Active Diagnosis of Discrete-Event Systems

Meera Sampath, *Member, IEEE*, Stéphane Lafortune, *Senior Member, IEEE*, and Demosthenis Teneketzis, *Senior Member, IEEE*

***Abstract*—The need for accurate and timely diagnosis of system failures and the advantages of automated diagnostic systems are well appreciated. However, diagnosability considerations are often not explicitly taken into account in the system design. In particular, design of the controller and that of the diagnostic subsystem are decoupled, and this may significantly affect the diagnosability properties of a system. In this paper the authors present an *integrated approach to control and diagnosis*. More specifically, they present an approach for the design of *diagnosable systems* by appropriate design of the system controller. This problem, which they refer to as the *active diagnosis* problem, is studied in the framework of discrete-event systems (DES's); it is based on prior and new results on the theory of diagnosis for DES's and on existing results in supervisory control under partial observations. They formulate the active diagnosis problem as a supervisory control problem where the legal language is an "appropriate" regular sublanguage of the regular language generated by the system. They present an iterative procedure for determining the supremal controllable, observable, and diagnosable sublanguage of the legal language and for obtaining the supervisor that synthesizes this language. This procedure provides both a controller that ensures diagnosability of the closed-loop system and a diagnoser for online failure diagnosis. The procedure can be implemented using finite-state machines and is guaranteed to converge in a finite number of iterations. The authors illustrate their approach using a simple pump–valve system.**

***Index Terms*—Discrete-event systems, failure diagnosis, finite-state machines.**

I. Introduction

THE NEED for accurate and timely diagnosis of system failures, in the interests of safety, reliability, and economy, has prompted widespread interest in the area of failure diagnosis both in industry and in academia. A great deal of research effort has been and is being spent on the design and development of automated diagnostic systems. A variety of schemes, differing both in their theoretical framework and in their design and implementation philosophy, have been proposed. From the conceptual viewpoint most existing methods of failure diagnosis can be classified as: 1) fault-tree based methods; 2) quantitative, analytical model-based methods; 3) expert systems; 4) model-based reasoning methods; and 5) discrete-event system (DES)-based methods (see [9], [11], and the references therein). From the implementation standpoint these diagnostic systems can be classified as offline or online. Offline methods assume that the system is in a testbed and is to be tested for possible prior failures, while in online diagnosis, the system is assumed to be operational and the diagnostic subsystem is designed so as to continuously monitor the system behavior as well as identify and isolate failures.

In most industrial systems, design of the online monitoring and diagnostic subsystem is done *after* the initial system design, and the diagnostic subsystem is added on as a separate module to the existing system. In other words, diagnosability considerations are often not explicitly taken into account in the system design. In particular, design of the controller and that of the diagnostic system are decoupled. Depending on the nature of the controller and the system, this decoupling can significantly affect the diagnosability properties of the system. As we shall see later in this paper (cf., Section V), when diagnosability is not a design specification, it is possible to design two different control protocols for a given system such that they both achieve all the desired design objectives, and yet one of these may result in a diagnosable system (a system in which it is possible to detect and isolate occurrences of failures) while the other may result in a nondiagnosable system.

In this paper we present an integrated approach to control and diagnosis, which we refer to as the *active diagnosis* problem. The term active is used to distinguish this method from passive diagnosis wherein the role of the diagnostic module is simply to observe the system behavior and draw inferences about potential failures; the active diagnosis problem, on the other hand, is one of combined observation and control. Almost universally, "control" in the context of failure diagnosis has referred to the notion of "testing" or "probing," and is more of an offline diagnosis problem; potential prior failures of the system are diagnosed by issuing test commands and observing the system responses. The challenge in these problems is the design of the "probe sequence" or the "test vector." While the active diagnosis problem studied in this paper can be used to generate test vectors for diagnosis, the work in this paper differs significantly from prior work in this area in its basic philosophy. *The emphasis here is on the design of diagnosable systems by appropriate design of the system controller.*

We study the active diagnosis problem in the framework of DES's. This paper is built on the theory of diagnosis for DES's developed in [10] and [11] and on the rich body of literature on supervisory control of DES's (cf., [8], [13], and

Manuscript received March 7, 1997; revised November 6, 1997. Recommended by Associate Editor, E. K. P. Chong. This work was supported in part by the NSF under Grants ECS-9057967, ECS-9312134, NCR-9204419, and ECS-9509975 and by ARO under Grant DAAH04-96-1-0377.

M. Sampath is with the Joseph C. Wilson Centre for Research and Technology, Xerox Corporation, Webster, NY 14580 USA.

S. Lafortune and D. Teneketzis are with the Department of EECS, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: stephane@eecs.umich.edu.

Publisher Item Identifier S 0018-9286(98)04896-X.

[2]). In [10] and [11] the authors propose an approach to failure diagnosis where the system is modeled as a DES in which the failures are treated as unobservable events; diagnosis is the process of detecting occurrences of these events from observed event sequences. The level of detail in a discrete-event model appears to be quite adequate for a large class of systems and for a wide variety of failures to be diagnosed. The approach is applicable whenever failures cause a distinct change in the system status but do not necessarily bring the system to a halt. In [10] the authors provide a definition of diagnosability in the framework of formal languages and establish necessary and sufficient conditions for diagnosability of systems. Also presented in [10] is a systematic approach for online diagnosis of failures using *diagnosers*.

The work in [10] and [11] deals with passive diagnosis. In this paper, we are interested in using control actions to alter the diagnosability properties of a given system, i.e., in restricting the behavior of a nondiagnosable system by appropriate control, to obtain a diagnosable system. The control issues are posed and addressed in the framework of supervisory control theory. Supervisory control theory deals with the design of controllers for a given DES that ensures that the controlled system meets certain qualitative specifications. These specifications define the *legal language* for the system. A *supervisor* for a DES is then an external agent or controller that, based on its partial view of the system, dynamically enables or disables the controllable events of the system in order to ensure that the resulting closed-loop language lies within the legal language. For an introduction to the basic ideas of supervisory control theory, we refer the reader to [8], [13], and [2] and the references therein.

Proceeding along the lines of the standard supervisory control problem, we adopt the following procedure to solve the active diagnosis problem. Given the nondiagnosable language generated by the system of interest, we first select an "appropriate" sublanguage of this language as the legal language. Choice of the legal language is a design issue and will typically depend on considerations such as acceptable system behavior (which ensures that we do not restrict the system behavior more than necessary in order to eventually make it diagnosable) and detection delay for the failures. As we shall see later in this paper (cf., Section IV-E), the diagnoser can provide guidelines on the choice of this legal language. Once the appropriate legal language is chosen, we then design a controller, which we refer to as a *diagnostic controller*, that achieves a closed-loop language that is within the legal language and is diagnosable. This controller can be designed based on the formal framework and the synthesis techniques that supervisory control theory provides, with the additional constraint of diagnosability. Recalling that the standard problem of supervisory control under partial observations can be stated as the problem of determining the supremal controllable and observable sublanguage[1] of the legal language, the active diagnosis problem can also be stated as determining the supremal controllable, observable, and diagnosable sublanguage[1] of the legal language and the supervisor that synthesizes this language. We emphasize that the active diagnosis problem is nontrivial for the following reasons: 1) the union of diagnosable languages need not be a diagnosable language; 2) a subset of a diagnosable language need not be diagnosable; and 3) the active diagnosis problem does not rule out terminating traces. The last reason motivates the study of nonlive languages in Section III.

We note that the starting point of the active diagnosis problem can either be an uncontrolled physical system or process, or it can be a controlled system itself. An example of the latter case is where the system has to satisfy prespecified legal language constraints, in addition to being diagnosable. In this case, a controller that ensures legality can first be designed following the usual supervisor design procedures, and the closed-loop language generated by the physical system with the above supervisory controller can serve as the starting point for the active diagnosis problem.

In Section II of this paper we present the necessary background for this paper; in particular we present the system model, discuss the notation that we shall use in the rest of the paper, and review the main results of [10] on diagnosability of DES's. In Section III we discuss diagnosability issues pertaining to nonlive languages. Section IV presents the main results of this paper: the formulation of the active diagnosis problem, a solution procedure along with details of its implementation, and an illustrative example. In Section V we demonstrate how the theory developed in this paper can be used to design a diagnostic controller for a simple pump–valve system. Finally, we give some concluding remarks in Section VI.

II. Background and Notation

A. The System Model

The system of interest is modeled as a deterministic finite-state machine (FSM) or generator

$$G = (X, \Sigma, \delta, x_0) \tag{1}$$

where, as usual (cf., [8]), X is the finite state space, Σ is the finite set of events, δ is the partial transition function, and x_0 is the initial state of the system. The model G accounts for the normal *and* failed behavior of the system. The behavior of the system is described by the *prefix-closed language* [8] $L(G)$ generated by G. Henceforth, we shall denote $L(G)$ by L. L is a subset of Σ^*, where Σ^* denotes the Kleene closure [4] of the set Σ and includes the empty trace denoted by ϵ. Note that the language L is regular [4] since X is finite.

The event set Σ is partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o represents the set of observable events and Σ_{uo} represents the set of unobservable events. Σ is also partitioned as $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, where Σ_c represents the set of controllable events and Σ_{uc} represents the set of uncontrollable events.

Let $\Sigma_f \subseteq \Sigma$ denote the set of failure events which are to be diagnosed. We assume, without loss of generality, that $\Sigma_f \subseteq \Sigma_{uo}$, since an observable failure event can be trivially diagnosed. Further, we partition the set of failure events into disjoint sets corresponding to different failure types

$$\Sigma_f = \Sigma_{f1} \dot{\cup} \cdots \dot{\cup} \Sigma_{fm}. \tag{2}$$

[1] Whenever such a supremal element exists.

Let Π_f denote this partition. The partition is motivated primarily by diagnostic requirements. Often, we may not require that every failure event be identified uniquely; we may simply be interested in knowing if one of a set of failure events has happened, as for example, when the effect of the set of failures on the system is the same. In other words, we require unique identification not of the failure event itself, but of the type of failure, when such an event occurs in the system. Hereafter, when we write that "a failure of type F_i has occurred" we will mean that some event from the set Σ_{fi} has occurred.

We make the following assumptions about the system under investigation.

- A1) The language L generated by G is *live*. This means that there is a transition defined at each state x in X, i.e., the system cannot reach a point at which no event is possible.
- A2) There are no cycles of unobservable events in G, i.e., $\exists n_o \in N$ such that $(\forall st \in L)\ s \in \Sigma_{uo}^* \Rightarrow \|s\| \leq n_o$ where $\|s\|$ denotes the length of trace s.
- A3) $\Sigma_c \subseteq \Sigma_o$, i.e., no unobservable event can be prevented from occurring by control.

The liveness assumption on L is made for the sake of simplicity. In Section III of this paper, we discuss diagnosability issues pertaining to nonlive languages. Assumption A2) ensures that observations occur with some regularity. Since detection of failures is based on observable transitions of the system, we require that G does not generate arbitrarily long sequences of unobservable events. Assumption A3) is essential for the existence of a solution to the active diagnosis problem as posed in this paper. Under Assumption A3), controllability and normality together imply observability.[2]

B. Notation

In this paper, we assume that all the languages of interest are prefix-closed. Let $\bar{s}$ denote the prefix-closure of the set $\{s\} \subset \Sigma^*$. Let $L(G, x)$ denote the set of all traces that originate from state x of G. We denote by L/s the postlanguage of L after s, i.e.,

$$L/s = \{t \in \Sigma^* \mid st \in L\}. \tag{3}$$

Let s_f denote the final event of trace s. We define

$$\Psi(\Sigma_{fi}) = \{s \in L : s_f \in \Sigma_{fi}\} \tag{4}$$

i.e., $\Psi(\Sigma_{fi})$ denotes the set of all traces of L that end in a failure event belonging to the set Σ_{fi}. Consider $\sigma \in \Sigma$ and $s \in \Sigma^*$. We use the notation $\sigma \in s$ to denote the fact that σ is an event in the trace s. With slight abuse of notation, we write $\Sigma_{fi} \in s$ to denote the fact that $\sigma_f \in s$ for some $\sigma_f \in \Sigma_{fi}$, or, formally, $\bar{s} \cap \Psi(\Sigma_{fi}) \neq \emptyset$.

Let P: $\Sigma^* \rightarrow \Sigma_o^*$ denote the usual projection operator that "erases" the unobservable events in a trace [8]. The inverse projection operator P_L^{-1} is defined as

$$P_L^{-1}(y) = \{s \in L : P(s) = y\}. \tag{5}$$

[2]This result has been proved in the unpublished literature [6].

Given a system G, a partial observation supervisor S_P for G is a map

$$S_P : P[L(G)] \rightarrow \{\tau \in 2^{\Sigma} : \Sigma_{uc} \subseteq \tau\}.$$

In other words, S_P gives the set of enabled events following any observed event sequence. The resulting closed-loop system is denoted by S_P/G. A realization of the supervisor S_P for G which achieves $L(S_P/G) = K$ is given by the pair (R, φ) where $R = (X_R, \Sigma_o, \delta_R, x_{R,0})$ is a recognizer for $P(K)$ and $\varphi : X_R \rightarrow \{\tau \in 2^{\Sigma} : \Sigma_{uc} \subseteq \tau\}$. Under the assumption that $\Sigma_c \subseteq \Sigma_o$, the map $\varphi(x)$ can simply be given by the active event set of R at state x. In this case the supervisor S_P may simply be realized by the FSM R, the feedback map φ being implicit in the transition structure of R.

Given a language M over the alphabet Σ and a language $K \subseteq M$ we denote by $K^{\uparrow C}$ the supremal controllable sublanguage of K with respect to M and $\Sigma_{uc} \subseteq \Sigma$ (where the appropriate M and Σ_{uc} are understood by context). Likewise we denote by $K^{\uparrow CO}$ the supremal controllable and observable sublanguage of K with respect to M, P, and Σ_{uc} (whenever this language exists). We denote by $K^{\uparrow N}$ the supremal normal sublanguage of K with respect to M and P and by $K^{\uparrow CN}$ the supremal controllable and normal sublanguage of K with respect to M, P, and Σ_{uc}.

We use the notation $G_1 \sqsubseteq G_2$ to denote that the FSM G_1 is a submachine (i.e., subgraph) of G_2 (cf., [5]). Finally, given two FSM's G_1 and G_2, we take $G_1 = G_2$ to mean that G_1 is identical to G_2 up to a renaming of the states.

C. Diagnosability and Diagnosers

1) The Notion of Diagnosability: In this section, we summarize some of the main results of [10] on diagnosability of DES's that will be frequently referred to in this paper. In particular, we discuss briefly the notion of diagnosability, the structure of diagnosers, and the necessary and sufficient conditions for diagnosability. For a detailed treatment of these ideas we refer the reader to [10].

Roughly speaking, a language L is diagnosable if it is possible to detect with a finite delay occurrences of failures of any type using the record of observed events. Here finite delay is taken to mean a finite number of event occurrences following the failure. Formally, we define diagnosability as follows:

Definition 1: A prefix-closed and live language L is said to be *diagnosable* with respect to the projection P and with respect to the partition Π_f on Σ_f if the following holds:

$$(\forall i \in \Pi_f)(\exists n_i \in N)(\forall s \in \Psi(\Sigma_{fi}))(\forall t \in L/s)[\|t\| \geq n_i \Rightarrow D]$$

where the diagnosability condition D is

$$\omega \in P_L^{-1}[P(st)] \Rightarrow \Sigma_{fi} \in \omega.$$

Let s be any trace generated by the system that ends in a failure event of a particular type, say, F_i, and let t be any sufficiently long continuation of s. Diagnosability then requires

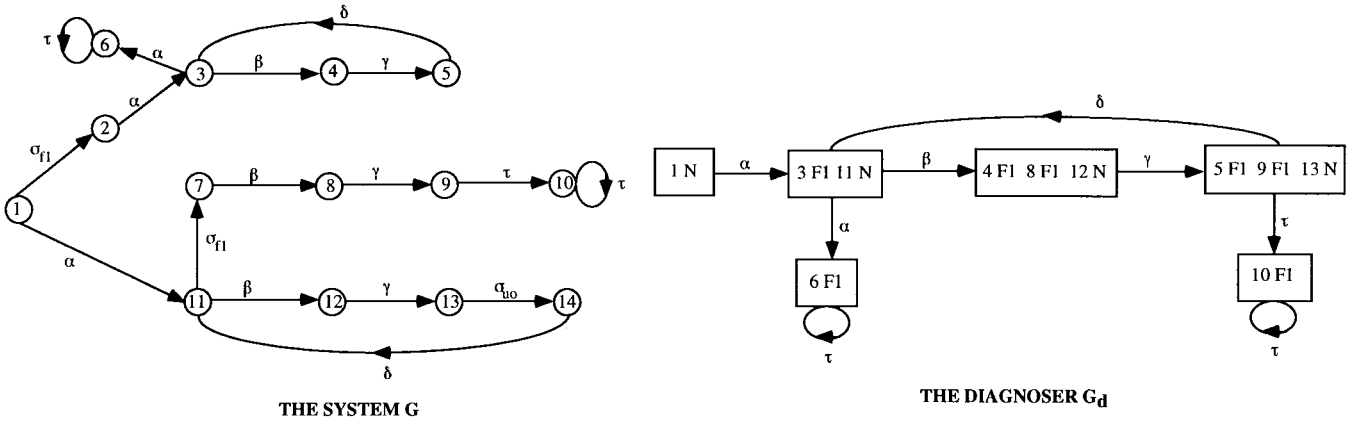

Fig. 1. Example of a system with an F_1-indeterminate cycle in its diagnoser G_d.

that every trace belonging to the language that produces the same record of observable events as the trace st should contain in it a failure event of the type F_i. This implies that along every continuation t of s one can detect the occurrence of a failure of the type F_i with a finite delay. In other words, diagnosability requires that every failure event leads to observations distinct enough to enable unique identification of the failure type with a finite delay. Note that continuations t which are of length less than n_i need not be considered in Definition 1 since we allow for a finite delay (of up to n_i events) in detecting the failure. Also note that this definition applies to any language $L \subseteq \Sigma^*$, regular or not.

