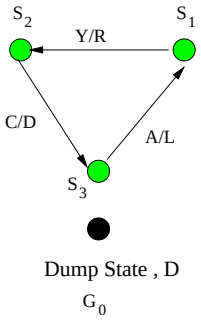

Fig. 1. Normal behavior of the system

system transitions from one state to the other on an observed input-output pair. The input-output labels used carry the following meaning: Y: Ready (initialized), R: Call Request, C: Call Accept, D: Data, A: Acknowledge, and L: Clear (terminate the call).

We consider input, transition, and output faults.

- An input fault occurs when a state receives an unexpected input. A transition to the dump state occurs due to an input fault, and then the system stops execution. For example, when S_1 receives C as input, an input fault occurs (Figure 2) and it transitions to the state D.
- A transition fault occurs when a state transitions to an unexpected state on an expected input-output pair. For example, S_1 transitions to S_3(Figure 4) or remains at S_1 (Figure 3) on input-output pair Y/R due to a transition fault.
- An output fault occurs when a state transitions to the expected state but gives an erroneous output. For example in Figure 5 we see that S_1 transitions to S_2 (the expected state) but gives a wrong output due to an output fault. When either a transition or an output fault occurs, the system continues functioning.

The various faults occurring in the system are modeled using FSMs in Figures 2-5. G_1,G_2,G_3 model the various input faults (Figure 2), G_4,G_5,G_6,G_7,G_8,G_9 model the various transition faults (Figure 3 and Figure 4), and G_{10},G_{11},G_{12} model the various output faults (Figure 5), respectively.

A. Overall System Model

In order to obtain a model for the overall system with normal and faulty behavior combined, one obvious approach is to introduce a new initial state S_{New}. From this state, one adds ϵ-transitions to all the initial states of all the systems G_i, $i \geq 0$. This gives the non-deterministic finite state machine of the system with and without faults. Using the

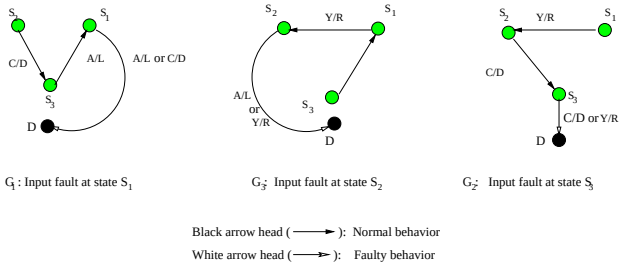

Fig. 2. Models showing input faults

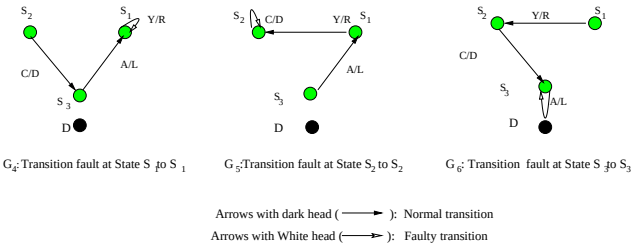

Fig. 3. Models showing transition faults

fact that at most one fault can exist at any given time we can give a more compact and intuitive model of the overall system as follows.

Algorithm 1: (To build model G of overall system)

1) Since initially there is no fault, initial state S_1 is augmented by the label N.
2) Next from the initial state (S_1, N), consider all the possible input-output pairs that can be observed. For our example, the input-output pairs can be one among Y/R or C/D or A/L. When the input-output pair is Y/R, the system transitions to either (S_2, N) indicating a normal behavior, or to (S_1, F_1) indicating the transition fault F_1, or to (S_3, F_2) indicating the transition fault F_2. If we observe Y/D or Y/L (unexpected output on expected input) we know an output fault has occurred, and the system transitions to (S_2, F_5). On A/L the system transitions to state (D, F_4) indicating input fault on A/L. Newly reached states are recursively considered for further state transitions in a similar fashion.
3) If a state has a label F_i, $i > 0$, then since only one fault can exist at a given time, the ensuing transitions are non-faulty. Considering (S_1, F_1), it executes Y/R and transitions to (S_2, F_1), where it executes C/D and transitions to (S_3, F_1), and finally executes A/L and transitions back to (S_1, F_1).

Figure 6 shows the NDFSM built using Algorithm 1.

B. Detectability

Detectability means the ability to detect the occurrence of a fault within bounded delay.

