Diagnosis of Active Systems by Semantic Patterns

Gianfranco Lamperti, *Member, IEEE,* and Xiangfu Zhao, *Member, IEEE*

***Abstract*—A gap still exists between complex discrete-event systems (DESs) and the effectiveness of the state-of-the-art diagnosis techniques, where faults are defined at component levels and diagnoses incorporate the occurrences of component faults. All these approaches to diagnosis are context-free, in as much diagnosis is anchored to components, irrespective of the context in which they are embedded. By contrast, since complex DESs are naturally organized in hierarchies of contexts, different diagnosis rules are to be defined for different contexts. Diagnosis rules are specified based on associations between context-sensitive faults and regular expressions, called semantic patterns. Since the alphabets of such regular expressions are stratified, so that the semantic patterns of a context are defined based on the interface symbols of its subcontexts only, separation of concerns is achieved, and the expressive power of diagnosis is enhanced. This new approach to diagnosis is bound to seemingly contradictory but nonetheless possible scenarios: a DES can be normal despite the faulty behavior of a number of its components; also, it can be faulty despite the normal behavior of all its components.**

***Index Terms*—Artificial intelligence, decision support systems, discrete-event systems (DESs), fault diagnosis, intelligent systems.**

I. Introduction

In THE LAST decades, automated diagnosis has become increasingly important for the safety of society. It suffices to consider the India major blackout in July 2012, the largest power outage in history, occurring as two separate yet contiguous events, which affected over 620 million people (half of India's population), across 22 states, with an estimated 32 gigawatts of generating capacity being taken offline. The investigation committee concluded that among other factors responsible for the blackout was the loss of a 400-V transmission line caused by misbehavior of the protection system [1].

Automated diagnosis of discrete-event systems (DESs) [2] may play an important role in the prevention of harmful events like a large power outage. A DES is a discrete-state, event-driven system whose state evolution depends on the occurrence of discrete events over time.[1] A wide variety of techniques for diagnosis of DESs have been proposed in the last years. However, after the seminal work of Sampath *et al.* [3], the basic notions of fault and diagnosis have been remaining conceptually unchanged until recent works [4]–[6]. Researchers have mainly focused on relevant yet different aspects, including incrementality [7], [8], distribution/decentralization [9]–[16], uncertainty/incompleteness [17]–[22], planning/SAT approaches [23], [24], and time [25]–[27]. In [7], a technique for modular diagnosis, amenable to parallel implementation, is presented. The main goal of the technique is the reconstruction of the behavior of the active system starting from a set of observable events. The diagnostic process involves three steps: 1) interpretation; 2) merging; and 3) diagnosis generation. Interpretation generates a representation of the behavior of a part of the active system based on observable events. Merging combines the result of several interpretations into a new, broader interpretation. The eventual diagnostic information is generated on the basis of fault events possibly incorporated within the reconstructed behavior. In [8], an incremental diagnosis technique is proposed, assuming that observations are partially ordered and represented by finite automata. The technique consists in slicing the observation automaton and in computing diagnosis slices to be eventually combined in the global diagnosis. In [9], the diagnosis technique proposed in [7] is further extended by a preliminary step called reconstruction planning, which draws a hierarchical decomposition of the behavior reconstruction problem. The modular approach is formally defined and applied to the power transmission network domain. In [10], a general method for diagnosis of large DESs is proposed. The method results from the combination of two diagnosis techniques: diagnosers and simulation. Diagnosers are generated based on local behaviors to compute local diagnoses. The diagnosis of the system is determined based on the coordination of such local diagnoses. In [11], a coordinated decentralized architecture is presented, which consists of local sites communicating with a coordinator that is responsible for diagnosing the failures occurring in the system. The notion of diagnosability, originally introduced in [3] for centralized systems, is extended for such an architecture. To realize the architecture, three protocols are specified and analyzed in terms of diagnostic properties. The decentralized diagnosis approach proposed in [10] is further extended in [12] by means of incrementality to provide online diagnosis and assist supervision operators. In [13], the effect of the communication delays on the performance of a coordinated decentralized architecture is studied, as a refinement of [11]. Specifically, the assumption that messages are received by the coordinator in the order in which they are sent globally is relaxed. In [14], the decentralized approach to diagnosis is extended to reconfigurable DESs. After modeling the decentralized DES in terms of system topology and component behavior, the notion of reconfiguration is formally

Manuscript received July 8, 2013; accepted October 19, 2013. Date of publication January 27, 2014; date of current version July 15, 2014. This work was supported in part by the NSFC under Grant 61003101 and in part by the Zhejiang Provincial Natural Science Foundation under Grant Y1100191. This paper was recommended by Associate Editor F. Y. Wang.