2) The Diagnoser: Given a DES represented by an FSM G, the diagnoser for the system is the deterministic FSM $G_d = (Q_d, \Sigma_o, \delta_d, q_0)$ with $L(G_d) = P(L)$. The diagnoser G_d can be thought of as an *extended observer* for G which gives: 1) an estimate of the current state of the system after the occurrence of every observable event and 2) information on potential past failure occurrences in the form of failure labels attached to the state estimates. The diagnoser performs diagnostics when it observes online the behavior of G. Failures are diagnosed by simply checking the labels associated with the state estimates; if the diagnoser transitions into a state q such that every state estimate in q contains the label F_i, then we conclude for certain that a failure of type F_i has occurred regardless of what the current state of G is. We refer to such a state q as an F_i-*certain* state. Likewise, an F_i-uncertain state of the diagnoser is one that contains at least one state estimate which carries the label F_i and at least one estimate which does not carry the label F_i.

3) The Conditions for Diagnosability: The diagnoser G_d is used not only to perform online diagnostics, but also to verify offline the diagnosability of the system. In other words, the necessary and sufficient conditions for diagnosability of a given language can be stated as conditions on its diagnoser. The notion of an F_i-indeterminate cycle in the diagnoser is the most crucial element in the development of necessary and sufficient conditions for diagnosability and is central to the problem of active diagnosis studied in this paper. Informally speaking, an F_i-indeterminate cycle in G_d is a cycle composed exclusively of F_i-uncertain states for which there exist:

1) a corresponding cycle (of observable events) in G involving only states that carry F_i in their labels in the cycle in G_d and states that are reached only via unobservable events;
2) a corresponding cycle (of observable events) in G involving only states that do *not* carry F_i in their labels in the cycle in G_d and states that are reached only via unobservable events.

Example 2.1: Consider the system G and the corresponding diagnoser G_d represented in Fig. 1. Let $\Sigma_f = \Sigma_{f1} = \{\sigma_{f1}\}$ and let $\Sigma_{uo} = \Sigma_f \cup \{\sigma_{uo}\}$. The diagnoser in Fig. 1 has a cycle of F_1-uncertain states with the corresponding event sequence $\beta\gamma\delta$. Corresponding to this cycle in the diagnoser, there are two cycles in the state machine G: the first involves states 3–5 which appear with an F_1 label in the cycle in the diagnoser, and the second involves states 11–13 which carry an N label in the cycle in the diagnoser, and third, state 14 is reached via the unobservable event σ_{uo}. Thus, the cycle in G_d is an F_1-indeterminate cycle.

In the preceding example, the cycle in G_d corresponds directly to the cycles in G, in the sense that the cycles in G are completed with just one completion of the cycle in the diagnoser G_d. However, in general, more than one traversal of the F_i-indeterminate cycle in G_d may be required to complete one or both of the corresponding cycles in G. In this case we say that the cycles in G are of multiplicity greater than one. For further details we refer the reader to [10].

We now quote the following result from [10].

Theorem 1: A regular, prefix-closed, and live language L is diagnosable if and only if its diagnoser G_d satisfies the following condition: there are no F_i-indeterminate cycles in G_d, for all failure types F_i.

The condition of the above theorem, together with the liveness assumption on G, implies that every F_i-uncertain state in G_d leads to an F_i-certain state in a bounded number of transitions of the diagnoser. We then have the following corollary from [10].

Corollary 1: Consider a regular, prefix-closed, and live language L. Let n_o refer to the length of the longest sequence of unobservable events in L. Let $\Sigma_{fi}, i = 1, 2, \cdots, m$ denote disjoint sets of failure events in Σ. If L is diagnosable with

delay n_i corresponding to failure type F_i, then its diagnoser G_d detects occurrences of failure events of the type F_i in at most n_i+n_o events of L following the occurrence of a failure event of type F_i.

We conclude our review of the diagnosability property of DES's with the following remark. For the reader familiar with [10], the diagnoser G_d that we refer to in this paper is the diagnoser G_d^{mf} of [10]. The diagnoser G_d^{mf} was introduced in [10] to handle the case of multiple failures of the same type (possibly) occurring along any trace of L. In this paper, we do not distinguish between the cases of multiple failures and single failures of the same type, and hence, in order to simplify the presentation, we use the diagnoser G_d^{mf} of [10] for both of these cases.

III. On Diagnosability of Nonlive Languages

The theory of diagnosability developed in [10] is based on the assumption that the language L is live. Even when the system of interest G is such that it satisfies the liveness assumption, it is possible that when the behavior of the system is restricted under control, as in the case of the active diagnosis problem that is discussed in this paper, the language generated by the controlled system may not be live, since the use of control could result in blocking or deadlock. Hence, we now generalize the results of [10] to account for nonlive languages as well. In this section, we present the generalized definition of diagnosability and a procedure to check for diagnosability of nonlive languages using the results developed in [10] for live languages. In addition to allowing us to deal with nonlive languages arising as a result of control, the results of this section also allow us to deal with systems that are not live to begin with, i.e., systems that do not satisfy assumption A1) as we shall see from what follows. Note that the language L and the FSM G used in this section are generic and do not refer to the system model introduced in Section II-A and used in the active diagnosis problem studied in Section IV.

A. The Notion of Diagnosability for Nonlive Languages

Definition 2: A prefix-closed language L is said to be *diagnosable* with respect to the projection P and with respect to the partition Π_f on Σ_f if the following holds:

$$(\forall i \in \Pi_f)\ (\exists n_i \in \mathbb{N})(\forall s \in \Psi(\Sigma_{fi}))(\forall t \in L/s)$$
$$[(||t|| < n_i) \wedge (L/st = \emptyset) \Rightarrow D_1] \wedge [||t|| \geq n_i \Rightarrow D_2]$$

where the diagnosability conditions D_1 and D_2 are

$$D_1 : \big(\omega \in P_L^{-1}[P(st)]\big) \wedge \big(L/\omega = \emptyset\big) \Rightarrow \Sigma_{fi} \in \omega$$

and

$$D_2 : \omega \in P_L^{-1}[P(st)] \Rightarrow \Sigma_{fi} \in \omega.$$

The above definition of diagnosability is the same as Definition 1 in Section II-C (for live languages) except that we now require the diagnosability condition to hold for terminating traces as well. Note that even if the language L has a terminating trace s that ends in a failure event of type F_i, L may still be diagnosable as long as there does not exist in L a trace s' such that s' is also terminating and generates the same projection as the trace s but does not contain a failure event of type F_i. Note that as in the case of Definition 1, the above definition applies to regular as well as nonregular languages.

In order to diagnose failures in a nonlive system and to check for diagnosability of the language it generates, we use the results developed in [10] for live languages as follows. Given a nonlive language L we extend it to a live language L^{live} by adding a new event "Stop" to Σ where $\text{Stop} \in \Sigma_o \cap \Sigma_{uc}$ and by defining

$$L^{\text{live}} = L \cup_{i \geq 0} \{s\,\text{Stop}^i : L/s = \emptyset\}. \tag{6}$$

It is obvious from the above definition that L^{live} is live. By comparing Definition 2 above with [10, Definition 1], it can be seen that

$$L \text{ diagnosable } \Leftrightarrow L^{\text{live}} \text{ diagnosable}. \tag{7}$$

In other words, checking for the diagnosability of L is equivalent to checking for the diagnosability of L^{live}.

We conclude our discussion on nonlive languages and diagnosability with the following observation.[3]

Proposition 1: A sublanguage M of a diagnosable language K is itself diagnosable if all terminating traces of M are also terminating traces of K.

Proof: The proof is by contradiction. Suppose that M is not diagnosable. Then there exist at least two traces $s_1, s_2 \in M$ such that either D_1 or D_2 (in the above Definition 2) is violated. If D_1 is violated, i.e., s_1 and s_2 are terminating traces, then by assumption, s_1 and s_2 are also terminating traces in K; thus K is not diagnosable. If D_2 is violated by s_1 and s_2, then K is also nondiagnosable since s_1 and s_2 are in K. □

B. The Diagnoser G_d^{live}

For the remainder of this section on diagnosability of nonlive languages, we assume that L is regular. Let $G^{\text{live}} = (X, \Sigma \cup \{\text{Stop}\}, \delta^{\text{live}}, x_0)$ be the FSM generating L^{live}. The transition function δ^{live} of G^{live} is defined as follows. First, let

$$X^{\text{dead}} = \{x \in X : \delta(x, \sigma) \text{ is undefined } \forall \sigma \in \Sigma\}. \tag{8}$$

Then define

$$\delta^{\text{live}}(x, \sigma) = \delta(x, \sigma) \quad \forall x \in X - X^{\text{dead}}, \qquad \forall \sigma \in \Sigma \tag{9}$$

$$\delta^{\text{live}}(x, \text{Stop}) = \begin{cases} x, & \text{if } x \in X^{\text{dead}} \\ \text{undefined}, & \text{otherwise.} \end{cases} \tag{10}$$

Thus, G^{live} is obtained from G by adding at every dead state of G, a self-loop due to the event Stop. It is straightforward to see that $L(G^{\text{live}}) = L^{\text{live}}$. We then construct from G^{live} the diagnoser $G_d^{\text{live}} = (Q_d^{\text{live}}, \Sigma_o \cup \text{Stop}, \delta_d^{\text{live}}, q_0)$ corresponding to L^{live}. This diagnoser is used to perform online failure diagnosis of the system G; also the diagnosability of L can be tested by checking for indeterminate cycles in G_d^{live} (as per Theorem 1). Henceforth, we will refer to G_d^{live} as the "live" diagnoser of G (or, of L).

[3] We are indebted to an anonymous reviewer for making this observation.

1) Procedure to Construct G_d^{live} from G_d: While the diagnoser G_d^{live} may be built from G^{live} following the usual construction procedure of the diagnoser, it may also be obtained as a simple extension of $G_d = (Q_d, \Sigma_o, \delta_d, q_0)$, the diagnoser corresponding to G, if G_d is already available. First, note that the only difference between G_d^{live} and G_d is due to the Stop events defined at the dead states X^{dead} of G. Next, for $q \in Q_d$ define

$$D(q) = \{(x,\ell) \in q : (\forall u \in L(G,x)) \\ [u \in \Sigma_{uo}^* \wedge \delta(x,u) \in X^{\text{dead}}]\}. \tag{11}$$

Let $Q^{\text{dead}}(G_d)$ denote the set of all states q in G_d such that $D(q)$ is nonempty. Note that $q \in Q^{\text{dead}}(G_d)$ does not necessarily imply that q is a dead state of G_d; it simply means that there exists an (x,ℓ) pair in q where x is such that no further observable event is possible in G once G gets into state x. Consider any $q \in Q^{\text{dead}}(G_d)$ and any $(x,\ell) \in D(q)$. Let $\delta(x,u) = y$; since $y \in X^{\text{dead}}$, then, in G^{live}, the corresponding live machine, we have $\delta^{\text{live}}(x, u\text{Stop}) = y$ and $\delta^{\text{live}}(y, \text{Stop}) = y$. This implies that in the diagnoser G_d^{live} corresponding to G^{live}

$$\delta_d^{\text{live}}(q, \text{Stop}) = L(q) \tag{12}$$

and

$$\delta_d^{\text{live}}(L(q), \text{Stop}) = L(q) \tag{13}$$

where

$$L(q) = \{(y,\ell') : (\delta^{\text{live}}(x, u\text{Stop}) = y)(\ell' = \tilde{LP}(\ell,u)) \\ \text{for some } (x,\ell) \in D(q), u \in \Sigma_{uo}^*\} \tag{14}$$

and[4]

$$\tilde{LP}(\ell,u) = \begin{cases} \ell, & \text{if } \forall i(\Sigma_{fi} \notin u) \\ \ell \cup \{F_i : \Sigma_{fi} \in u\}, & \text{otherwise.} \end{cases}$$

Finally, note that if q is such that $\forall (x,\ell) \in q$, $x \in X^{\text{dead}}$, then the state $L(q)$ is the same as q, i.e., $\delta_d^{\text{live}}(q, \text{Stop}) = q$. Thus, we see that G_d^{live} is identical to G_d except that at each state $q \in Q^{\text{dead}}(G_d)$ as above, we either have a self-loop due to the Stop event, or, we have a transition defined due to the Stop event to a state $L(q)$ as above and a self-loop at $L(q)$ due to the Stop event. Therefore, once the set $Q^{\text{dead}}(G_d)$ is identified, G_d^{live} can be built simply as an extension of G_d. We will refer to G_d^{live} as the "live diagnoser extension" of G_d.

2) Properties of G_d^{live}: We now state some properties of the live diagnoser G_d^{live} that we shall find useful in the solution of the active diagnosis problem studied in this paper.

P1) Let t be any terminating trace of L and $p \in \bar{t}$ such that $p_f \in \Sigma_o$ and $t/p \in \Sigma_{uo}^*$. Then there exists $q' \in Q_{\text{dead}}(G_d)$ such that $\delta_d(q_0, P(p)) = \delta_d(q_0, P(t)) = \delta_d^{\text{live}}(q_0, P(p)) = q'$; $\delta_d^{\text{live}}(q', \text{Stop}) = q$ and $\delta_d^{\text{live}}(q, \text{Stop}) = q$.

Proof: The proof is straightforward from the definition of the language L^{live} and from the construction of the diagnoser G_d^{live}. □

P2) Let t be any terminating trace in L such that $\Sigma_{fi} \in t$ and such that t violates the diagnosability condition D_1 of Definition 2. Then the state $q = \delta_d^{\text{live}}(q_0, P(t\,\text{Stop}))$ forms an F_i-indeterminate cycle in G_d^{live} due to the single event Stop.

Proof: Since t a terminating trace in L, $\Sigma_{fi} \in t$, and t violate diagnosability, then $\exists\ \omega \in P_L^{-1}[P(t)]$ such that $L/\omega = \emptyset$ and $\Sigma_{fi} \notin \omega$. Let $\delta(x_o, t) = x, \delta(x_o, \omega) = y$. Then we have $(x,\ell) \in q, (y,\ell') \in q$, $F_i \in \ell$, and $F_i \notin \ell'$. Thus, q is F_i-uncertain. From the definitions of the language L^{live} and the FSM G^{live} and from the construction of the diagnoser G_d^{live}, we have $\delta_d^{\text{live}}(q, \text{Stop}) = q$, $\delta(x, \text{Stop}) = x$, and $\delta(y, \text{Stop}) = y$. Therefore, the state q forms an F_i-indeterminate cycle due to the event Stop. □

P3) Let t be a terminating trace of L such that $\Sigma_{fi} \in t$ and such that t does not violate the definition of diagnosability. Then the state $q = \delta_d^{\text{live}}(q_0, P(t\,\text{Stop}))$ is F_i-certain.

Proof: If t a terminating trace in L, $\Sigma_{fi} \in t$, and t do not violate the definition of diagnosability, then $\forall \omega \in P_L^{-1}(P(t))$ such that $\Sigma_{fi} \notin \omega$, we have $L/\omega \neq \emptyset$. In other words, $\forall \omega' \in P_L^{-1}(P(t))$ such that $L/\omega = \emptyset$, we have $\Sigma_{fi} \in \omega'$. Hence, in the live diagnoser G_d^{live}, the state $q = \delta_d^{\text{live}}(q_0, P(t\,\text{Stop}))$ is F_i-certain. □

Remark: Property P3) implies that if no observable event occurs in the system G after the diagnoser G_d enters the state $\delta_d(q_0, P(t))$, then we can conclude that a failure of type F_i has occurred. This inferencing is captured in the live diagnoser G_d^{live} by the fact that $\delta_d^{\text{live}}(q_0, P(t\,\text{Stop})) = \delta_d^{\text{live}}(q, \text{Stop})$ is an F_i-certain state. Thus, occurrence of the "artificial" Stop event at state q of the diagnoser G_d^{live}, which is the same as the occurrence of *no event* at the state of the "real" diagnoser G_d, leads us to diagnose the occurrence of a failure of type F_i.

We present the following definition of a Stop-indeterminate state that we shall use in the solution of the active diagnosis problem presented in the following section.

Definition 3: Let $q' \in Q_{\text{dead}}(G_d)$ such that $\delta_d^{\text{live}}(q', \text{Stop}) = q$ and $\delta_d^{\text{live}}(q, \text{Stop}) = q$. The state q' is defined to be a Stop-indeterminate state of the pair (G_d, G_d^{live}) if the state q forms an indeterminate cycle in the live diagnoser G_d^{live} due to the event Stop.

All the ideas of Section III-B are illustrated by the following example.