Definition 1: Given a set of normal state behaviors L_0, a set of all state behaviors L, the pair (L_0, L) is said to be detectable if: $(\exists n \in N)(\forall \Pi \in L - L_0)(\forall \Pi_1 = \Pi\Pi_2 \in$

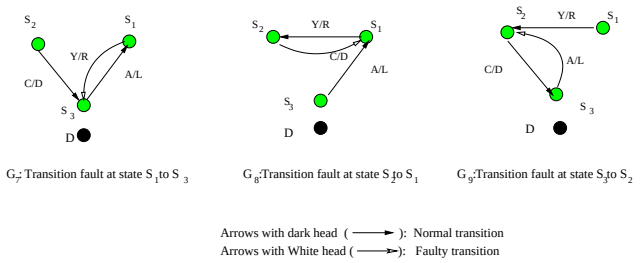

Fig. 4. More models showing transition faults

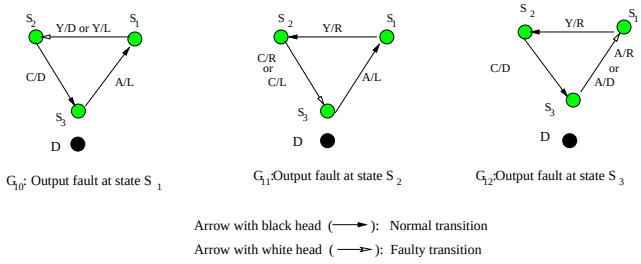

Fig. 5. Models showing output faults

$L, |\Pi_2| \geq n$ or Π_1 deadlocks) $\Rightarrow (\forall \Pi_3 \in L, tr(\Pi_1) = tr(\Pi_3))(\Pi_3 \in L - L_0)$

(If Π is a faulty trace in $L - L_0$, and Π_1 is either a sufficiently long extension, or a deadlocking trace, then every trace Π_3 in L that has the same event trace as Π_1, i.e., $tr(\Pi_3) = tr(\Pi_1)$, should be a faulty trace.) We apply the Jiang *et al* [3] algorithm to determine detectability. Since we are concerned about the detection of a fault and not it's fault type, we modify the non-deterministic finite state machine we obtained earlier, by replacing each F_i label with label F. The steps are:

Algorithm 2: (For detectability)

1) Build the NDFSM G_D for detectability by replacing each F_i label by F in G.
2) Construct $G_D \| G_D$.
3) Check for ambiguous cycles or ambiguous deadlock states. (A state in $G_D \| G_D$ is ambiguous if the two coordinates carry different fault labels.) Detectable iff no ambiguous cycles or ambiguous deadlock states.

Example 2: G_D for our example is show in Figure 7. From the synchronous composition of G_D with itself (Figure 8) we find that there are no ambiguous cycles or deadlocks. So, the system is detectable.

When a system is such that it is not detectable we can find a detectable sub-system as follows:

Algorithm 3: (To find detectable sub-system)

1) In $G_D \| G_D$, identify the traces leading to ambiguous cycles or ambiguous deadlocks. Each trace in $G_D \| G_D$ is a pair of indistinguishable trace of G_D.
2) Pick any of the indistinguishable traces, and remove it from G_D.