G. Lamperti is with the Department of Information Engineering, University of Brescia, Brescia 25123, Italy (e-mail: lamperti@ing.unibs.it).

X. Zhao is with the College of Mathematics, Physics and Information Engineering, Zhejiang Normal University, Zhejiang 321004, China (e-mail: xiangfuzhao@zjnu.cn).

Digital Object Identifier 10.1109/TSMC.2013.2296277

[1]DESs are a special class of dynamic systems. In a dynamic system, a state is a collection of real numbers, and the state evolution is determined by a fixed rule that describes what future states follow from the current state.

defined: if reconfiguration fulfills a property called safety then the decentralized diagnosis approach can be extended to reconfigurable DESs. In [15], the problem of diagnosing large DESs is addressed. Specifically, a formal framework for online decentralized diagnosis (and relevant implementation) is presented, which does not require the computation of the global model of the DES. The framework is applied to the monitoring of a real telecommunication network. In [16], a technique for decentralized diagnosis of DESs is presented, where multiple diagnosers exist, each possessing its own set of sensors, without involving any communication among diagnosers or to any coordinators. The notion of codiagnosability is introduced, that requires a failure to be detected by one of the diagnosers within a bounded delay. Efficient algorithms are provided for testing codiagnosability, for computing the bound in delay of diagnosis, for offline synthesis of individual diagnosers, and for online diagnosis. In [17], the notion of an uncertain temporal observation of a DES is introduced, which is independent of any specific diagnosis technique. Such an observation is represented by an acyclic graph, where nodes are marked by uncertain observable events while arcs define a partial temporal ordering between nodes. In [18], a method for similarity-based diagnosis of DESs is proposed: the solution of a diagnosis problem is supported by the solution of another problem, provided the two problems are somewhat similar. In [19], a technique for model-based diagnosis of DESs with an incomplete model is presented, that is based on the P-synchronization product of automata, a generalization of the classical synchronization product. In [20], the notion of monotonic monitoring of DESs is introduced, which is supported by specific constraints on the fragmentation of the uncertain temporal observation, leading to the notion of stratification, specifically, stratified observations support monotonic monitoring of active systems. In [21], the diagnosis of a DES with an incomplete model is performed with the support of a diagnoser constructed using a learning technique, by forming hypotheses that explain the discrepancies between the actual output and the output derived from the model. In [22], a technique for reasoning on partially ordered observable events based on two temporal windows is presented. In [23], an exploration of the use of planning technology for the automated generation of diagnoses is studied. In [24], a conflict-based approach to diagnosis of DESs is proposed, which does not require the reconstruction of the system behavior, where test of consistency is implemented by a SAT solver. In [25], diagnosis in the context of time automata is studied. An algorithm for checking diagnosability and a technique for constructing a diagnoser for a diagnosable timed automaton are provided. In [26], a method for fault detection and diagnosis of real-time DESs is presented, including diagnosability checking. In [27], a novel approach to diagnosis of complex systems modeled by communicating timed automata is presented, where each component is modeled by a timed automaton integrating different operating modes, while the communication between components is carried out by a control module. This modeling supports formal verification of the complex-system model and its diagnoser.

However, despite this flourishing of techniques, a fault is invariably associated with the occurrence of an event (or transition) at component level within the evolution of the system. Thus, diagnosing a DES amounts to uncovering the set of component faults occurring during the system evolution. This paper claims that anchoring faults at component levels is too restrictive an approach when complex DESs are involved. In fact, it is our belief that monitoring and diagnosing complex systems, such as a nuclear plant or a large power network, requires some sort of abstraction and separation of concerns.