Example 3.1: Consider the nonlive system G of Fig. 2 with $\Sigma_f = \{\sigma_{f1}\}$ and $\Sigma_{uo} = \Sigma_f \cup \{\sigma_{uo}\}$. The live extension G_d^{live} of G, the diagnoser G_d corresponding to G, and the live diagnoser G_d^{live} of G are depicted in Fig. 2. Here $Q_{\text{dead}} = \{\{(4,F1),(10,F1),(3,N)\}, \{(5,F1), (11,N)\}\}$. For $q = \{(4,F1),(10,F1),(3,N)\}$, we have $D(q) = \{(7,N)\}$ and $\delta_d^{\text{live}}(q, \text{Stop}) = L(q) = \{(8,F1)\}$. For $q = \{(5,F1),(11,N)\}$, we have $D(q) = q$ and $L(q) = q$.

[4] $\tilde{LP}$ is equivalent to the label propagation function of the diagnoser defined in [10]; it propagates the failure information labels associated with a state to its successors.

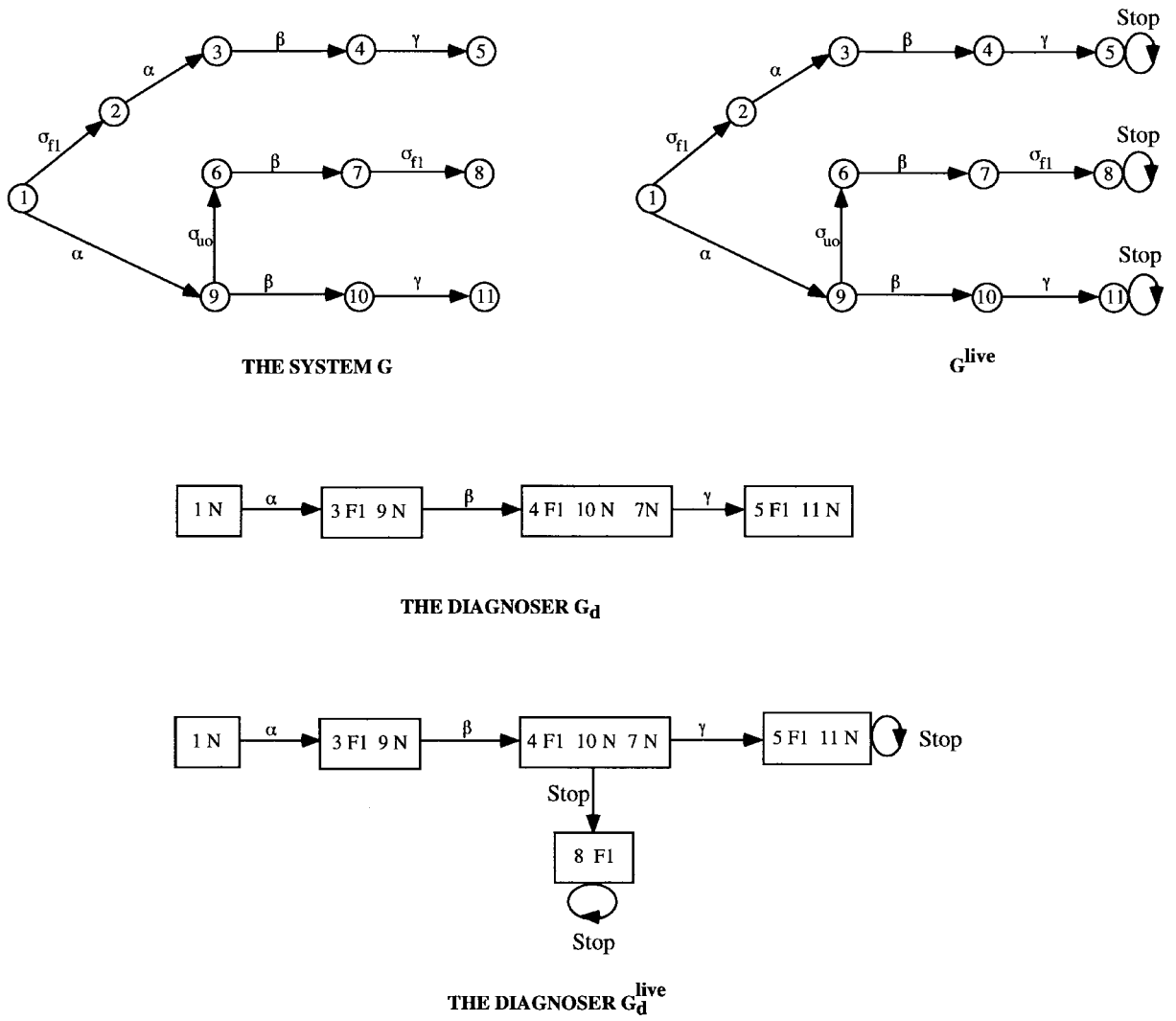

Fig. 2. The diagnoser G_d^{live} for a nonlive system G.

Consider the trace $t_1 = \sigma_{f1}\alpha\beta\gamma$. This trace violates condition D_1 of diagnosability since there exists $\omega \in L$ such that $\omega = \alpha\beta\gamma$, $P(t_1) = P(\omega)$, ω is a terminating trace, and $\Sigma_{fi} \notin \omega$. Note that the trace $P(t_1)$ leads to the F_1-uncertain state $\{(5, \{F1\}), (11, \{N\})\}$ of G_d^{live} with a self-loop due to the Stop event. It is easy to verify by inspection of G^{live} that this state forms an F_1-indeterminate cycle.

Consider next the trace $t_2 = \alpha\sigma_{uo}\beta\sigma_{f1}$ that is a terminating trace of L. It is easy to see by inspection of G that this trace t_2 does not violate condition D_1 of diagnosability. Further, we see that if the event sequence $\alpha\beta$ is observed, with no further event thereafter, then we can conclude for sure that the system executed the trace t_2; hence we can conclude that the failure σ_{f1} occurred. This diagnostic information can be obtained from the diagnoser G_d^{live} by noting that the trace t_2 followed by the Stop event leads to the F_1-certain state $\{(8, \{F1\})\}$.

This concludes our discussion on the diagnosability of nonlive languages. Based on the above discussion, we see that even when the language L generated by the system G is not live, it can be extended to a live language in a straightforward manner. Therefore, we note that Assumption A1) that we have made about the system G leads to no loss of generality.

IV. The Active Diagnosis Problem

In this section, we formulate the active diagnosis problem, propose a solution procedure, demonstrate its correctness, and finally illustrate the proposed solution procedure with an example.

A. Problem Formulation

We formulate the active diagnosis problem (ADP) as follows.

Active Diagnosis Problem: Given the regular, live language L generated by the system G (of Section II-A), and given a regular, normal (with respect to L and P) language $K \subseteq L$ such that every live sublanguage of K is diagnosable, find a (partial observation) supervisor S_P for G such that $L(S_P/G) = L^{\text{act}}$ where

C1) $L^{\text{act}} \subseteq K$;

C2) L^{act} is diagnosable;

C3) L^{act} is as large as possible.

From standard results on supervisory control under partial observations [7], we know that a supervisor S_P for G, such that $L(S_P/G) = L^{\text{act}}$, exists if and only if $L^{\text{act}} \neq \emptyset$ and L^{act} is controllable with respect to L and Σ_{uc} and observable with respect to L and P. Therefore, the ADP is to find the supremal controllable, observable, and diagnosable sublanguage of (the legal language) K. We note that: 1) diagnosability is not a property that is preserved under union of languages[5] and 2) a subset of a diagnosable language need not be diagnosable.[6] Therefore, the supremal diagnosable sublanguage of a given nondiagnosable language need not always exist. However, we will show later, *by a constructive proof*, that the desired supremal element does indeed exist under the assumptions made in this paper, i.e., the ADP is well formulated. We shall refer to the supervisor S_P that solves the above ADP as a *diagnostic controller* for G.

In view of Proposition 1, the assumption that every live sublanguage of K is diagnosable is equivalent to the assumption that the supremal live sublanguage of K is diagnosable. This condition can be checked by building the diagnoser for the supremal live sublanguage of K; this sublanguage is obtained by removing all terminating traces of K. In Section IV-E we will present a class of languages $K_l \subseteq L$, $l \geq 0$ that satisfy the above-mentioned properties of K, i.e., every live sublanguage of K_l, for any given l, is diagnosable, and K_l is normal with respect to L and P. This class of languages can be obtained using the diagnoser corresponding to L.

B. Solution Procedure

We first present a description of the solution procedure. The procedure computes the supremal controllable, normal, diagnosable sublanguage of K and consists of three stages. We start with the initial condition (language K) and at the first stage we compute the supremal controllable and normal[7] sublanguage of K. This calculation is presented in Module A of the procedure below and is based on a result of Brandt *et al.* [1, Th. 4]. At the second stage we compute the supremal diagnosable and normal sublanguage of the language resulting from the first stage. This computation is performed in Module B of the procedure below and uses the assumption that every live sublanguage of K is diagnosable and uses Properties P1) and P2) of Section III. The third stage (Module C) is a test for convergence; this is necessary since the construction described in the second stage may result in a language that is not controllable, and further iterations of the above procedure may be required to obtain a language that is controllable, normal, and diagnosable. It is shown in Theorem 2 that this solution procedure converges in a finite number of iterations to the supremal controllable, normal, and diagnosable sublanguage of K. The resulting language is also the supremal controllable, observable, and diagnosable sublanguage of K because whenever $\Sigma_c \subseteq \Sigma_o$ [Assumption A3)], normality, and controllability together imply observability as was mentioned in Section II-A.[8]

We now present the solution procedure. Implementation of this procedure is discussed in Section IV-C.

Initialization:

Step 0-1: Obtain an FSM generator of K, henceforth referred to as G^{legal}.

Step 0-2: Build the diagnoser G_d^{legal} corresponding to G^{legal}.

Step 0-3: Let $i = 0; H_d(0) = H_d^{\text{live}}(0) = G_d^{\text{legal}}$; and $M_d(0) = L(H_d(0))$.

Iteration:

Module A:

Step A-1: Compute the supremal controllable sublanguage $M_d^{\uparrow C}(i)$ of $M_d(i)$ with respect to $P(L)(= L(G_d))$ and $\Sigma_{uc} \cap \Sigma_o$.

Let $H_d^{\uparrow C}(i)$ denote the FSM generating $M_d^{\uparrow C}(i)$.

Step A-2: Compute $M(i) = P_L^{-1}[M_d^{\uparrow C}(i)]$. Let the FSM $H(i)$ be such that $L(H(i)) = M(i)$.

Module B:

Step B-1: Let $k = 0$; $\tilde{H}(0) = H(i)$; $\tilde{M}(0) = L(\tilde{H}(0)) = M(i)$; and $\tilde{H}_d(0) = H_d^{\uparrow C}(i)$;

Step B-2: Extend $\tilde{M}(k)$ to the live language $\tilde{M}^{\text{live}}(k)$ as per (6) in Section III-A. Let $\tilde{H}^{\text{live}}(k)$ denote the FSM that generates $\tilde{M}^{\text{live}}(k)$, and let $\tilde{H}_d^{\text{live}}(k)$ denote the diagnoser corresponding to $\tilde{H}^{\text{live}}(k)$.

Step B-3: Eliminate from $\tilde{H}_d(k)$ all states q such that q is a Stop-indeterminate state of the pair $(\tilde{H}_d(k), \tilde{H}_d^{\text{live}}(k))$; let $\tilde{H}_d(k+1)$ denote the accessible part of the resulting machine.

Step B-4: If $\tilde{H}_d(k+1) = \tilde{H}_d(k)$, then let $H_d(i+1) = \tilde{H}_d(k)$; let $M_d(i+1) = L(H_d(i+1))$; let $H_d^{\text{live}}(i+1) = \tilde{H}_d^{\text{live}}(k)$ and go to Step C-1. Else let $\tilde{M}(k+1) = P_L^{-1}(L(\tilde{H}_d(k+1)))$; let $\tilde{H}(k+1)$ be the FSM generator of $\tilde{M}(k+1)$; let $k = k+1$; and go to Step B-1.

Module C:

Step C-1: If $H_d(i+1) = H_d(i)$, stop. The solution to the ADP is $L^{\text{act}} = M(i)$ and the corresponding supervisor S_P is realized by the FSM $H_d(i)$. Else go to Step A-1.

C. Implementation of the Solution Procedure

We explain how to implement, using FSM's, all the steps of the preceding procedure. The relationships between the various FSM's in the procedure are depicted in Fig. 3; in that figure, solid lines indicate actual computations while dotted dashed lines indicate relationships.

[5] Consider $L_1 = \sigma_f \alpha^*$ and $L_2 = \alpha^*$ where $\Sigma_f = \{\sigma_f\}$ and $\alpha \in \Sigma_o$. Then L_1 and L_2 are trivially diagnosable but $L_1 \cup L_2 = \sigma_f \alpha^* + \alpha^*$ is not diagnosable as can be seen by choosing $s = \sigma_f$, $t = \alpha^k$, and $\omega = \alpha^k$, for an arbitrarily large k, in Definition 2.

[6] Consider $L_1 = \overline{\{a\sigma_f d, a\sigma_u c\}}$ with $\Sigma_f = \{\sigma_f\}$ and $\Sigma_o = \{a, d, c\}$, and $L_2 = \overline{\{a\sigma_f, a\sigma_u\}}$; we have that L_2 is not diagnosable while L_1 is. Refer to Proposition 1.

[7] Henceforth, in this paper, normality and observability are taken to be with respect to L and P, and controllability is taken to be with respect to L and Σ_{uc} unless otherwise mentioned.

[8] For computationally efficiency, the solution procedure does not actually compute the supremal diagnosable and normal sublanguage; rather, it stops with computing its projection. The inverse language is retrieved in the subsequent iteration of Module A.

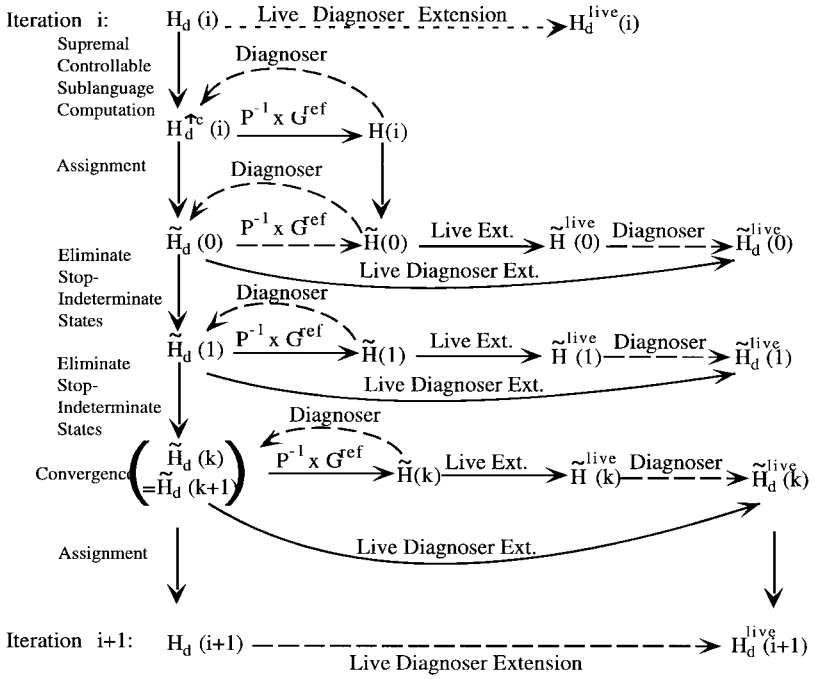

Fig. 3. Finite-state machines involved in the solution procedure.

Step 0-1: (To obtain G^{legal}) Given the regular language K, an FSM G^{legal} that generates this language can always be obtained (see, e.g., [4]). We also build a *refined* system model G^{ref} with the following properties:

- $L(G^{\text{ref}}) = L$;
- $G^{\text{legal}} \sqsubseteq G^{\text{ref}}$.

Given G^{legal}, the refined system model G^{ref} can be obtained from the system model G following the refinement procedure in [5].

Step 0-2: (To obtain G_d^{legal}) G_d^{legal} and G_d^{ref}, the diagnosers corresponding to G^{legal} and G^{ref}, respectively, can be built from G^{legal} and G^{ref}, respectively, following the usual construction procedure of the diagnoser. From Step 0-1 above, it follows that $G_d^{\text{legal}} \sqsubseteq G_d^{\text{ref}}$. Also note that $L(G_d^{\text{ref}}) = L(G_d) = P(L)$ and $L(G_d^{\text{legal}}) = P(K)$.

Step A-1: [To obtain $H_d^{\uparrow C}(i)$] Given two FSM's G_1 and G_2 such that $L(G_1) \subseteq L(G_2)$, several finite step procedures to obtain $G_1^{\uparrow C}$ (where $G_1^{\uparrow C}$ is the generator of $L(G_1)^{\uparrow C}$ with respect to $L(G_2)$ and the set of uncontrollable events in G_2) exist. We follow the procedure in [5]. This procedure requires that $G_1 \sqsubseteq G_2$. Lemma 1 in Appendix A establishes that $H_d(i) \sqsubseteq G_d^{\text{ref}}$ for all $i \geq 0$. Hence the procedure of [5] can be used to compute $H_d^{\uparrow C}(i)$ from G_d^{ref} and $H_d(i)$, for all $i \geq 0$.