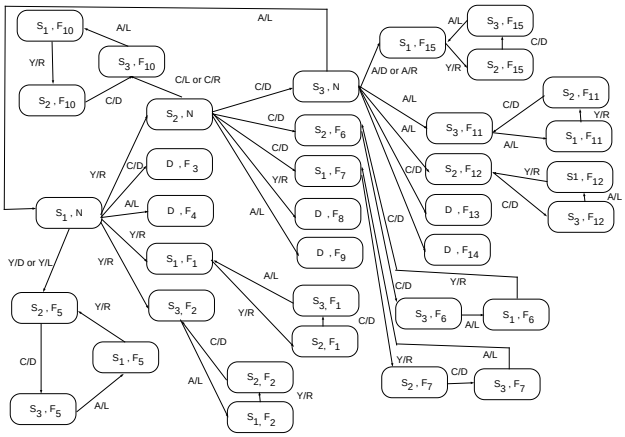

Fig. 6. NDFSM G of overall behavior

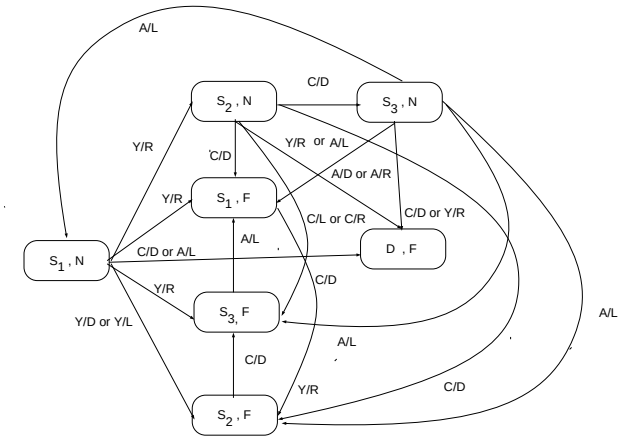

Fig. 7. NDFSM G_D for detectability

C. Diagnosability

Identification of a particular kind of fault is known as the diagnosis. A system is diagnosable for fault i if it is possible to identify within bounded delay occurrences of failures of type i using the observation of events.

Definition 2: A system with state behavior L, and fault i behavior L_i is said to diagnosable for fault i if: $(\exists n_i \in N)(\forall \Pi \in L)(\forall \Pi_1 = \Pi\Pi_2 \in L, |\Pi_2| \geq n_i$ or Π_1 deadlocks) $\Rightarrow (\forall \Pi_3 \in L, tr(\Pi_3) = tr(\Pi_1))(\Pi_3 \in L_i)$

(If Π is a trace in L with a failure of type F_i, and Π_1 is either a deadlocking or a sufficiently long extension of Π, then every trace Π_3 in L that has the same event trace as Π_1, i.e., $tr(\Pi_3) = tr(\Pi_1)$, should contain in it a type i failure.) To check for diagnosability of fault i we apply the algorithm by Jiang *et al* [3].

Algorithm 4: (For diagnosability)

1) Construct the non-deterministic finite state machine G_{Fi} by replacing all the fault labels F_j $(j \neq i)$ by N in G (when we are diagnosing for fault i, fault j can

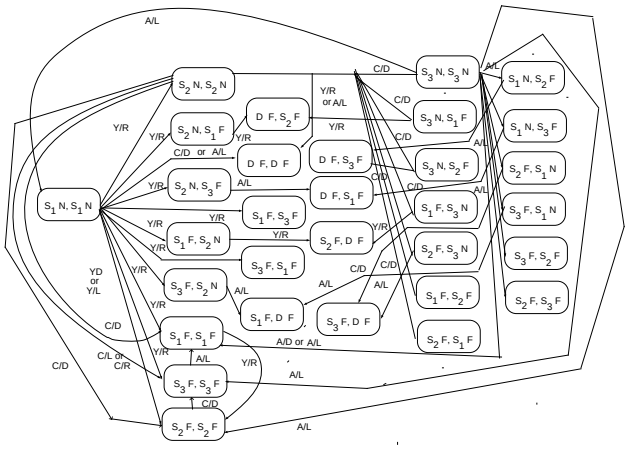

Fig. 8. $G_D \parallel G_D$

be treated as normal).

2) Construct $G_{F_i} \| G_{Fi}$.
3) Check for ambiguous cycles or ambiguous deadlock states. Diagnosable iff no ambiguous cycles or ambiguous deadlock states.

Example 3: For diagnosis of fault F_4, figure 9 shows the model of the state behavior with only fault F_4 indicated. In Figure 10 there are ambiguous deadlock states (DF_4, DN), and (DN, DF_4), showing fault F_4 is not-diagnosable. To

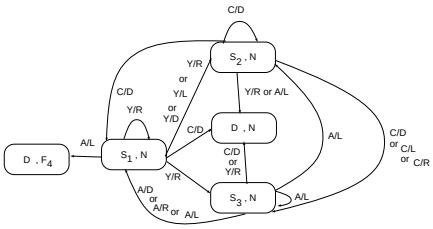

Fig. 9. NDFSM G_{F_4}

find the diagnosable part we remove all the transitions on A/L, excepting the one, which leads to the input fault F_4. The resulting NDFSM, G'_{F_4} is shown in Figure 11. Figure 12 considers $G'_{F_4} \| G'_{F_4}$ and shows that there are no ambiguous deadlock states and no ambiguous states in cycle (as expected). So, we can conclude that G'_{F_4} is fault F_4 diagnosable.