The remainder of the paper is organized as follows. Section II introduces the notion of context-sensitivity in diagnosis of DESs. Section III recalls the class of asynchronous DESs called active systems. Section IV defines the notion of diagnosis problem. Section V defines the notion of history projection. Section VI defines in declarative terms what the solution of a diagnosis-problem is. Section VII presents a technique for offline preprocessing diagnosis rules into deterministic automata called semantic spaces. Section VIII presents the diagnosis technique performed online. Section IX formally proves the soundness and completeness of the diagnosis technique. Section X provides some discussion and comparison about the state-of-the-art relevant approaches. Section XI concludes the paper.

II. Context-Sensitivity

The topology of a complex DES is organized in a hierarchy, where the root corresponds to the whole system, leaves to components, and intermediate nodes to subsystems. Since in a DES faults are traditionally defined at component levels, there is no possibility to provide a hierarchy of diagnoses adhering to the hierarchy of the system. Trivially, a sub-DES is faulty if and only if it includes (at least) one faulty component. We call this commonly-used approach context-free diagnosis. While context-free diagnosis may be adequate for simple systems, it is doomed to be unsatisfactory when applied to complex DESs. The hierarchical structure of a complex DES suggests to organize the diagnosis rules within a hierarchy that parallels the hierarchical structure of the DES: each subsystem has its proper set of diagnosis rules, which may or may not depend on the rules of inner subsystems. We call this alternative approach context-sensitive diagnosis.

The idea of context-sensitivity in diagnosis was inspired by the problem of (formal) language specification. The classical hierarchy proposed by Chomsky [28] incorporates four types of generative grammars of growing expressiveness. Within the hierarchy, Type-2 and Type-1 grammars are means to specify the syntax of context-free and context-sensitive languages, respectively. For practical reasons, Type-2 grammars (in BNF notation) have become the standard for the specification of the syntax of programming languages. However, since, generally speaking, the syntax rules of programming languages depend on the context, production rules in BNF notation are unable to force all the syntax constraints of the language.[2] In compilers, the check of these (context-sensitive) constraints is generally left to semantic analysis.

[2]For example, the list of actual parameters in a function call shall match, in number and types, the list of formal parameters.

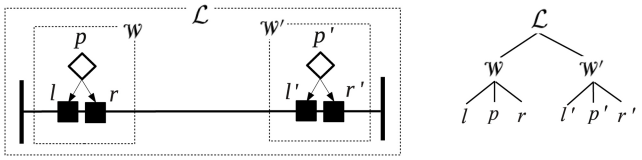

Fig. 1. Protected power transmission line (left). Context hierarchy (right).

Generally, the idea of context-sensitivity can be profitably injected into diagnosis of DESs in three ways.

1) The transition of a component is not considered either normal or faulty on its own: it depends on the context in which such a transition is triggered.
2) The fault of a component is not necessarily ascribed to one transition: it can be the result of a pattern of transitions.
3) Normal or faulty behavior of a subsystem is not necessarily determined by the normal or faulty behavior of its components (or inner subsystems).

Context-sensitive diagnosis is bound to seemingly contradictory but nonetheless possible results, called paradoxes.

A. Positive Paradox

The positive paradox states that a (sub)system may be normal despite the faulty behavior of a number of its components (or inner subsystems). Example 1 aims to clarify this assertion.

Example 1: Shown in the left side of Fig. 1 is a simplified representation of a power transmission line $\mathcal{L}$. The line is protected on both sides by a redundant protection hardware, called $\mathcal{W}$ and $\mathcal{W}'$, respectively, involving one protection device and two breakers. For instance, $\mathcal{W}$ incorporates protection device p, and breakers l and r. When a lightning strikes the line, a short circuit may occur on the latter. The protection system is designed to open the breakers to isolate the line, which eventually causes the extinction of the short. To this end, when detecting a short circuit, a protection device is expected to trigger both breakers to open. If this causes the extinction of the short circuit, the protection device commands both breakers to close to reconnect the line to the power network. A protection device is faulty either when, upon the detection of the short circuit it commands breakers to close (instead of opening) or, after the extinction of the short circuit, it commands breakers to open (instead of closing). A breaker is faulty when, after receiving the command from the protection device to change state, it remains in its state. A minimal (nonredundant) protection hardware would incorporate one single protection device and one single breaker on each side of the line. Thus, when one of the two devices is faulty, the line cannot be isolated. By contrast, in the redundant protection hardware in Fig. 1, it suffices the normal behavior of the protection device and of one breaker (for each side) to guarantee the isolation of the line. Consider the following two scenarios.