Step A-2: [To obtain $H(i)$] Let $\delta_d^{\uparrow C}(i)$ denote the transition function of $H_d^{\uparrow C}(i)$. At each state q of $H_d^{\uparrow C}(i)$ add a new transition as follows: $\delta_d^{\uparrow C}(i)(q, \sigma) = q$, $\sigma \in \Sigma_{uo}$. In other words, add self-loops at every state of $H_d^{\uparrow C}(i)$ due to all unobservable events in Σ_{uo}. With slight abuse of notation we refer to this operation as $P^{-1}(H_d^{\uparrow C}(i))$. Then $H(i) = P^{-1}(H_d^{\uparrow C}(i)) \times G^{\text{ref}}$.

Step B-1: [To obtain $\tilde{H}^{\text{live}}(k)$ and $\tilde{H}_d^{\text{live}}(k)$] $\tilde{H}^{\text{live}}(k)$, the live extension of the FSM $\tilde{H}(k)$, can be obtained from $\tilde{H}(k)$ following the procedure described in Section III-B [cf., (9)]. While $\tilde{H}_d^{\text{live}}(k)$, the diagnoser corresponding to $\tilde{H}^{\text{live}}(k)$, can be obtained following the usual construction procedure of the diagnoser, it can be obtained as a simple extension of $\tilde{H}_d(k)$ as follows, thereby reducing computation. Since $\tilde{H}^{\text{live}}(k)$ is the live extension of $\tilde{H}(k)$, then $\tilde{H}_d^{\text{live}}(k)$ can be constructed from $\tilde{H}_d(k)$ following the diagnoser extension procedure described in Section III-B1 if it can be established that $\tilde{H}_d(k)$ is the diagnoser corresponding to $\tilde{H}(k)$. This result is proved in Lemma 4 in Appendix A. Hence the extension procedure of Section III-B1 can be used to obtain $\tilde{H}_d^{\text{live}}(k)$ from $\tilde{H}_d(k)$ once the "dead" states $Q^{\text{dead}}(\tilde{H}_d(k))$ are identified. Identification of $Q^{\text{dead}}(\tilde{H}_d(k))$ in general involves examining every state q of $\tilde{H}_d(k)$ and checking, from $\tilde{H}(k)$, if $D(q)$ (cf., (11) in Section III-B1) is nonempty. However, since we have at hand G_d^{ref}, the diagnoser of the live language $L \supseteq K$ (constructed in Step 0-2) the identification of $Q^{\text{dead}}(\tilde{H}_d(k))$ can be simplified by: 1) comparing the transitions defined at each state of $\tilde{H}_d(k)$ with those defined in G_d^{ref} and 2) for only those states $q \in \tilde{H}_d(k)$ such that there exists a transition out of q in G_d^{ref} which is not present in $\tilde{H}_d(k)$, checking if $D(q)$ is nonempty.

Step B-2: [To obtain $\tilde{H}_d(k+1)$] From the assumption on K (that every live sublanguage of K is diagnosable) and from Property P2) (cf., Section III-B2) every indeterminate cycle in $\tilde{H}_d^{\text{live}}(k)$, if any, is caused by a self-loop at some F_i-uncertain state of $\tilde{H}_d^{\text{live}}(k)$ due to the Stop event; therefore the indeterminate cycles in $\tilde{H}_d^{\text{live}}(k)$ and hence the stop-indeterminate states of the pair $(\tilde{H}_d(k), \tilde{H}_d^{\text{live}}(k))$ can be determined simply by identifying in $\tilde{H}_d^{\text{live}}(k)$ F_i-uncertain states with a self-loop due to the Stop event for any failure type F_i. Once the stop-indeterminate states are determined, $\tilde{H}_d(k+1)$ is obtained by eliminating these states from $(\tilde{H}_d(k)$ and obtaining the accessible part of the resulting machine.

Step B-3: [To obtain $\tilde{H}(k+1)$] $\tilde{H}(k+1)$ can be obtained from $\tilde{H}_d(k+1)$ following the same procedure as in Step A-2 for obtaining $H(i)$ from $H_d^{\uparrow C}(i)$, i.e., $\tilde{H}(k+1) = P^{-1}(\tilde{H}_d(k+1)) \times G^{\text{ref}}$.

Note: The procedure to obtain the refined system model G^{ref} (Step 0-1) is of polynomial complexity (cf., [5]). The procedure to obtain the diagnosers G_d^{legal} and G_d^{ref} (Step 0-2) is of exponential complexity (cf., [10]). Each step of the iterative part of the solution procedure (i.e., Steps A-1 through C-1) is of polynomial complexity. Thus, once the initialization part of the procedure is completed the rest of the procedure can be implemented with polynomial complexity.

D. Correctness of the Solution Procedure

We now prove that the iterative procedure presented in Section IV-B converges in a finite number of steps to the solution of the ADP.

Theorem 2: The iterative solution procedure of Section IV-B for solving the ADP converges in a finite number of iterations. $M(i)$ at convergence is the supremal controllable, observable, and diagnosable sublanguage of K and is a regular language. The supervisor S_P that achieves the closed-loop language $M(i)$ can be realized by $H_d(i)$, the diagnoser corresponding to the generator $H(i)$ of $M(i)$.

Proof of Theorem 2: From Steps 0-1 through Step A-2 of the solution procedure we have

$$\begin{aligned} M(0) &= P_L^{-1}[M_d^{\uparrow C}(0)] \\ &= P_L^{-1}[L(H_d(0))^{\uparrow C}] \\ &= P_L^{-1}[L(G_d^{\text{legal}})^{\uparrow C}] \\ &= P_L^{-1}[P(K)^{\uparrow C}] \\ &= K^{\uparrow CN} \end{aligned}$$

where the last equality follows from [1, Th. 4] and from the assumption that K is normal. Let

$$T(M(0)) = \{\text{Terminating traces in } M(0) \text{ that violate condition } D_1 \text{ of diagnosability}\}.$$

Note that since K is such that every live sublanguage of K is diagnosable, any trace in K that violates diagnosability is a terminating trace. For any trace t denote by $\text{Pre}_o(t)$ the longest prefix of t the last event of which is observable. Define $\text{Pre}_o(T(M(0))) = \{\text{Pre}_o(t) : t \in T(M(0))\}$. By Properties P1) and P2) of Section III-B2 every trace in $T(M(0))$ leads to a stop-indeterminate state in $H_d^{\uparrow C}(0) = \tilde{H}_d(0)$. The first iteration of Module B (i.e., Steps B-1 through B-4) of the solution procedure removes the stop-indeterminate states of the pair $(H_d^{\uparrow C}(0), H_d^{\text{live}}(0))$. Further iterations (if necessary) of Module B remove the Stop-indeterminate states of the resulting pairs $(\tilde{H}_d(k), \tilde{H}_d^{\text{live}}(k))$ that cause a violation of diagnosability and, on convergence, result in the machine $H_d(1)$. By construction, $P_L^{-1}(L(H_d(1))$ is the supremal normal and diagnosable sublanguage of $M(0)$ since the only traces removed are traces that violate diagnosability, and Lemma 5 of Appendix A proves that such removal preserves normality. The language $P_L^{-1}(L(H_d(1)))$ may not be controllable since $\Sigma_c \subseteq \Sigma_o$, and hence, the last observable events that are eliminated in the last iteration inside Module B may not be controllable. If $P_L^{-1}(L(H_d(1)))$ is not controllable the solution procedure proceeds with the iteration, with $P_L^{-1}(L(H_d(1)))$ in the place of K, and with $H_d(1)$ in the place of $H_d(0) = G_d^{\text{legal}}$.

From Lemma 2 in Appendix A, we have $H_d(i+1) \sqsubseteq H_d(i)$ $\forall i \geq 0$. Since $H_d(0)$ is an FSM, the solution procedure is guaranteed to converge in a finite number of steps. The language $M(i)$, obtained when the solution procedure converges, is regular because it is realized by the FSM $H(i)$. Furthermore, $M(i)$ is, by construction, the supremal controllable, normal, and diagnosable sublanguage of K. Under Assumption A3), controllability and normality together imply observability. Consequently, $M(i)$ is the supremal controllable, observable, and diagnosable sublanguage of K. The supervisor S_P that synthesizes the closed-loop language $M(i)$ is realized by $H_d(i)$ (= $H_d^{\uparrow C}(i)$, the diagnoser corresponding to the generator $H(i)$ of $M(i)$, at convergence). Furthermore, online failure diagnosis of the closed-loop system S_P/G can be performed using the FSM $H_d^{\text{live}}(i)$ which is the live diagnoser of $H(i)$ (cf., Lemma 6). □

Remark: Since $H_d^{\text{live}}(i)$ can be used to perform online failure diagnosis of the closed-loop system formed by system G and the supervisor $H_d(i)$, we see that the solution procedure of Section IV-B provides both a controller that ensures diagnosability of the closed-loop system and a diagnoser for online failure diagnosis.

E. On the Choice of K and the Class of Languages K_l

In this section, we introduce a class of languages $K_l \subseteq L$, $l \geq 0$ that can be used as initial conditions (i.e., as the language K) for the active diagnosis problem discussed in the preceding sections. First, we present the following notation.

Suppose that the states $q_i, i = 1, \cdots, n$ form an indeterminate cycle in the diagnoser G_d. We shall refer to this cycle as an *elementary* indeterminate cycle if $q_i \neq q_j$, $\forall i, j \in \{1, \cdots, n\}$. We then define an *interleaved* indeterminate cycle to be an indeterminate cycle composed of at least two elementary indeterminate cycles such that they share at least one common state (in a manner such that it is possible for the diagnoser to keep alternating between these two cycles). Fig. 4 provides examples of some interleaved cycles.

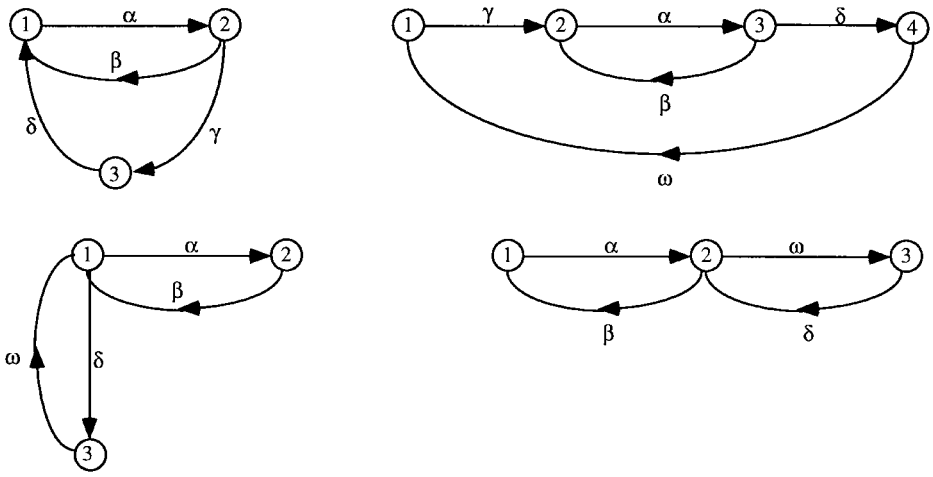

Fig. 4. Some examples of interleaved cycles.

Definition 4:

1) A trace $\omega \in L(G_d)$ is said to go through an elementary indeterminate cycle in G_d k times if $\omega = s(\sigma_1\sigma_2\cdots\sigma_n)^k t$ where $s,t \in \Sigma_o^*$; $\delta_d(q_0,s) = q_1$; $\delta_d(q_i,\sigma_i) = q_{(i+1)\bmod n}$, $i = 1,\cdots,n$; and $\{q_i\}_{i=1}^n$ forms an elementary indeterminate cycle in G_d.
2) A trace $\omega \in L(G_d)$ is said to go through an interleaved indeterminate cycle in G_d k times if it goes through each elementary cycle in the given interleaved cycle k times.

Henceforth, whenever we refer to a cycle in the diagnoser simply as an indeterminate cycle, we will mean either an elementary or an interleaved cycle.[9]

Definition 5: A trace $\omega \in L$ is said to go through an indeterminate cycle in its diagnoser G_d k times if $P(\omega) \in L(G_d)$ goes through an indeterminate cycle in G_d k times.

We are now ready to define the languages K_l.

Definition 6: Given a nondiagnosable and live language L, define

$$K_l = L - \{st \in L : s \text{ goes through an indeterminate cycle in } G_d \ l+1 \text{ times}\}.$$

Therefore, K_0 consists of all traces in L except those that complete any elementary indeterminate cycle in G_d, K_1 consists of those traces in L that complete any elementary indeterminate cycle in G_d at most once, and K_l consists of traces in L that complete any elementary indeterminate cycle in G_d at most l times. Note that K_l is defined with respect to the diagnoser G_d. Therefore, it depends on the system G and not only on the language L generated by G. In other words, given a language L and two different FSM's G and G' such that they both generate the language L, any K_l obtained from G may be different than that obtained from G'. However, it is not difficult to see that these two languages will differ only in the number of times the cycles corresponding to an indeterminate cycle in the diagnoser are included.

Example 4.1: For the system represented in Fig. 1 we have (with a slight abuse of notation in the usage of *)

$$K_0 = \Pr(\{\alpha\beta\gamma\sigma_{uo}, \alpha\sigma_{f1}\beta\gamma\tau\tau^*, \sigma_{f1}\alpha\beta\gamma, \sigma_{f1}\alpha\alpha\tau^*\})$$

and

$$K_i = \Pr(\{\alpha(\beta\gamma\sigma_{uo}\delta)^i\beta\gamma\sigma_{uo}, \alpha(\beta\gamma\sigma_{uo}\delta)^i\sigma_{f1}\beta\gamma\tau\tau^*, \sigma_{f1}\alpha(\beta\gamma\delta)^i\beta\gamma, \sigma_{f1}\alpha(\beta\gamma\delta)^i\alpha\tau^*\})$$

where $\Pr(L) = \bar{L}$. Note that this is an example of K_l in the case of elementary indeterminate cycles.

1) Properties of the Languages K_l: We now present some properties of the languages K_l defined above.

Property 1: The languages K_l, $l \geq 0$, are regular.

Proof: The languages K_l can be realized by FSM's. In Appendix B we present a procedure to obtain a finite-state generator of any K_l, $l \geq 0$, given the system G and the set of indeterminate cycles in the diagnoser G_d for the case where G_d does not contain any interleaved indeterminate cycles. In the case where G_d does contain interleaved cycles we present appropriate arguments to show that the languages K_l can be realized by FSM's. Hence it follows that K_l, $l \geq 0$ are regular. □

Property 2: The languages K_l, $l \geq 0$ are normal.

Proof: From Steps 1 and 2 of the procedure in Appendix B to calculate a finite state generator of K_l we have that $K_l = L(G^{\text{legal}}) = P^{-1}[L(G'_d)] \cap L$ where $L(G'_d) = P(K_l) \subseteq P(L)$. From Lemma 7 in Appendix A it follows that K_l is normal. □

Property 3: Given a nondiagnosable and live language L, and K_l, $l \geq 0$

$$(M \subseteq K_l) \wedge (M \text{ live}) \Rightarrow M \text{ diagnosable}.$$

Proof: Since K_l is obtained by removing from L all traces that go through an indeterminate cycle in G_d more than l times, it is obvious that K_l and hence $M \subseteq K_l$ do not contain any nonterminating trace of L that violate the definition of diagnosability. (Recall Theorem 1 and the

[9] In the case of interleaved cycles, K_l could alternately be defined by modifying Definition 4 as follows: a trace ω is said to go through an interleaved indeterminate cycle in G_d k times if the total number of times that it goes through *any combination* of elementary cycles in the given interleaved cycle is equal to k. It is not difficult to see that the resulting definition of K_l is complementary to the one that results from Definition 4. All of the discussions in this section that follow hold true for the modified definition as well.

explanation preceding it.) This implies that if M is live, then it is diagnosable. □

In other words, Property 3 states that if $M \subseteq K_l$ is nondiagnosable, then M cannot be live; furthermore, any trace in M that violates the definition of diagnosability has to be a terminating trace. Note that M does not have to be regular.

Property 4: Given a nondiagnosable language L and $K_l,\ l \geq 0$

$$K_0, K_1, \cdots K_l, \cdots \text{ diagnosable} \not\Rightarrow \bigcup_{l=1}^{\infty} K_l \text{ diagnosable}.$$

Proof: The proof is obvious by noting that $\bigcup_{l=1}^{\infty} K_l = L$. □

This result simply states that even if every sublanguage of L obtained by including only those traces in L that visit the states of an indeterminate cycle in the diagnoser G_d a finite and bounded number of times (if at all), is diagnosable, L itself is nondiagnosable since it contains traces that can visit an indeterminate cycle in G_d an arbitrarily large number of times.

We now present some additional properties of the languages K_l that are used primarily to motivate the choice of a particular K_l as initial condition for the ADP. The proofs of Properties 5 and 7 are not presented here for the sake of brevity. The interested reader can find these proofs in Appendix B.

Property 5: Given a nondiagnosable language L and $K_l,\ l \geq 0$

$$K_0 \text{ diagnosable} \Leftrightarrow K_l \text{ diagnosable}.$$

This property means that if the language obtained by cutting an indeterminate cycle in G_d before it gets completed once is not diagnosable, then so is the language obtained by including the loop l times and then cutting it before it gets completed for the $l+1$th time, and vice versa. Therefore, as far as diagnosability is concerned, there is nothing to be gained by simply extending a nondiagnosable language by extra traversals of an indeterminate cycle.