D. The Diagnoser

A diagnoser is a discrete event system which passively observes the input-output sequence executed by the system and determines the possible faults that may have occurred. Each state in the diagnoser carries a label indicating whether or not a fault might have occurred. It turns out that we can use our model of overall behavior given as G itself as a diagnoser. The finite state machine G is an "off-line"

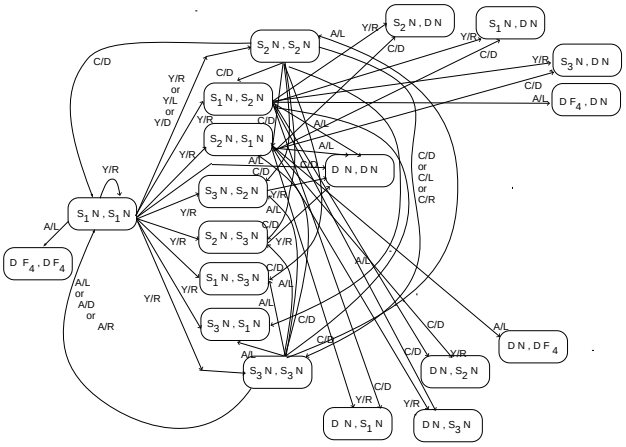

Fig. 10. $G_{F_4} \parallel G_{F_4}$

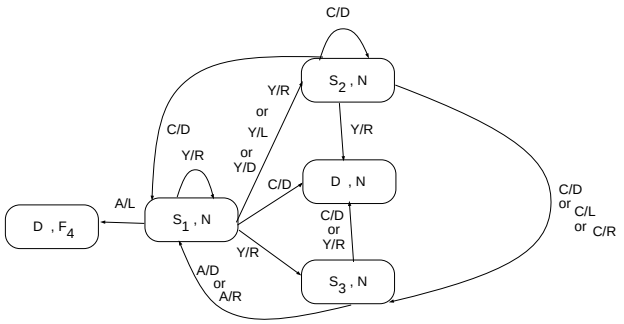

Fig. 11. NDFSM G'_{F_4}

diagnoser as it models all the behaviors, and identifies fault for all possible behaviors. One does not need to consider the entire G for performing diagnosis on-line. Only the part reachable by the observed input-output sequence needs to be considered.

A diagnoser can be local or global, where a local diagnoser observes the input-output sequence of one node only, and a global diagnoser observes the input-output sequence of more than one node. An example of local diagnoser is shown in Figure 13, where the local observation occurs at the sender node only. An example of a global diagnoser is shown in Figure 14, where it observes the behavior at both the nodes.

The local diagnoser is built as follows:

Algorithm 5: (For local diagnoser)

1) Initial state is given by (S_1, N).
2) On observation of input-output pair i_k/o_k at state x_k with label l_k, the reachable states are computed using the transition function of G.

Example 4: Consider local observation at sender end. When the observation Y/R, C/D, A/L, C/D occurs, the reachable part of automaton G is given in Figure 15. As can be seen from Figure 15 at this point either F_7 has occurred

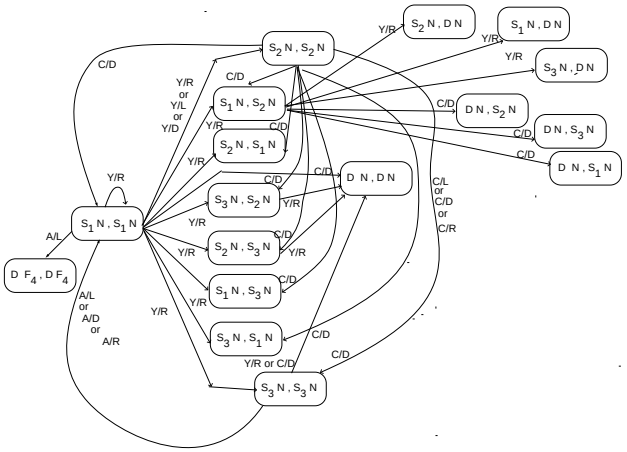

Fig. 12. $G'_{F_4} \parallel G'_{F_4}$

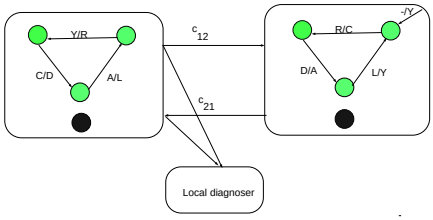

Fig. 13. Architecture of a local diagnoser

or fault F_6 has occurred.

A global diagnoser can be built by performing an input-output synchronization of the local observers of the nodes being observed. For simplicity we consider a global diagnoser that observes a pair of nodes. For such a global observer we use the following notation: x_i^j denotes a state i at node j, F_i^j denotes a fault i at node j, $(i_k^j/o_k^j, i_k^{j\prime}/o_k^{j\prime})$ denotes the kth input-output pair at nodes j and $j\prime$.