1) In $\mathcal{W}$, l is faulty, while p and r are normal. In $\mathcal{W}'$, l' is faulty, while p' and r' are normal. The diagnosis is $\delta_1 = \{l, l'\}$. Owing to redundancy, the behavior of the protected line is in fact normal, as the line is isolated, despite the faulty behavior of the two breakers.
2) In $\mathcal{W}$, l is faulty, while p and r are normal. In $\mathcal{W}'$, p' is faulty, while l' and r' are normal. The diagnosis is $\delta_2 = \{l, p'\}$. However, owing to p', $\mathcal{W}'$ fails to open, causing the failure of the whole protected line.

Albeit the two diagnoses δ_1 and δ_2 differ in one component only (l' versus p'), the behavior of the protected line in the first scenario is normal, while it is faulty in the second one. The point is, the given diagnoses do not explicitly account for such a distinction. More generally, we can consider several subsystems of the protected line within a hierarchy, and require for each of them a diagnosis. Such a hierarchy is displayed in the right side of Fig. 1, where $\mathcal{L}$ denotes the whole protected line. According to such a hierarchy, the diagnosis of the first scenario is $\{l, l'\}$, while in the second scenario the diagnosis is $\{l, p', \mathcal{W}', \mathcal{L}\}$. Comparing the two diagnoses, we conclude that in the first scenario two components are faulty but their misbehavior is not propagated to higher subsystems. By contrast, in the second scenario, besides two faulty components (l and p'), $\mathcal{W}'$ and $\mathcal{L}$ are faulty too.

B. Negative Paradox

The negative paradox states that a (sub)system can be faulty despite the normal behavior of all its components (or inner subsystems). This is true also in systems other than DESs.

1) A software system can be faulty even if all its software components are bug-free.
2) Injustice may occur even if all laws are respected.
3) A human society can be bound to dictatorship notwithstanding all its democratic institutions.[3]

Example 2 instantiates the negative paradox to the referenced application-domain of power transmission networks.

Example 2: With reference to Fig. 1, the isolation of the line causes the extinction of the short circuit because the short is no longer fed by any current. This is why the protection system is designed to reconnect the line to the network once the short is extinguished (by closing breakers). Suppose now that, instead of a lightning, what causes the short is a tree fallen on the line. In this case, the reconnection of the line to the network presumably activates the short circuit again, as the tree is likely to be still on the line, thereby causing the line to be isolated anew (and, this time, permanently). Clearly, even assuming that the behavior of protection devices and breakers was normal, the behavior of the line is actually faulty.

[3]An anecdote about the Austrian logician, mathematician, and philosopher Kurt Friedrich Gödel is told in [29]. In December 1947, Gödel went to his American citizenship hearing in New Jersey. As his witnesses, Gödel brought his two closest friends, Oskar Morgenstern and Albert Einstein. Gödel, in his usual manner, had read extensively in preparing for the hearing. In the course of his studies, Gödel decided that he had discovered a flaw in the U.S. Constitution, a contradiction which would allow the U.S. to be turned into a dictatorship. Gödel seemed to feel a need to make this known. However, his friends Morgenstern and Einstein warned Gödel that it would be a disaster to confront his citizenship examiner with visions of a Constitutional flaw leading to an American dictatorship. The judge happened to remark how fortunate it was that the U.S. was not a dictatorship, which Gödel took as a cue to explain his discovery. This, however, did not compromise is citizenship hearing, which he eventually obtained in April 1948.

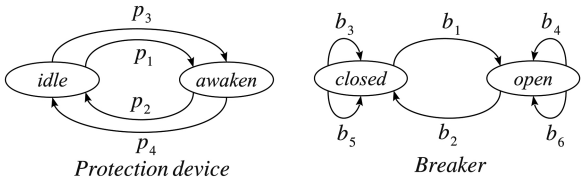

Fig. 2. Behavioral model of protection device (left). Breaker (right).