Definition 7: Given a diagnosable language L and $s \in L$, define the equation shown at the bottom of the page. In other words, the delay(s, L) denotes the maximum number of event occurrences possible in L after the trace s before which the occurrence of the failure event s_f cannot be diagnosed.

Property 6: Given a nondiagnosable language L, and $K_l,\ l \geq 0$ diagnosable, then $\forall p \in K_l$, delay$(p, K_l) \leq$ delay(p, K_{l+1}).

Proof: The proof is straightforward from Definition 7 of delay and from the fact that $K_l \subseteq K_{l+1}$. Note that the above inequality becomes an equality if the maximum delay in detecting a failure event in K_{l+1} occurs along a trace s such that s is also contained in K_l. □

Property 7: Given a nondiagnosable language L, and K_l, $l \geq 0$ diagnosable, there exist $m > 0$ and $p \in K_0$ such that for all $n \geq 1$

$$\text{delay}(p, K_0) < \text{ delay}(p, K_m) < \text{delay}(p, K_{m+n}).$$

Property 7 says that when the languages $K_l,\ l \geq 0$ are diagnosable, then for large l, the maximum delays in detecting failures occurs along traces that go through indeterminate cycles in the diagnoser; further, each additional traversal of the cycle results in additional delay in detecting the failure.

We now show how the languages K_l form good initial conditions to the ADP posed in Section IV-A. Recall that given a nondiagnosable system, the goal of the ADP is to restrict the language generated by the system to a diagnosable sublanguage. Ideally, we would like this language to be as large as possible since we do not want the control policy to restrict the behavior of the system more than necessary in order to make it diagnosable. Further, from the discussion in Section II-C (see Theorem 1 and the discussion preceding it), in order to obtain a diagnosable and normal sublanguage of any given nondiagnosable language, we need to eliminate the F_i-indeterminate cycles in the corresponding diagnoser, for all failure types F_i. The languages K_l, by definition, are obtained by eliminating the traces that go through indeterminate cycles in the diagnoser, and these languages differ from L *only* in those traces associated with indeterminate cycles. Therefore, they are good candidates for the legal language K if it can be shown that they satisfy the assumptions made on K in the formulation of the ADP, namely, that every live sublanguage of K_l is diagnosable and that K_l is normal. While the languages K_l may themselves not be diagnosable (since eliminating from L those traces that go through indeterminate cycles in the diagnoser may result in nonlive languages, i.e., K_l may contain terminating traces that violate diagnosability), we have from Property 3 that any live sublanguge of K_l is diagnosable. Further, from Property 2 we have that K_l is normal. Therefore, the languages K_l satisfy the required assumptions.

Thus, we see that each of the languages K_l of Definition 6 is a candidate for the initial condition K. The question then is: which of these K_l does one choose? The following factors motivate the choice of K_l for a particular problem.

- From Property 5, we have that if K_i is diagnosable, then K_j is diagnosable, and vice-versa for all i, j. Hence diagnosability considerations do not have to be taken into account in the choice of K_l.
- Consider the two languages K_m and K_{m+1} and let L_m (L_{m+1}) denote the closed-loop behavior resulting when K_m (K_{m+1}) is chosen as the desired K_l in the ADP. Then it is obvious that $L_m \subseteq L_{m+1}$ since $K_m \subseteq K_{m+1}$, i.e., we get a larger closed-loop behavior if we choose K_{m+1}.

$$\text{delay}(s, L) = \begin{cases} \text{Max}\{||t|| : (t \in L/s) \wedge (\text{st satisfies } D_1 \wedge D_2) \wedge \\ \quad (\forall u \in \overline{st}, u \text{ does not satisfy } D_1 \vee D_2)\}, & \text{if } s_f \in \Sigma_{fi} \text{ for some } i \in \Pi_f \\ \text{undefined}, & \text{otherwise} \end{cases}$$

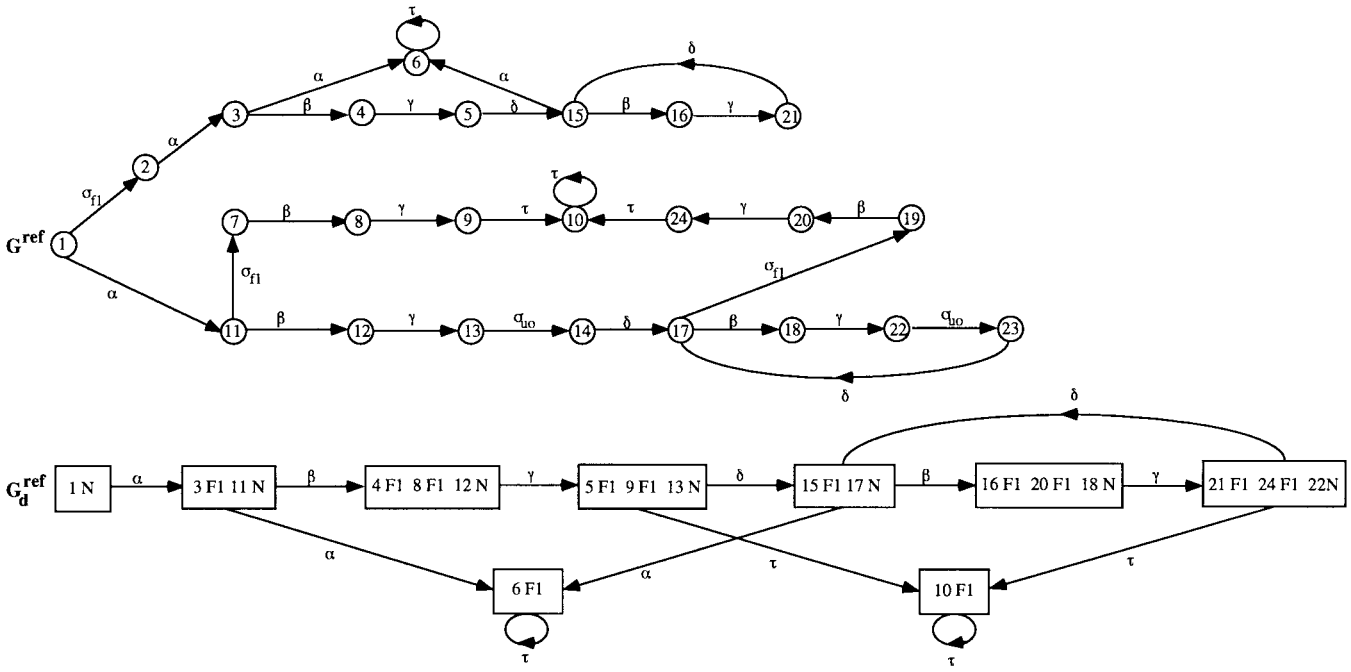

Fig. 5. The refined system model G^{ref} and the diagnoser G_d^{ref}.

- From Properties 6 and 7, the detection delay of all failures in K_{m+1} is greater than or equal to the corresponding delay in K_m, and for a large enough m, the detection delay in K_{m+1} is strictly greater than that in K_m.

Thus, the choice of K_l is dictated primarily by two considerations: closed-loop behavior and failure detection delay. This choice reflects the designer's tradeoff between minimal detection delay and maximal closed-loop behavior. Once the appropriate K_l is chosen based on the above design considerations, we look for the largest sublanguage L^{act} of K_l that is diagnosable and that can be achieved by control. The solution to the ADP is then given by that supervisor S_P that synthesizes this closed-loop language L^{act}.

F. Illustrative Example

We now present a simple example to illustrate the steps of the solution procedure for the ADP. In Section V we present a physical system and illustrate the application of the theory developed in this paper to the design of a diagnostic controller for this system.

Example 4.2: Consider the system represented in Fig. 1. As before let $\Sigma_f = \Sigma_{f1} = \{\sigma_{f1}\}$ and let $\Sigma_{uo} = \Sigma_f \cup \{\sigma_{uo}\}$. Also let $\Sigma_{uc} = \Sigma_f \cup \{\delta\}$. As seen in Example 2.1 this system has one indeterminate cycle formed by the three states $\{(3,\{F1\}),(11,\{N\})\}$, $\{(4,\{F1\}),\ (8,\{F1\}),(12,\{N\})\}$, and $\{(5,\{F1\}),(9,\{F1\}),(13,\{N\})\}$ with the corresponding event sequence $\beta\gamma\delta$, and it is not diagnosable.

We solve the ADP for this system with $K = K_1$. Figs. 5–7 illustrate the various steps of the solution procedure for this problem.

Initialization: Fig. 5 depicts the refined system G^{ref} and the corresponding diagnoser G_d^{ref}. Note that the indeterminate cycle has been expanded out once in G_d^{ref}. G^{legal} is the same as G^{ref} in Fig. 5 except that the event δ, which is the final event that completes the indeterminate cycle, is not defined at states 21 and 23 of G^{ref}. Likewise, G_d^{legal} is the same as G_d^{ref} except that δ is not defined at the state $\{(21,\{F1\}),(24,\{F1\}),(22,\{N\})\}$ of G_d^{ref}. By definition $H_d(0) = G_d^{\mathrm{legal}}$.

Iteration 1: Since the event δ is uncontrollable, $H_d^{\uparrow C}(0)$ excludes from $H_d(0)$ the state $\{(21,\{F1\}),\ (24,\{F1\}),\ (22,\{N\})\}$ and the transitions associated with this state (see Fig. 6). The corresponding "inverse" machine is $H(0)$. We now set $\tilde{H}_d(0) = H_d^{\uparrow C}(0)$ and $\tilde{H}(0) = H(0)$. $\tilde{H}(0)$, its live extension $\tilde{H}^{\mathrm{live}}(0)$, and the corresponding live diagnoser $\tilde{H}_d^{\mathrm{live}}(0)$, are as depicted in Fig. 6. Inspection of $\tilde{H}_d^{\mathrm{live}}(0)$ reveals an F_1-indeterminate cycle formed by the F_1-uncertain state $\{(16,\{F1\}),(20,\{F1\}),(18,\{N\})\}$ and the Stop event. Hence the state $\{(16,\{F1\}),(20,\{F1\}),(18,\{N\})\}$ is a stop-indeterminate state of the pair $(\tilde{H}_d(0),\ \tilde{H}_d^{\mathrm{live}}(0))$. We obtain $\tilde{H}_d(1)$ by eliminating this stop-indeterminate state from $\tilde{H}_d(0)$ (see Fig. 7). Since $\tilde{H}_d(1) \neq \tilde{H}_d(0)$ we compute $\tilde{H}(1)$, the inverse machine of $\tilde{H}_d(1)$, and continue the iteration of Module B. Fig. 7 depicts $\tilde{H}(1)$, its live extension $\tilde{H}^{\mathrm{live}}(1)$, and the corresponding live diagnoser $\tilde{H}_d^{\mathrm{live}}(1)$. Inspection of $\tilde{H}_d(1)$ and $\tilde{H}_d^{\mathrm{live}}(1)$ (see Fig. 7) shows that there are no stop-indeterminate states and hence $\tilde{H}_d(2) = \tilde{H}_d(1)$. This completes the iteration inside Module B, and we set $H_d(1) = \tilde{H}_d(1)$ and $H_d^{\mathrm{live}}(1) = \tilde{H}_d^{\mathrm{live}}(1)$. Since $H_d(1) \neq H_d(0)$ we continue to the second iteration of the solution procedure.

Iteration 2: Since the event β leading into state $\{(16,\{F1\}),(20,\{F1\}),(18,\{N\})\}$ [that was eliminated in iteration 1 to obtain $\tilde{H}_d(1) = H_d(1)$] is controllable, $H_d^{\uparrow C}(1)$ is the same as $H_d(1) = \tilde{H}_d(1)$ (of iteration 1) and $H(1) = \tilde{H}(1)$ (of Iteration 1). Moving onto Module B of the solution procedure, as before, we set $\tilde{H}_d(0) = H_d^{\uparrow C}(1)$ and $\tilde{H}(0) = H(1)$.

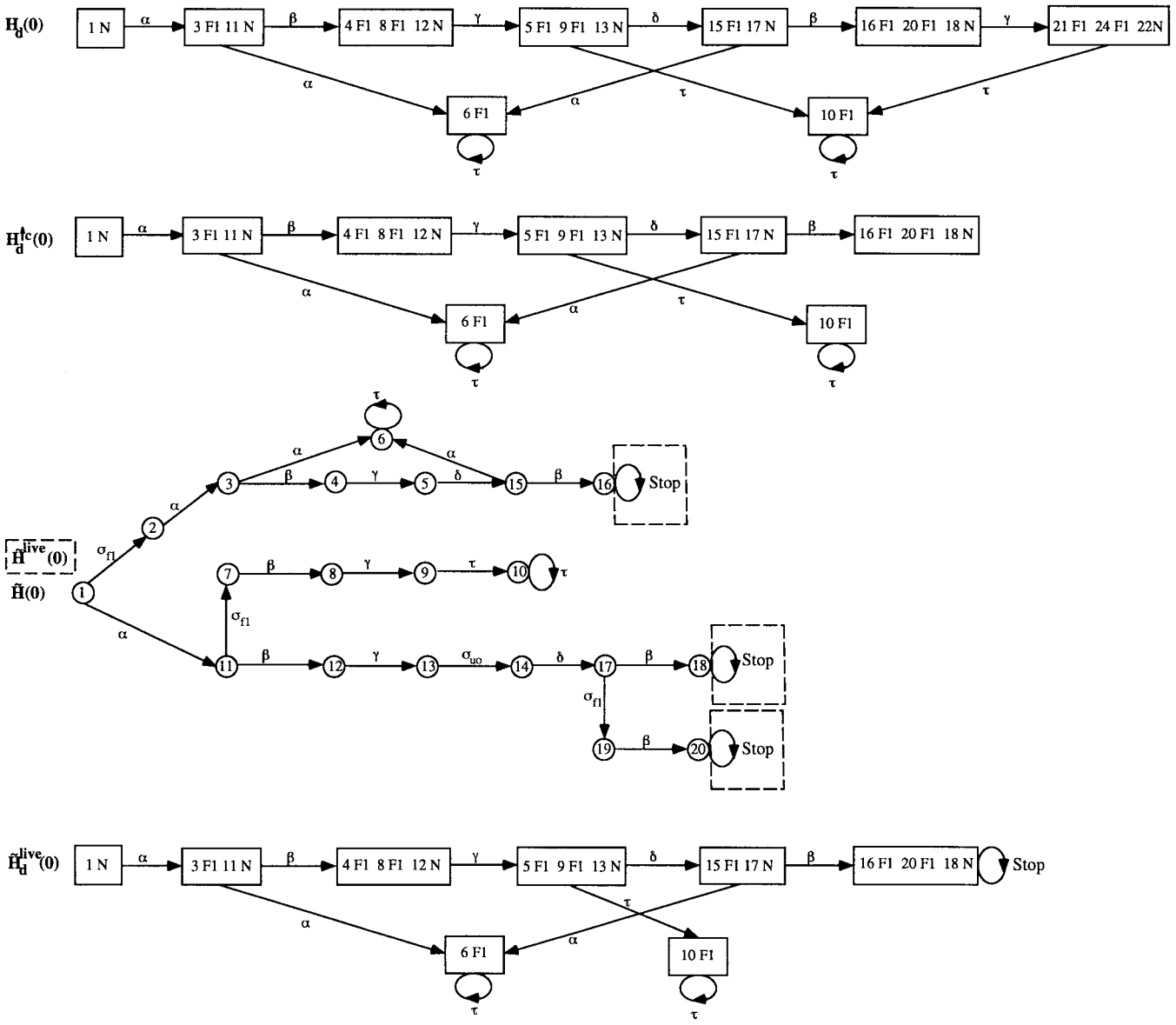

Fig. 6. First part of solution procedure.

Since $\tilde{H}_d(0)$ (of Iteration 2) $= H_d^{\uparrow C}(1) = H_d(1) = \tilde{H}_d(1)$ of Iteration 1, $\tilde{H}_d^{\text{live}}(0)$ of Iteration 2 is equal to $\tilde{H}_d^{\text{live}}(1)$ of Iteration 1. It follows that the pair $(\tilde{H}_d(0), \tilde{H}_d^{\text{live}}(0))$ contains no stop-indeterminate states; hence $\tilde{H}_d(1) = \tilde{H}_d(0)$ and the procedure exits out of Module B with $H_d(2) = \tilde{H}_d(0)$ and $H_d^{\text{live}}(2) = \tilde{H}_d^{\text{live}}(0)$. Since $H_d(2) = \tilde{H}_d(0) = H_d(1)$, the solution procedure terminates at the second iteration.

The solution to the ADP is $L^{\text{act}} = M(1) = L(H(1)$ and the supervisor S_P that synthesizes $M(1)$ is realized by $H_d(1)$, the diagnoser of $H(1)$. Further, $H_d^{\text{live}}(1)$ can be used to perform online diagnosis of the closed-loop formed by the system G and the controller $H_d(1)$.

V. Application: Design of a Diagnostic Controller for a Pump–Valve System

In this section we demonstrate the application of the theory developed in this paper to the design of a diagnostic controller for a pump–valve system.