Algorithm 6: (For global diagnoser)

1) Initial state is $((S_1, N), (S_1, N))$.
2) On kth observation $(i_k^j/o_k^j, i_k^{j'}/o_k^{j'})$ at states $(x_k^j, l_k^j), (x_k^{j'}, l_k^{j'})$ of the nodes j and j' a transition to a new state-pair occurs according to transition function of G.

Example 5: When the observation Y/R, C/D, A/L, C/D occurs at node 1, and -/Y, R/C, D/A, L/C occurs at node 2, the reachable part of automaton G is given in Figure 16. As can be seen from Figure 16 that at this point the fault F_7^2 has occurred.

IV. Conclusions

In our work we have shown how discrete event system can be used to do passive fault detection and diagnosis in communication networks. The network nodes were modeled as finite state machines and the faults considered were output, input, and transition faults. For the example

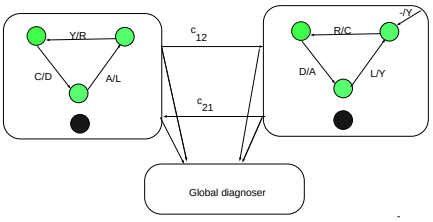

Fig. 14. Architecture of a global diagnoser

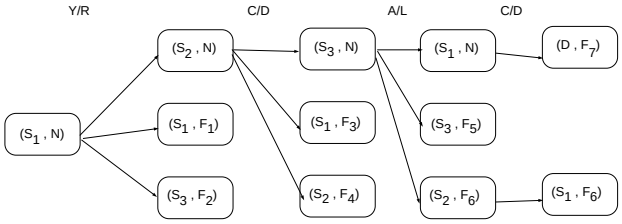

Fig. 15. A local diagnoser

we considered, we find that all the faults are detectable; output faults are diagnosable, but input, and transitions faults are not diagnosable. Also, by using the algorithm for detectability/diagnosability, the traces which are not detectable/diagnosable can be identified. After removing such traces a detectable/diagnosable sub-system can be obtained. In the algorithm used to build the diagnoser we did not need to perform any backward analysis as is the case with the work of [8]. We introduce a method to obtain a global diagnoser, by extending the notion of local diagnoser.

References

[1] L. E. Holloway and S. Chand. Distributed fault monitoring in manufacturing systems using concurrent discrete-event observations. *Integrated Computer-Aided Engineering*, 3(4), 1996.

[2] Z. Huang, V. Chandra, S. Jiang, and R. Kumar Modeling of discrete event systems with faults using a rules based modeling formalism. *Mathematical and Computer Modeling of Dynamical Systems*, 9(3):233–254,2003.

[3] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 46(8):1318-1321, 2001.

[4] S. Jiang and R. Kumar. Failure diagnosis of discrete event systems with linear-time temporal logic fault specifi cations. *IEEE Transactions on Automatic Control*, To appear, 2004.

[5] S. Jiang, R. Kumar, and H. E. Garcia. Diagnosis of repeated failures in discrete event systems. *IEEE Transactions on Robotics and Automation*, 19(2):310-323, 2003.

[6] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, Boston, MA, 1995.

[7] D. Lee, A. Netravali, K. Sabnani, B. Sugla, and A. John. Passive test and application to network management. In *Proceedings of 1997 IEEE International Conference on Network Protocols*, 1997.

[8] R. Miller and K. Arisha. Fault identifi cation in networks using a cfsm model by passive testing. Technical report, Department of Computer Science University of Maryland, College Park, USA, 2001.

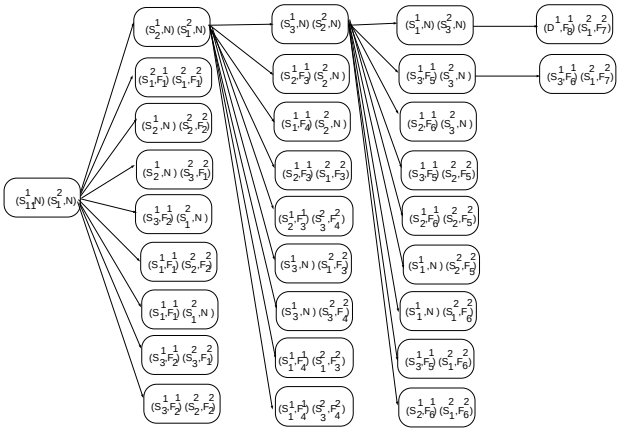

Fig. 16. A global diagnoser

[9] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, September 1995.

[10] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis Failure Diagnosis Using Discrete Event Models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.