Coping with negative paradoxes is essential when the behavior of a complex DES cannot be foreseen at design-time (which is almost always the case for complex DESs). To face this uncertainty, a set of constraints can be associated with the nodes of the DES hierarchy, aimed at intercepting relevant faulty behavior. These constraints parallel the requirements of software systems, which need to be validated independently of the correct behavior of software components.

III. Active Systems

Active systems [30] are a subclass of DESs, where behavior composition is asynchronous (rather than synchronous as in [3] and most other approaches). An active system is a network of components that are connected to one another through links. Each component is modeled as a communicating automaton [31], that reacts to events either coming from the external world or from neighboring components through links, and is endowed with input and output terminals. When an event e is ready at the input terminal of a component C that is in state S, and there exists a transition T leaving state S upon event e, transition T is said to be triggerable. As such, a transition is triggered by an input event and generates a (possibly empty) set of output events. The latters are thus made available as input events at the input terminals of neighboring components, while the input (triggering) event is consumed. However, a transition can be triggered only if all links, toward which output events are generated, are empty (no event in links). The actual triggering of a transition may generate other events at some output terminals of the same component. The mode in which a system can behave is constrained by its topology and the component models. The whole (even unbounded) set of evolutions of an active system is confined to a finite automaton representing the global model of the system. In principle, the global model of a system can be generated automatically based on the model of components and their connections. However, in real large-scale systems, this becomes prohibitive because of the huge number of states. Therefore, a strong assumption on diagnosis of active systems is the unavailability of the global model. Thus, the global model is intended for formal reasons only. The global model of a system Σ, rooted at the initial state Σ_0, is called the behavior space of Σ, written $Bsp(\Sigma, \Sigma_0)$. A state in $Bsp(\Sigma, \Sigma_0)$ is final when all links are empty (all events have been consumed). A history in $Bsp(\Sigma, \Sigma_0)$ is the sequence of component transitions marking the arcs of a path rooted in Σ_0 and ending at a final state. The component relevant to a transition T of a history is denoted C_T.

Example 3: With reference to Example 1, shown in Fig. 2 are the behavioral models of protection device and breaker.

TABLE I
Details for Transitions of Protection Device p

T	*Details for transitions of protection device* p
p_1	p detects low voltage and outputs *op* event
p_2	p detects normal voltage and outputs *cl* event
p_3	p detects low voltage, yet it outputs *cl* event
p_4	p detects normal voltage, yet it outputs *op* event

TABLE II
Details for Transitions of Breaker b

T	*Details for transitions of breaker* b
b_1	b consumes *op* event and opens
b_2	b consumes *cl* event and closes
b_3	b consumes *op* event, yet it remains closed
b_4	b consumes *cl* event, yet it remains open
b_5	b consumes *cl* event
b_6	b consumes *op* event

The automaton of the protection device includes two states, *idle* and *awaken*, and four transitions, $p_1 \cdots p_4$. Details for transitions of the protection device are given in Table I.

Accordingly, p_1 moves the protection device from *idle* to *awaken* upon the detection of the short circuit (lowering of voltage), thereby causing the output of *op* event (to open the relevant breakers), while p_2 moves the protection device from *awaken* to *idle* upon the detection of the extinction of the short circuit (voltage becomes normal), thereby causing the output of *cl* event (to close the relevant breakers). However, the protection device may be faulty in two ways: either it outputs event *cl* instead of *op* at low-voltage detection (p_3), or it outputs event *op* instead of *cl* at normal-voltage detection (p_4). The automaton of the breaker includes two states, *closed* and *open*, and six transitions, namely, $b_1 \cdots b_6$. Details for transitions of the breaker are given in Table II.