Consider a simple system consisting of a pump and a valve. Suppose that the valve has two failure modes, a stuck-open failure mode and a stuck-closed failure mode. We assume that the valve can get stuck open only from its open state and it can get stuck closed only from its closed state. Let the system be equipped with just one sensor, a flow sensor that can read one of two possible values: F, indicating that there is a flow and NF, indicating that there is no flow. Suppose that we need to design a controller for this system that achieves the following objectives.

1) When there is a load on the system, the controller must respond by starting the pump and opening the valve.
2) When there is no load on the system, the controller must respond by stopping the pump and closing the valve.
3) The sequence of start-pump, stop-pump, close-valve, and open-valve commands has to be chosen in a manner such that the resulting system is diagnosable.

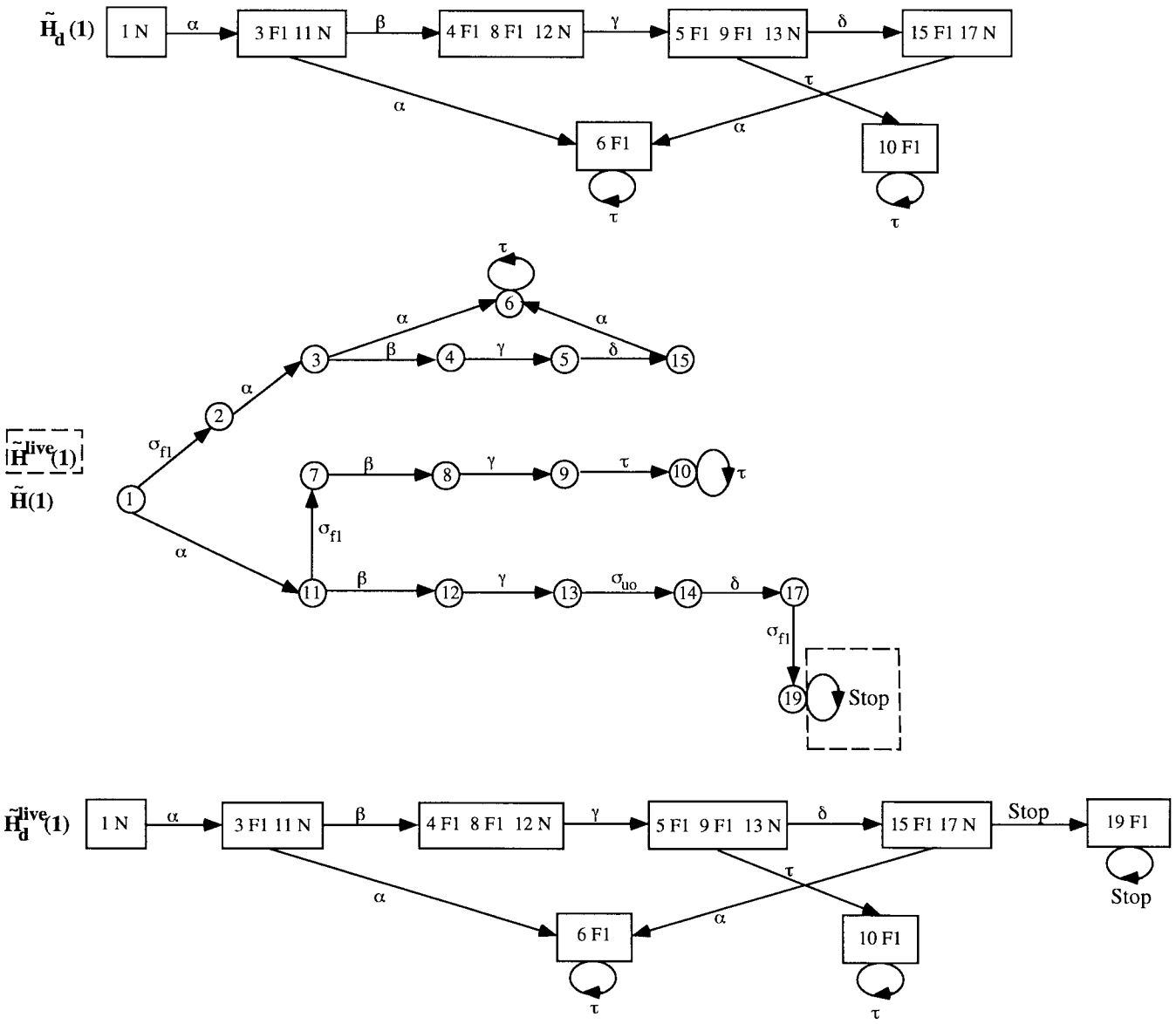

Fig. 7. Second part of solution procedure.

Fig. 8 depicts the FSM models for the pump and the valve. Also shown in Fig. 8 is a controller model. This controller captures requirements 1) and 2) above and is straightforward to build. The problem now is to design a second controller that ensures diagnosability of the closed-loop system while still meeting objectives 1) and 2). We now demonstrate how this problem can be solved in the framework of the ADP.

The first step is to obtain the system model G. In this example G is chosen to be the closed-loop system formed by the pump, the valve, and the controller of Fig. 8. A systematic methodology to obtain the global system model starting from the individual component models (including the controller model) and from the sensor map (listed in Table I)[10] can be found in [11]. However, it is necessary to modify the modeling methodology of [11] to account for the controllability/uncontrollability of events, as explained below. The modeling formalism presented in [11] translates all sensor information into the event set. As a result, the global system model consists of "composite" events of the form ⟨COMMAND, RESULTANT SENSOR READINGS⟩, as for example, the event ⟨START_PUMP, F⟩. Note that while

TABLE I
THE GLOBAL SENSOR MAP FOR THE PUMP–VALVE SYSTEM

h(VC, POFF, •)	=	NF
h(VO, POFF, •)	=	NF
h(SC, POFF, •)	=	NF
h(SO, POFF, •)	=	NF
h(VC, PON, •)	=	NF
h(VO, PON, •)	=	F
h(SC, PON, •)	=	NF
h(SO, PON, •)	=	F

[10]The •'s in Table I stand for the state of the controller and are used to indicate that the sensor map is independent of the controller state.

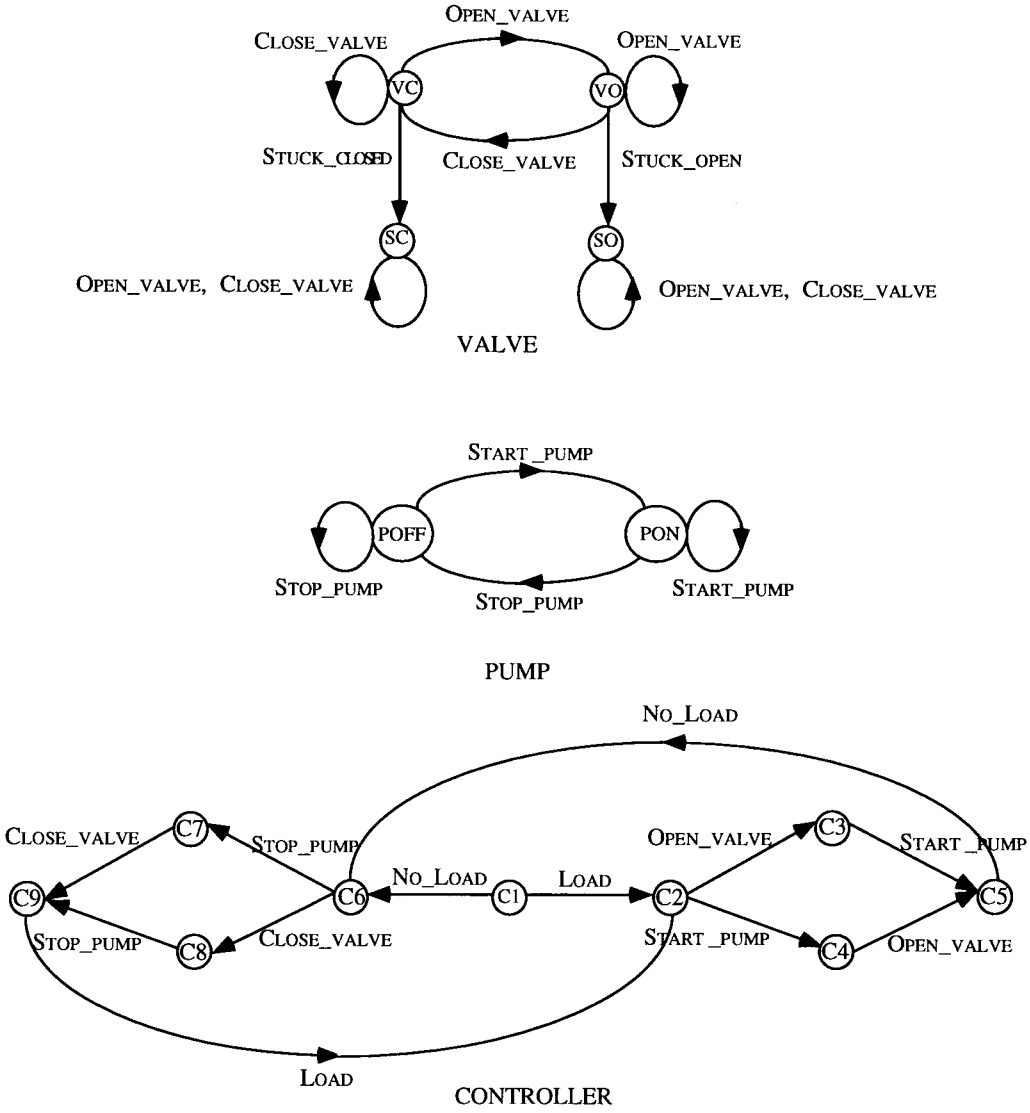

Fig. 8. Component models for the pump–valve system.

the command event is a controllable event, the event corresponding to the resultant sensor readings is uncontrollable. Hence, where it is necessary to partition the event set of the system into controllable and uncontrollable events, as in the active diagnosis problem studied here, use of the composite events poses a difficulty. Two approaches could be followed to overcome this difficulty. The first approach is to break every composite event in the global system model into two events: 1) the command and 2) the resultant sensor readings; the command event is then treated as a controllable event while the event corresponding to the sensor readings is considered to be uncontrollable. Note that this leads to the introduction of new states in the system model. The second approach is based on the notion of control patterns [3], [12]. Simply speaking, the use of control patterns implies that the set of controllable events that are to be enabled or disabled at any point of time cannot be arbitrarily chosen but are constrained to be within prespecified subsets, i.e., certain events may only be enabled or disabled together as a group. In our modeling framework, this amounts to classifying the composite events as controllable events and requiring that all events of the form ⟨COMMAND A, Y1⟩, ⟨COMMAND A, Y2⟩, $\cdots$, ⟨COMMAND A, YN⟩ should be enabled/disabled as a group.

While the procedure of [5] for computing the supremal controllable sublanguage of a given language does not handle control patterns, it can be modified to do so in a straightforward manner. Thus, either of the two approaches above can be used. However, for ease of representation, the event labels in the figures and tables that follow are left as composite events. A complete listing of the transition table for the pump–valve system can be found in [9, Appendix C]. If the first approach of breaking the composite events is adopted, then the controllable events in this system are the command events, OPEN_VALVE, CLOSE_VALVE, STOP_PUMP, and START_PUMP, while the uncontrollable events are the failure events, STUCK_CLOSED and STUCK_OPEN, and the events corresponding to the presence and the absence of a load on the system, namely, LOAD, and NO_LOAD. On the other hand, if

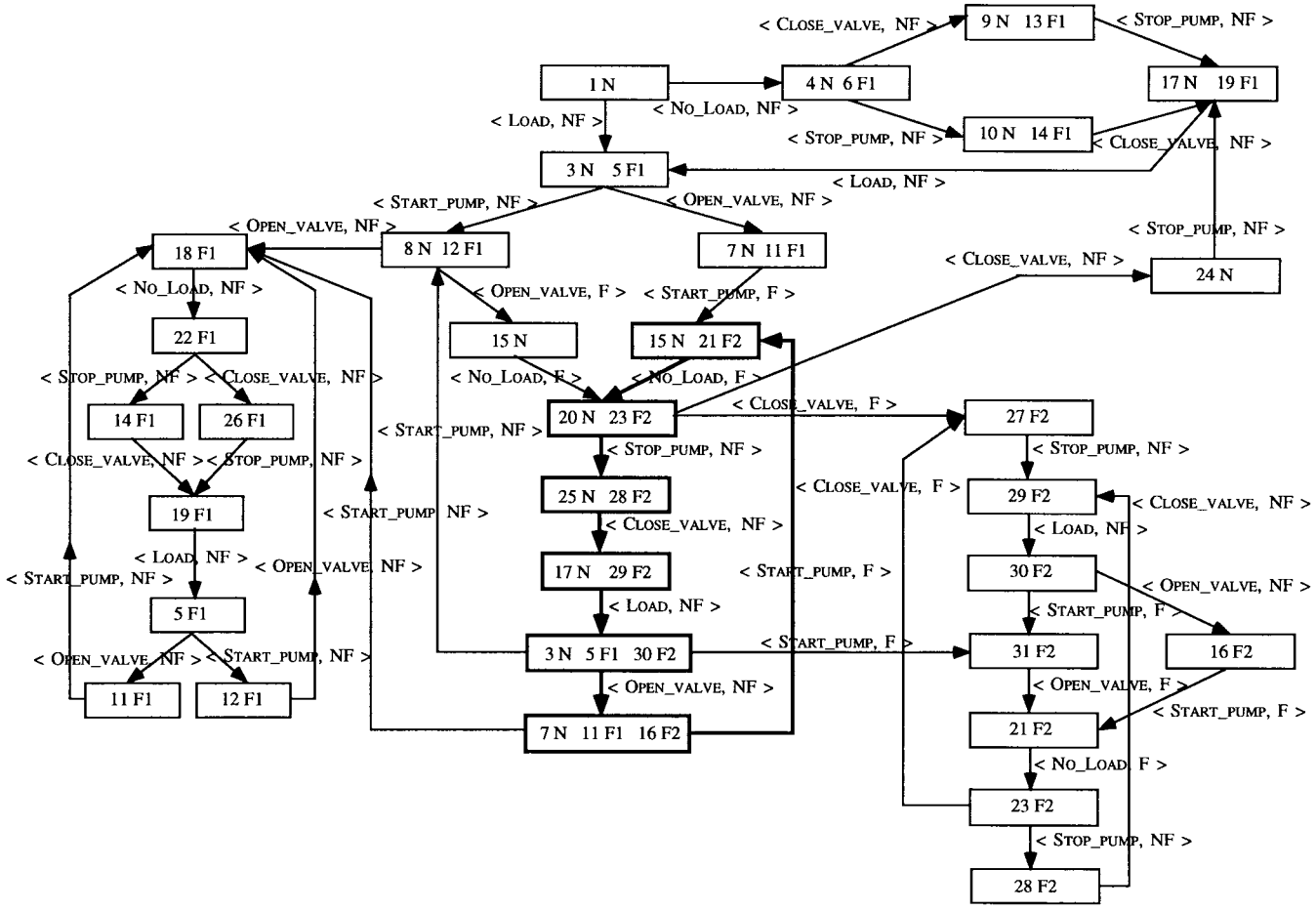

Fig. 9. The diagnoser for the pump–valve system.

we use control patterns, then the controllable subsets involve events of the form ⟨OPEN_VALVE, X⟩, ⟨CLOSE_VALVE, X⟩, ⟨STOP_PUMP, X⟩, and ⟨START_PUMP, X⟩, where X can take either of the two values, F or NF. Finally, the set of failure events is partitioned into two sets $\Sigma_{f1} = \{\text{STUCK\_CLOSED}\}$ and $\Sigma_{f2} = \{\text{STUCK\_OPEN}\}$.

The diagnoser G_d for this system (with the composite events) consists of 32 states and is shown in Fig. 9. (It is not difficult to see that the diagnoser for this system with the composite events broken up into two events will consist of 64 states.) A careful examination of the diagnoser G_d and the system G (cf., [9, Appendix C]) reveals that the cycle formed by the highlighted states in Fig. 9 is an F_2-indeterminate cycle; as long as the diagnoser remains in this cycle, one cannot detect stuck-open failures of the valve.

We solve the ADP for this system with $l = 0$. A partial realization (up to the point where failures are diagnosed) of the diagnostic controller is depicted in Fig. 10. (For the sake of brevity, we have presented only the final solution here, omitting the various steps of the solution procedure; it can be verified that the procedure converges to the solution of the ADP in just one iteration with again, just one iteration of Module B.) Note that this controller satisfies all three design objectives: it guarantees that the system responds appropriately to the presence or absence of a load *and* it results in a diagnosable system.

Finally, it is interesting to note that this design approach results in a much less restrictive system behavior than what could be obtained by a fixed control protocol, which also achieves the same design objectives. Consider for instance the control sequence, ⟨START_PUMP⟩, ⟨OPEN_VALVE⟩, ⟨CLOSE_VALVE⟩, ⟨STOP_PUMP⟩. This control sequence satisfies both design objectives. Further, it is not difficult to see from the diagnoser of Fig. 9 that this would result in a diagnosable system. However, it is evident that the controller of Fig. 10 obtained as a solution of the ADP achieves a much larger closed-loop behavior. Consider next the control sequence ⟨OPEN_VALVE⟩, ⟨START_PUMP⟩, ⟨STOP_PUMP⟩ ⟨CLOSE_VALVE⟩. While this protocol also achieves the desired control objectives, it does not result in a diagnosable system, as can be verified by tracing this control sequence through the diagnoser of Fig. 9. However, the diagnostic controller for this system allows the above control sequence to occur, provided that this sequence is followed by the sequence ⟨START_PUMP⟩, ⟨OPEN_VALVE⟩ which results in a diagnosable system. (Again, refer to Fig. 9.) Thus, starting from a controller that allows maximum flexibility of the closed-loop system while achieving the desired control objectives, and then restricting its behavior to yield a diagnosable system, results in a much larger closed-loop behavior than that obtainable by a fixed control protocol.