Accordingly, b_1 moves the breaker from *closed* to *open* upon the consumption of event *op* (sent by the protection device), while b_2 moves the breaker from *open* to *closed* upon the consumption of event *cl*. The breaker may be faulty in two ways: either it remains closed upon the consumption of *op* event (b_3) or it remains open upon the consumption of *cl* event (b_4). Finally, in transitions b_5 and b_6 the breaker just consumes an event (either *cl* or *op*, respectively) which makes the breaker to remain in the same state. Based on Fig. 1, active system $\mathcal{W}$ consists of protection device p and breakers l and r. The behavior space of $\mathcal{W}$ is outlined in Fig. 3, where arcs are marked by indices of component transitions.[4] It includes 57 states, with each state B being identified in range [0..56], where 0 is the initial state $\mathcal{W}_0$, while final states are 0, 6, 7, 8, 9, 15, 16, and 17. The actual content of each state B is outlined in Table III. It consists of two parts: a triple $(S(l), S(p), S(r))$ including states for breaker l, protection device p, and breaker r, respectively, and a pair $(Q(l), Q(r))$ representing the queues of events within links from p to l and from p to r, respectively. We assume that each link can store at most one event.[5]

[4]A transition of the protection device is indicated by a single digit (its number). A transition of a breaker are indicated by two characters: the number of the transition and the identifier of the breaker (either l or r). For example, 1 stands for p_1, 2l stands for $b_2(l)$, and 3r stands for $b_3(r)$.

[5]This means that a link can be either empty or store just one event.

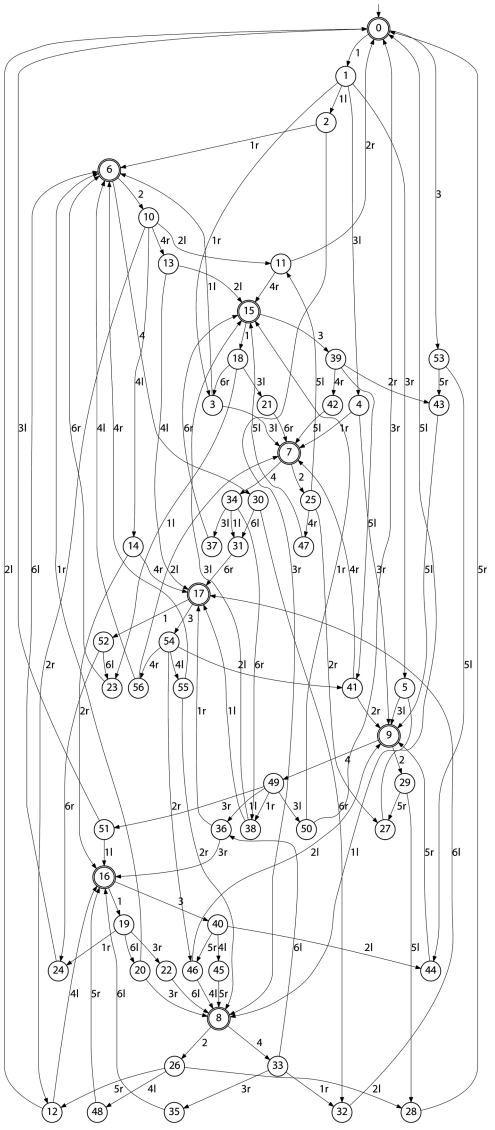

Fig. 3. Behavior space $Bsp(\mathcal{W}, \mathcal{W}_0)$ (node details are in Table III).