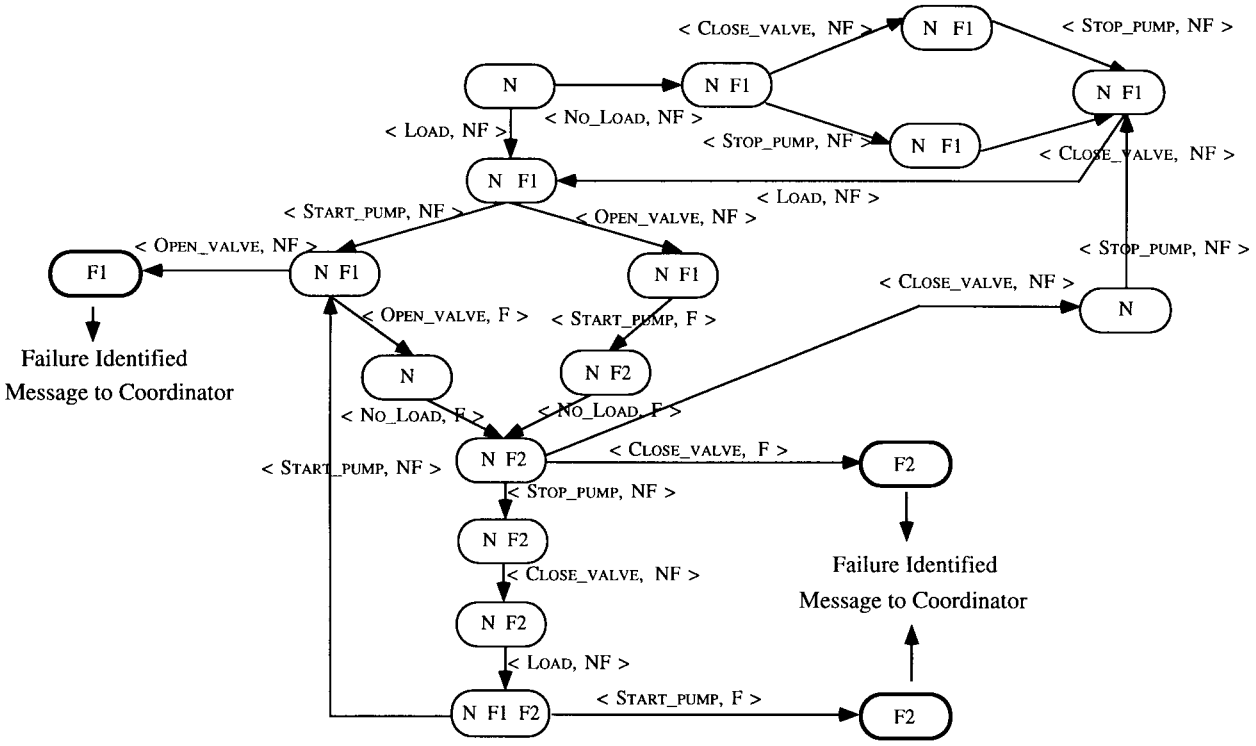

Fig. 10. Part of the diagnostic controller for the pump–valve system.

VI. Conclusion

Research in the area of failure diagnosis has so far focussed primarily on answering the following question: "given a system with several possible failure modes, how does one detect and diagnose these failures?" Several approaches have been proposed for the design and implementation of diagnostic modules for such systems. However, not much research effort has been directed at *building* systems that are *diagnosable*. In other words, there have been few attempts at answering the question, "given a system with multiple failure modes, and given a set of diagnostic requirements, how do we ensure that a system satisfies these requirements?" One way to ensure diagnosability is to equip the system with an appropriate set of sensors; the challenge then is to determine the optimal, feasible set of sensors that will meet the requirements. An alternate approach, where applicable, is to restrict the behavior of the system, by control, in a manner such that it results in a diagnosable system. In other words, the system controller is to be designed in such a way that it not only satisfies other specified control objectives, but it also results in a diagnosable system. In this paper we have investigated the above problem of *integrated control and diagnostics* in the framework of DES's. In particular, we have presented a procedure for the design of diagnostic controllers for DES's, based on the theory of failure diagnosis for DES developed in [10] and on existing results in supervisory control under partial observations.

The solution methodology presented here for the design of diagnostic controllers can also be used, with some modifications, to solve the problem of test/probe vector determination for offline diagnosis. Loosely speaking, the set of all probe sequences that achieve diagnosis of system failures may be determined, in the framework of the active diagnosis problem, by starting from a completely flexible system that allows for all feasible probe sequences and eliminating those that do not achieve diagnosis with a finite delay. This amounts to eliminating those traces that lead to indeterminate cycles in the corresponding diagnoser. Finally, one may further restrict the set of feasible test vectors by taking into consideration factors such as detection delay, cost of probing, etc.

Appendix A
Proofs of Technical Results Used in the Solution of the ADP

Lemma 1: $H_d(i) \sqsubseteq G_d^{\text{ref}}$ for all $i \geq 0$.

Proof: The above submachine requirement is satisfied for $H_d(0)(=G_d^{\text{legal}})$ since $G_d^{\text{legal}} \sqsubseteq G_d^{\text{ref}}$ (cf., Step 0-2 in Section IV-C); hence we can follow the procedure of [5] to obtain $H_d^{\uparrow C}(0)$ from G_d^{ref} and $H_d(0)$. Next, as can be seen from [5], $H_d^{\uparrow C}(0)$ resulting from the procedure is a submachine of $H_d(0)$. From Steps B-0 and B-2 of the solution procedure we have $H_d^{\uparrow C}(0) = \tilde{H}_d(0)$; $\tilde{H}_d(k) \sqsubseteq \tilde{H}_d(0)\ \forall k \geq 0$. Since $\tilde{H}_d(0)$ is an FSM, it follows that the iteration of Steps B-0 through B-3 will converge in a finite number of steps, and hence from Step B-3 of the solution procedure $H_d(1) = \tilde{H}_d(k)$, for some finite $k \geq 0$. It follows that $H_d(1) \sqsubseteq H_d(0)$ which then implies that $H_d(1) \sqsubseteq G_d^{\text{ref}}$. Following the same arguments as above we see that $H_d(i) \sqsubseteq G_d^{\text{ref}}$, for all $i \geq 0$. □

Lemma 2: $H_d(i+1) \sqsubseteq H_d(i)$, $\forall i \geq 0$.

Proof: The proof follows from the Proof of Lemma 1. □

Lemma 3: $H_d^{\uparrow C}(i)$ is the diagnoser corresponding to $H(i)$ for all $i \geq 0$.

Proof: From the proof of Lemma 1 and from Step A-1 of the solution procedure we note that

$$H_d^{\uparrow C}(i) \sqsubseteq G_d^{\text{ref}} \quad \text{and} \quad L\big(H_d^{\uparrow C}(i)\big) = M_d^{\uparrow C}(i). \tag{15}$$

Let $D(M(i))$ denote the diagnoser corresponding to the language $M(i)$ represented by the FSM $H(i)$. Then

$$L(D(M(i)) = P(M(i)) = P\big(P_L^{-1}\big[M_d^{\uparrow C}(i)\big]\big) = M_d^{\uparrow C}(i) \tag{16}$$

where the second equality follows from Step A-2 of the solution procedure. Since $G^{\text{ref}} = P^{-1}(G_d^{\text{ref}}) \times G^{\text{ref}}$, $H(i) = P^{-1}(H_d^{\uparrow C}(i)) \times G^{\text{ref}}$, and since $H_d^{\uparrow C}(i)$ is a submachine of G_d^{ref}, then $H(i)$, the generator of $M(i)$, is a submachine of G^{ref}. This implies that the diagnoser corresponding to $H(i)$ is a submachine of the diagnoser corresponding to G^{ref}, i.e.,

$$D(M(i)) \sqsubseteq G_d^{\text{ref}}. \tag{17}$$

From (15)–(17) we see that both FSM's $D(M(i))$ and $H_d^{\uparrow C}(i)$ are submachines of the same machine G_d^{ref}, and they generate the same language $M_d^{\uparrow C}(i)$. Since all of the above FSM's are deterministic [4], it follows that $D(M(i)) = H_d^{\uparrow C}(i)$. □

Lemma 4: For each iteration $k \geq 0$ of Module B of the solution procedure, $\tilde{H}_d(k)$ is the diagnoser corresponding to $\tilde{H}(k)$.

Proof: First, we have that $\tilde{H}_d(0)$ is the diagnoser corresponding to $\tilde{H}(0)$; this is because $\tilde{H}(0) = H(i)$ and $\tilde{H}_d(0) = H_d^{\uparrow C}(i)$ from Step B-0 of the solution procedure, and $H_d^{\uparrow C}(i)$ is the diagnoser corresponding $H(i)$ from Lemma 3 above. Next, $L(\tilde{H}_d(k)) = P(\tilde{M}(k)) = P(L(\tilde{H}(k)))$ from Step B-3. Finally, $\tilde{H}_d(k) \sqsubseteq \tilde{H}_d(0)$ from Step B-3 and hence it has the "structure" of a diagnoser. It then follows that $\tilde{H}_d(k)$ is the diagnoser corresponding to $\tilde{H}(k)$. □

Lemma 5: Consider $L = \bar{L} \subseteq \Sigma^*$ and $M = \bar{M} \subseteq L$ such that M is normal (with respect to P and L). Then $H = M - P_L^{-1}[P(\text{Pre}_o(T(M)))]\Sigma^*$ is normal with respect to P and L.

Proof: Let $s \in H$ and $s' \in L$ such that $P(s) = P(s')$. Since M is normal and $s \in M$, it follows that $s' \in M$. Furthermore, we claim that since $s \notin P_L^{-1}[P(\text{Pre}_o(T(M)))]\Sigma^*$, then $s' \notin P_L^{-1}[P(\text{Pre}_o(T(M)))]\Sigma^*$. To see this, observe that $\forall t \leq s$, $P(t) \notin P(\text{Pre}_o(T(M)))$. If $s' \in P_L^{-1}[P(\text{Pre}_o(T(M)))]\Sigma^*$, then $\exists t' \leq s'$ such that $P(t') \in P(\text{Pre}_o(T(M)))$. But $P(s) = P(s')$ implies that $\exists t'' \leq s$ such that $P(t'') = P(t')$, which in turn implies that $P(t'') \in P(\text{Pre}_o(T(M)))$. This leads to a contradiction. Hence $s' \notin P_L^{-1}[P(\text{Pre}_o(T(M)))]\Sigma^*$. Therefore, $s' \in M - P_L^{-1}[P(\text{Pre}_o(T(M)))]\,\Sigma^* = H$. Consequently, H is normal. □

Lemma 6: At convergence of the iterative solution procedure of Section IV-B for solving the ADP, the FSM $H_d^{\text{live}}(i)$ is the live diagnoser of $H(i)$; further, $H_d^{\text{live}}(i)$ is the live diagnoser extension of $H_d(i)$.

Proof: Since $\tilde{H}_d(k)$ is the diagnoser of $\tilde{H}(k)$ (cf., Lemma 4), $\tilde{H}_d^{\text{live}}(k)$ is the diagnoser of $\tilde{H}^{\text{live}}(k)$ (cf., Step B-1 of the solution procedure), and since $\tilde{H}^{\text{live}}(k)$ is the live extension of $\tilde{H}(k)$ (cf., Step B-1), then $\tilde{H}_d^{\text{live}}(k)$ is the live diagnoser of $\tilde{H}(k)$. At convergence of the solution procedure, we have $H_d^{\text{live}}(i) = \tilde{H}_d^{\text{live}}(k)$ (cf., Step B-3). Also, since at convergence, $H_d^{\uparrow C}(i) = H_d(i) = \tilde{H}_d(k)$ for some finite k, i (cf., Step B-3 of the solution procedure), we have that $P^{-1}(H_d^{\uparrow C}(i)) = P^{-1}(\tilde{H}_d(k))$. It follows from Steps A-2 and B-3 of the solution procedure that $H(i) = \tilde{H}(k)$. It follows that $H_d^{\text{live}}(i)$ is the live diagnoser of $H(i)$. Further, from Lemma 4 we have that $\tilde{H}_d(k)$ is the diagnoser corresponding to $\tilde{H}(k)$. Since at convergence $\tilde{H}_d(k) = H_d(i)$, and $\tilde{H}(k) = H(i)$, it follows that $H_d(i)$ is the diagnoser of $H(i)$. Since $H_d^{\text{live}}(i)$ is the live diagnoser of $H(i)$, then $H_d^{\text{live}}(i)$ is the live diagnoser extension of $H_d(i)$. □

Lemma 7: Consider $L = \bar{L} \subseteq \Sigma^*$ and $K = \bar{K} \subseteq L$. Then K is normal (with respect to P and L) iff K is of the form $K = P^{-1}(M) \cap L$ where $M \subseteq P(L)$ and $M = \bar{M}$.

Proof ($\Longrightarrow$)*:* K normal implies that $K = P^{-1}[P(K)] \cap L$. Choosing $M = P(K) \subseteq P(L)$ we have that $K = P^{-1}[M] \cap L$.

($\Longleftarrow$)*:* We prove that if $K = P^{-1}(M) \cap L$, then $P(K) = M$. It then follows that $K = P^{-1}[P(K)] \cap L$ which implies that K is normal.

1) To prove that $P(K) \subseteq M$: Let $t \in P(K)$. Then $\exists s \in K$ such that $P(s) = t$. Now, $s \in K$ implies that $s \in P^{-1}(M)$ and $s \in L$. Further, $s \in P^{-1}(M)$ implies that $P(s) \in M$ which in turn implies that $t \in M$.
2) To prove that $M \subseteq P(K)$: Pick $t \in M$

$$\begin{aligned} t \in M &\Rightarrow t \in P(L) \text{ since } M \subseteq P(L) \\ &\Rightarrow \exists s_L \in L : P(s_L) = t \\ &\Rightarrow s_L \in P^{-1}(t) \subseteq P^{-1}(M) \\ &\Rightarrow s_L \in P^{-1}(M) \cap L = K \\ &\Rightarrow P(s_L) \in P(K). \end{aligned}$$

□

APPENDIX B
ON THE LANGUAGES K_l

A. Proofs of Properties 5 and 7 of the Languages K_l

In proving Properties 5 and 7 below, we assume that the multiplicities of the cycles in the system model G corresponding to an indeterminate cycle in the diagnoser G_d (cf., paragraph following Example 2.1 and [10]) are equal to one. This assumption is not restrictive because it can be shown that Properties 5 and 7 that follow hold true for the general case of multiplicity greater than one as well. However, proofs of these properties for the general case are not presented here since these proofs are quite involved and since these properties are discussed primarily to motivate the choice of a particular K_l as initial condition for the active diagnosis problem.

Property 5: Given a nondiagnosable language L and K_l, $l \geq 0$

$$K_0 \text{ diagnosable} \Leftrightarrow K_l \text{ diagnosable}.$$

Proof ($\Leftarrow$)*:* Suppose K_0 is not diagnosable. Then we have from Property 3 that K_0 is not live and that there is a terminating trace in K_0 that violates Definition 2. Let s_0 be this trace; then $s_0 \in K_0$ is such that $K_0/s_0 = \emptyset$, $\Sigma_{fi} \in s_0$ for some failure type F_i, and $\exists \tilde{s_0} \in K_0$ such that $K_0/\tilde{s_0} = \emptyset$, $P(s_0) = P(\tilde{s_0})$, and $\Sigma_{fi} \notin \tilde{s_0}$. Now since K_0

is obtained from L (which is live) by removing traces that go through an F_i-indeterminate cycle in the diagnoser G_d of L, $s_0, \tilde{s_0}$ are of the form $s_0 = su_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_n$, and $\tilde{s_0} = \tilde{s}v_1\sigma_1\cdots v_{n-1}\sigma_{n-1}v_n$ where $\Sigma_{fi} \in s$, $s_f \in \Sigma_o$, $P(s) = P(\tilde{s})$, $u_j, v_j \in \Sigma_{uo}^*$, $\sigma_j \in \Sigma_o$, $\Sigma_{fi} \notin \tilde{s_0}$, $\delta_d(q_0, P(s)) = q_1$, $\delta_d(q_j, \sigma_j) = q_{(j+1)\bmod n}$ and $\{q_j\}_{j=1}^n$ form an F_i-indeterminate cycle in G_d. Consider next the traces $s_l, \tilde{s_l} \in K_l$, corresponding to the traces $s_0, \tilde{s}_0 \in K_0$ such that $s_l = s(u_1\sigma_1\cdots u_n\sigma_n)^l u_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_n$ and $\tilde{s}_l = \tilde{s}(v_1\sigma_1\cdots v_n\sigma_n)^l v_1\sigma_1\cdots v_{n-1}\sigma_{n-1}v_n$. Since $\delta(x_0, s_0) = \delta(x_0, s_l)$ by the definition of an indeterminate cycle, then $K_l/s_l = K_0/s_0 = \emptyset$ and $K_l/\tilde{s}_l = K_0/\tilde{s}_0 = \emptyset$. Therefore, s_l violates Definition 2 for K_l, and hence K_l is not diagnosable.