TABLE III
STATE DETAILS FOR $Bsp(\mathcal{W}, \mathcal{W}_0)$ OUTLINED IN FIG. 3

	Components			*Links*	
B	$S(l)$	$S(p)$	$S(r)$	$Q(l)$	$Q(r)$
0	*closed*	*idle*	*closed*	–	–
1	*closed*	*awaken*	*closed*	*op*	*op*
2	*open*	*awaken*	*closed*	–	*op*
3	*closed*	*awaken*	*open*	*op*	–
4	*closed*	*awaken*	*closed*	–	*op*
5	*closed*	*awaken*	*closed*	*op*	–
6	*open*	*awaken*	*open*	–	–
7	*closed*	*awaken*	*open*	–	–
8	*open*	*awaken*	*closed*	–	–
9	*closed*	*awaken*	*closed*	–	–
10	*open*	*idle*	*open*	*cl*	*cl*
11	*closed*	*idle*	*open*	–	*cl*
12	*open*	*idle*	*closed*	*cl*	–
13	*open*	*idle*	*open*	*cl*	–
14	*open*	*idle*	*open*	–	*cl*
15	*closed*	*idle*	*open*	–	–
16	*open*	*idle*	*closed*	–	–
17	*open*	*idle*	*open*	–	–
18	*closed*	*awaken*	*open*	*op*	*op*
19	*open*	*awaken*	*closed*	*op*	*op*
20	*open*	*awaken*	*closed*	–	*op*
21	*closed*	*awaken*	*open*	–	*op*
22	*open*	*awaken*	*closed*	*op*	–
23	*open*	*awaken*	*open*	–	*op*
24	*open*	*awaken*	*open*	*op*	–
25	*closed*	*idle*	*open*	*cl*	*cl*
26	*open*	*idle*	*closed*	*cl*	*cl*
27	*closed*	*idle*	*closed*	*cl*	–
28	*closed*	*idle*	*closed*	–	*cl*
29	*closed*	*idle*	*closed*	*cl*	*cl*
30	*open*	*idle*	*open*	*op*	*op*
31	*open*	*idle*	*open*	–	*op*
32	*open*	*idle*	*open*	*op*	–
33	*open*	*idle*	*closed*	*op*	*op*
34	*closed*	*idle*	*open*	*op*	*op*
35	*open*	*idle*	*closed*	*op*	–
36	*open*	*idle*	*closed*	–	*op*
37	*closed*	*idle*	*open*	–	*op*
38	*closed*	*idle*	*open*	*op*	–
39	*closed*	*awaken*	*open*	*cl*	*cl*
40	*open*	*awaken*	*closed*	*cl*	*cl*
41	*closed*	*awaken*	*open*	–	*cl*
42	*closed*	*awaken*	*open*	*cl*	–
43	*closed*	*awaken*	*closed*	*cl*	–
44	*closed*	*awaken*	*closed*	–	*cl*
45	*open*	*awaken*	*closed*	–	*cl*
46	*open*	*awaken*	*closed*	*cl*	–
47	*closed*	*idle*	*open*	*cl*	–
48	*open*	*idle*	*closed*	–	*cl*
49	*closed*	*idle*	*closed*	*op*	*op*
50	*closed*	*idle*	*closed*	–	*op*
51	*closed*	*idle*	*closed*	*op*	–
52	*open*	*awaken*	*open*	*op*	*op*
53	*closed*	*awaken*	*closed*	*cl*	*cl*
54	*open*	*awaken*	*open*	*cl*	*cl*
55	*open*	*awaken*	*open*	–	*cl*
56	*open*	*awaken*	*open*	*cl*	–

IV. DIAGNOSIS PROBLEM

When reacting, an active system performs a sequence of transitions (history), that moves the system from the initial state to a final state. Since a number of such transitions are perceived by the observer as visible labels, such a history generates a sequence of labels, called the trace of the history. A diagnosis problem $\wp$ for a system Σ is a quadruple

$$\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R}) \tag{1}$$

where Σ_0 is the initial state of the system, $\mathcal{V}$ the viewer, $\mathcal{O}$ the observation, and $\mathcal{R}$ the ruler, as detailed below.

A. Viewer and Observation

The viewer maps each component transition to a label, thereby establishing how transitions are perceived. If the label is ε (null label) the transition is invisible, otherwise it is visible. The observation is a directed acyclic graph where nodes are marked by candidate labels and arcs denote partial ordering between nodes. For each node, only one candidate label is the one associated with a component transition by the viewer. This is the actual label, which, however, is unknown to the observer.

Example 4: With reference to system $\mathcal{W}$ defined in Example 3 and models in Fig. 2, we assume a viewer $\mathcal{V}$ with the following visible transitions and relevant labels: b_{1l}: *opl*, b_{1r}: *opr*, b_{2l}: *cll*, b_{2r}: *clr*, p_1: *awk*, p_2: *ide*, p_3: *awk*, p_4: *ide*. In the left side of Fig. 4 is an observation $\mathcal{O}$ relevant to viewer

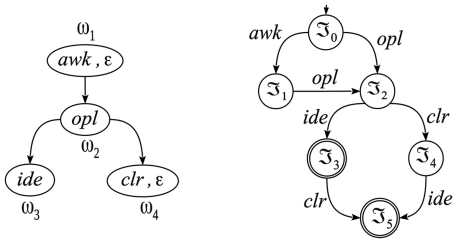

Fig. 4. Observation $\mathcal{O}$ (left). Relevant index space $Isp(\mathcal{O})$ (right).