($\Rightarrow$): Interchange K_l and K_0 in the above proof and follow the same arguments. □

Property 7: Given a nondiagnosable language L, and K_l, $l \geq 0$ diagnosable, there exist $m > 0$ and $p \in K_0$ such that for all $n \geq 1$

$$\text{delay}(p, K_0) < \text{delay}(p, K_m) < \text{delay}(p, K_{m+n}).$$

Proof: Consider $p \in K_0$ with $p_f \in \Sigma_{fi}$ for some failure type F_i, such that there exists $s_0 \in K_0$ where $s_0 = p\omega u_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_n$, $\omega_f \in \Sigma_o$, $u_j \in \Sigma_{uo}^*$, $\sigma_j \in \Sigma_o$, $\delta_d(q_0, P(p\omega)) = q_1$, $\delta_d(q_j, \sigma_j) = q_{(j+1)\bmod n}$ and $\{q_j\}_{j=1}^n$ form an F_i-indeterminate cycle in G_d. We know that such a p and s_0 exist by the definition of K_0. Let $t \in K_0/p$ be such that the maximum delay in detecting the failure p_f occurs along the trace t in K_0 and let $\text{delay}(p, K_0) = M$. Pick any t' in K_0/s_0 such that detection of the failure p_f along the trace s_0 occurs after the system executes the trace s_0t' but not before, i.e., $\forall \omega_1 \in P_{K_0}^{-1}[P(s_1t')]$ $(\Sigma_{fi} \in \omega_1)$ and $(\forall v \in \overline{t'})$ $\exists \omega_2 \in P_{K_0}^{-1}[P(sv)] : \Sigma_{fi} \notin \omega_2$. Note that t' may be the empty trace in the case where $K_0/s_0 = \emptyset$; in this case the failure is diagnosed right after s_0 by noting that no further event occurs in the system (recall the results of Section III on diagnosing failures in a nonlive language). Consider next the trace $s_m \in K_m$ where $s_m = p\omega(u_1\sigma_1\cdots u_n\sigma_n)^m u_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_n$. Since $\delta(x_0, s_0) = \delta(x_0, s_m)$, then the delay in detecting the failure p_f along the trace s_m in K_m is given by $|\omega| + (|u_1\sigma_1\cdots u_n\sigma_n u_n| \times m) + |u_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_n| + |t'|$. It is not difficult to see that by choosing a large enough m we can get $|\omega| + (|u_1\sigma_1\cdots u_n\sigma_n u_n| \times m) + |u_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_n| + |t'| > M$, i.e., the maximum delay in detecting the failure event p_f occurs along the trace $\omega(u_1\sigma_1\cdots u_n\sigma_n)^m u_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_nt'$. It then follows that $\text{delay}(p, K_m) < \text{delay}(p, K_{m+n})$ for all $n \geq 1$ since $\text{delay}(p, K_{m+n}) = |\omega| + (|u_1\sigma_1\cdots u_n\sigma_n u_n| \times (m+n) + |u_1\sigma_1\cdots u_{n-1}\sigma_{n-1}u_n| + |t'|$. □

B. Procedures to Obtain a Generator G^{legal} of K_l and the Refined Machine G^{ref} in the Case Where G_d Has No Interleaved Indeterminate Cycles

We now present: 1) a procedure to obtain an FSM that generates K_l (henceforth referred to as G^{legal}) from the system model G and from the diagnoser G_d and 2) a procedure to obtain the refined system model G^{ref} such that $L(G^{\text{ref}}) = L(G)$ and such that G^{ref} includes as a submachine the generator G^{legal} of K_l. These procedures are presented for the case where G_d has no interleaved indeterminate cycles. Recall from Section IV-C that the FSM G^{ref} is necessary to implement the solution procedure for the ADP presented in Section IV-B.

Procedure to Obtain the Generator G^{legal} of K_l: Given the language L generated by the system G, and given K_l, $l \geq 0$, the generator G^{legal} of K_l can be obtained from the system model G, and from the diagnoser G_d corresponding to L, by the following two-step procedure.

Step 1: Refine $G_d = (Q_d, \Sigma_o, \delta_d, q_0)$ such that every indeterminate cycle in G_d is "expanded out" l times but left "open" (not completed) after the $l+1$th copy. This step is explained in detail below (Steps 1-1 to 1-5).

Let $G_d' = (Q_d', \Sigma_o, \delta_d', q_0)$ denote the refined machine. Then $L(G_d') = P(K_l)$.

Step 2: Define $G^{\text{legal}} = P^{-1}(G_d') \times G$. In other words, G^{legal} is obtained from G_d' by: 1) adding self-loops at every state q of G_d' due to all $\sigma \in \Sigma_{uo}$ and 2) performing the product of the resulting machine with G.

It is straightforward to implement Step 2 above. We now focus on Step 1 of the procedure.

Procedure to Obtain the Refined FSM G_d' from the FSM G_d: The inputs to this procedure are as follows:

1) the FSM $G_d = (Q_d, \Sigma_o, \delta_d, q_0)$;
2) the set $C(G_d)$ of F_i-indeterminate cycles in G_d, for all failure types F_i.

We assume that each $C^i \in C(G_d)$ is given by $C^i = \{(q_1^i, q_2^i, \cdots, q_{k_i}^i), (\sigma_1^i, \sigma_2^i, \cdots, \sigma_{k_i}^i)\}$ where the set of states $\{q_j^i\}_{j=1}^{k_i}$ forms an indeterminate cycle with corresponding event sequence $\{\sigma_j^i\}_{j=1}^{k_i}$. Suppose that the set of states $S = \{q_i\}_{i=1}^n$ forms an indeterminate cycle in G_d with corresponding event sequence $\{\sigma_i\}_{i=1}^n$. Suppose further that $\exists z_1, z_2 \in Q_d - S$ and $\alpha_1, \alpha_2 \in \Sigma_o$ such that $\delta_d(z_1, \alpha_1) = q_i$ and $\delta_d(z_2, \alpha_2) = q_j$, $1 \leq i, j \leq n$, $i \neq j$. We refer to such states z_1 and z_2 as *entry states* to the cycle. Then we will consider as two distinct cycles, $C^1 = \{(q_i, q_{i+1}, \cdots, q_n, q_1, \cdots, q_{i-1})$ $(\sigma_i, \sigma_{i+1}, \cdots, \sigma_n, \sigma_1, \cdots, \sigma_{i-1})\}$, and $C^2 = \{(q_j, q_{j+1}, \cdots, q_n, q_1, \cdots, q_{j-1})$ $(\sigma_j, \sigma_{j+1}, \cdots, \sigma_n, \sigma_1, \cdots, \sigma_{j-1})\}$. In other words any set S of n elements that constitutes an indeterminate cycle could give rise to up to n distinct cycles $C^1, C^2, \cdots, C^n$ depending on the number of distinct entry states into the cycle, from states outside of the cycle.

For each indeterminate cycle $C^i \in C(G_d)$, we build an FSM H^i such that H^i refines the cycle C^i. The refined machine G_d' is then given by

$$G_d' = H^1 \times H^2 \cdots H^M$$

where M refers to the total number of indeterminate cycles in G_d.

We now present a five-step procedure to build H^i given G_d and given an indeterminate cycle

$$C^i = \{(q_1, q_2, \cdots, q_n), (\sigma_1, \sigma_2, \cdots, \sigma_n)\}.$$

Step 1-1: Define $G_1 = (Q_d, \Sigma_o, \delta_1, q_0)$ where

$$\delta_1(q,\sigma) = \begin{cases} \delta_d(q,\sigma), & \text{if } q \neq q_1 \vee \sigma \neq \sigma_1 \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

In other words, G_1 is the same as G_d, except that at state q_1, the transition due to σ_1 is deleted in G_1. Note that not all states of G_1 may be accessible.

Step 1-2: Define $G_2 = (Q_2, \Sigma_o, \delta_2, p_1^1)$ where we have the first set of equations shown at the bottom of the page. Note that n refers to the number of states in the cycle C^i and l refers to the number of copies of these states in the refined machine. Also note that the transition due to the event σ_n is not defined at the final state p_n^{l+1}. Thus, G_2 has the cycle C^i expanded out l times but left "open" (not completed) after the $l+1$th copy.

Step 1-3: Merge G_1 and G_2 with the state q_1 of G_1 set equal to the state p_1^1 of G_2. Let $G_{12} = (Q_{12}, \Sigma_o, \delta_{12}, q_0)$ denote the merged machine where

$$Q_{12} = Q_1 \cup Q_2 \text{ (with } q_1 = p_1^1)$$
$$\delta_{12}(q,\sigma) = \begin{cases} \delta_1(q,\sigma), & \text{if } \delta_1(q,\sigma) \text{ is defined} \\ \delta_2(q,\sigma), & \text{if } \delta_2(q,\sigma) \text{ is defined} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Note that the states of Q_1 are distinct from those of Q_2, except for the state q_1 which is set equal to p_1^1. At the state q_1 the transition due to σ_1 is not defined in G_1, while at the state p_1^1 of G_2 the only transition defined is due to σ_1. Thus, we see that δ_{12} is a well-defined function.

Step 1-3 simply "pastes" together the machines obtained in Steps 1-1 and 1-2.

Step 1-4: Complete the transition function δ_{12} of G_{12} at the states $\{p_j^k\}, k = 1, \cdots, n, j = 1, \cdots, l+1$ as follows:

$$\delta_{12}(p_j^k, \sigma) = \delta_d(q_j, \sigma)\ \forall \sigma \in \Sigma_o - \{\sigma_j\}, \quad j = 1, \cdots, n,\ k = 1, \cdots, l+1.$$

Recall that Q_d is the state space of the diagnoser G_d, δ_d is the transition function of G_d, and $C^i = \{(q_1, q_2, \cdots, q_n), (\sigma_1, \sigma_2, \cdots, \sigma_n)\}$.

Step 1-4 completes the transition function of G_{12} so as to ensure that the language generated by the complete machine is equal to the language generated by G_d excluding those traces that go through the indeterminate cycle C^i more than l times.

Step 1-5: Let $H^i = \text{Acc}(G_{12})$ where $\text{Acc}(G)$ denotes the accessible part of G.

Step 1-5 simply removes from G_{12} all states that are not reachable from the initial state q_0, and their corresponding transitions.

Procedure to Obtain the Refined System Model G^{ref}: Given the language L generated by the system G, and given K_l, $l \geq 0$, the procedure to obtain a refined system model G^{ref}, that includes as a submachine the generator G^{legal} of K_l, is identical to the above procedure for generating G^{legal} except for Step 1-2 which is modified as follows.

Step 1-2: Define $G_2 = (Q_2, \Sigma_o, \delta_2, p_1^1)$ where we have the second set of equations, also shown at the bottom of the page. Note that the transition due to the event σ_n is defined at the final state p_n^{l+1}. Thus, G_2 has the cycle C^i expanded out l times and completed after the $l+1$th time since $\delta_2(p_n^{l+1}, \sigma_n)$ is defined to be p_1^1, the initial state of G_2. This is unlike the previous procedure for obtaining the generator of K_l where the cycle is left open after the $l+1$th copy.

In this case we have $L(G_d') = L(G_d)$. It is then straightforward to see that this procedure results in a refined system model G^{ref} such that $L(G) = L(G^{\text{ref}})$ and further that $G^{\text{legal}} \sqsubseteq G^{\text{ref}}$.

Examples illustrating the above two procedures can be found in [9].

We conclude with the following remark on the case where the diagnoser G_d has interleaved indeterminate cycles. Examination of the above procedures reveals that the only step that needs to be modified in building a generator for K_l in this case is Step 1-2 which "expands" out the cycles. In the

---

$$Q_2 = \{p_j^k : j \in \{1, \cdots, n\}, k \in \{1, \cdots, l+1\}\}$$
$$\delta_2(p_j^k, \sigma_j) = \begin{cases} p_{j+1}^k, & \text{if } j = 1, \cdots, n-1, k = 1, \cdots, l+1 \\ p_1^{k+1}, & \text{if } j = n, k = 1, \cdots, l \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

$$Q_2 = \{p_j^k : j \in \{1, \cdots, n\}, k \in \{1, \cdots, l+1\}\}$$
$$\delta_2(p_j^k, \sigma_j) = \begin{cases} p_{j+1}^k, & \text{if } j = 1, \cdots, n-1, k = 1, \cdots, l+1 \\ p_1^{k+1}, & \text{if } j = n, k = 1, \cdots, l \\ p_1^1, & \text{if } j = n, k = l+1 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

case of interleaved cycles we need to expand every elementary cycle in the given interleaved cycle l times and further account for the fact that these cycles may appear in any order (i.e., l times the first elementary cycle followed by l times the second cycle, or, $l-2$ times the first cycle followed by l times the second cycle, again followed by the first cycle once, and so on) in the resulting expanded machine. While it is difficult to precisely write down the transition function of the FSM G_2 of Step 1-2 in this case, it is not difficult to see that it is still possible to build the FSM G_2 which in turn implies that it is possible to obtain an FSM G^{legal} (and G^{ref}) such that it that generates the language K_l in the case where the diagnoser G_d has interleaved indeterminate cycles.

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers whose comments helped improve the presentation of the paper. They also thank G. Barrett for his pertinent comments regarding the proof of Theorem 2.

REFERENCES

[1] R. D. Brandt, V. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham, "Formulas for calculating supremal controllable and normal sublanguages," *Syst. Contr. Lett.*, vol. 15, no. 2, pp. 111–117, Aug. 1990.
[2] C. Cassandras, S. Lafortune, and G. Olsder, "Introduction to the modeling, control and optimization of discrete event systems," in *Trends in Control. A European Perspective*, A. Isidori, Ed. New York: Springer-Verlag, 1995, pp. 217–291.
[3] C. Golaszewski and P. Ramadge, "Control of discrete event processes with forced events," in *Proc. 26th IEEE Conf. Decision and Control*, Los Angeles, CA, Dec. 1987, pp. 247–251.
[4] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.
[5] S. Lafortune and E. Chen, "The infimal closed controllable superlanguage and its application in supervisory control," *IEEE Trans. Automat. Contr.*, vol. 35, pp. 398–405, Apr. 1990.
[6] F. Lin, private communication, 1990.
[7] F. Lin and W. M. Wonham, "On observability of discrete-event systems," *Inform. Sci.*, vol. 44, pp. 173–198, 1988.
[8] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, pp. 81–98, Jan. 1989.
[9] M. Sampath, "A discrete event systems approach to failure diagnosis," Ph.D. dissertation, Dept. Electrical Engineering and Computer Science, Univ. Michigan, Ann Arbor, 1995.
[10] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete event systems," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 1555–1575, Sept. 1995.
[11] ——, "Failure diagnosis using discrete event models," *IEEE Trans. Contr. Syst. Technol.*, vol. 4, pp. 105–124, Mar. 1996.
[12] J. G. Thistle, "Control of infinite behavior of discrete-event systems," Ph.D dissertation, Systems Control Group Rep. 9012, Univ. Toronto, Jan. 1991.
[13] ——, "Supervisory control of discrete event systems," *Math. Computer Modeling*, vol. 23, nos. 11/12, pp. 25–53, 1996.

Meera Sampath (S'95–M'96) received the B.E. degree from the College of Engineering, Guindy, Madras, India, the M.Tech. degree from the Indian Institute of Technology, Kharagpur, India, and the Ph.D. degree from the University of Michigan, Ann Arbor, all in electrical engineering, in 1988, 1990, and 1995, respectively. She also received the certificate in Transportation Studies from the University of Michigan Intelligent Transportation Systems program in 1995.

During the summer of 1995, she was an intern at Johnson Controls Inc., Milwaukee, WI. Since August 1996, she has been with the Control and Diagnostics Laboratory in the Wilson Center for Research and Technology, Xerox Corporation. Her current research interests include automated failure diagnosis and discrete-event systems.

Dr. Sampath was a recipient of the University of Michigan Distinguished Dissertation Award and the University of Michigan College of Engineering Distinguished Achievement Award in 1996.

Stéphane Lafortune (S'78–M'86–SM'97) received the B.Eng. degree from École Polytechnique de Montréal in 1980, the M.Eng. degree from McGill University in 1982, and the Ph.D. degree from the University of California, Berkeley in 1986, all in electrical engineering.

Since 1986, he has been with the Department of Electrical Engineering and Computer Science at the University of Michigan where he is an Associate Professor. His research interests include discrete-event systems (modeling, supervisory control, failure diagnosis, and applications) and intelligent transportation systems.

Dr. Lafortune received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper coauthored with S. L. Chung and F. Lin).

Demosthenis Teneketzis (M'87–SM'97) received the B.S. degree in electrical engineering from the University of Patras, Greece, in l974 and the M.S., E.E., and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in l976, l977, and l979, respectively.

From 1979 to l980, he worked for Systems Control Inc., Palo Alto, CA, and from l980 to l984 he was with Alphatech Inc., Burlington, MA. Since September l984, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science. In winter and spring, l992, he was a Visiting Professor at the Institute for Signal and Information Processing of the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland. His current research interests include stochastic control, decentralized systems, queueing and communication networks, stochastic scheduling and resource allocation problems, and discrete-event systems.