$\mathcal{V}$, which is composed of four nodes, namely $\omega_1 \cdots \omega_4$, and three arcs, namely, (ω_1, ω_2), (ω_2, ω_3), and (ω_2, ω_4).

An observation implicitly embodies several candidate traces, denoted $\|\mathcal{O}\|$, with each one being obtained by choosing one label from each node of the observation based on partial ordering between arcs. Among such candidates is the (unknown) actual trace generated by the system history, the one involving actual labels only. For practical reasons, an *index space* of $\mathcal{O}$ is generated [17], namely, $Isp(\mathcal{O})$. This is a deterministic automaton, where arcs are marked by the visible labels of $\mathcal{O}$ (ε aside). The regular language of $Isp(\mathcal{O})$ equals $\|\mathcal{O}\|$, that is, the set of paths in $Isp(\mathcal{O})$ equals the set of candidate traces of $\mathcal{O}$.

Example 5: Shown in the right side of Fig. 4 is the index space of $\mathcal{O}$, where $\Im_3$ and $\Im_5$ are the final states. Notice how the language of $Isp(\mathcal{O})$ equals the set of candidate traces of $\mathcal{O}$, where each candidate is generated by choosing from each node of $\mathcal{O}$ one candidate label such that partial ordering is fulfilled. For example, we can choose from the sequence of nodes ω_1, ω_2, ω_4, and ω_3 labels *awk*, *opl*, ε, and *ide*, giving rise to string [*awk*, *opl*, *ide*] (being invisible, ε does not count). This string belongs also to the language of $Isp(\mathcal{O})$, as it can be generated by a path from the initial state $\Im_0$ to final state $\Im_3$, namely, $[\Im_0, \Im_1, \Im_2, \Im_3]$.

B. Ruler

The ruler $\mathcal{R}$ is a triple $(\Xi, \mathcal{H}, \mathcal{S})$, where Ξ is the *context domain*, $\mathcal{H}$ the *context hierarchy*, and $\mathcal{S}$ the *semantics*. These three elements are defined as follows.

1) The context domain $\Xi = \{\xi_1, \ldots, \xi_n\}$ is the set of subsystems of Σ, called *contexts*, that are designated to be relevant to the output of the diagnosis task.
2) The context hierarchy $\mathcal{H}$ defines a partition of each context $\xi_i \in \Xi$, $i \in [1..n]$, in terms of other sub-contexts (or components), namely, $(\xi_i, \{\xi_{i_1}, \ldots, \xi_{i_{n_i}}\}) \in \mathcal{H}$ (where $\xi_{i_1}, \ldots, \xi_{i_{n_i}}$ are the child nodes of ξ_i in $\mathcal{H}$); as such $\mathcal{H}$ implicitly defines a tree (more generally, a forest), where leaf nodes are components.
3) Let $\mathbf{N}$ be a set of identifiers called the *name space*, $\mathbf{F}$ a subset of $\mathbf{N}$ called the *fault space*, and $\mathbf{I}$ a subset of $\mathbf{N}$ called the *interface space*. The semantics $\mathcal{S}$ is a set of pairs $(\xi, \mathcal{P})$, where $\xi \in \Xi$ and $\mathcal{P} = [P_1, \ldots, P_k]$ is the list of *semantic patterns*, with each semantic pattern $P_j \in \mathcal{P}$, $P_j = (N_j, E_j)$, being the association between a name in $N_j \in \mathbf{N}$ and a regular expression E_j.

The alphabet of the regular expression E_j is inductively defined as follows.

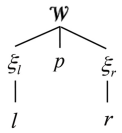

Fig. 5. Context hierarchy for protection hardware $\mathcal{W}$.

1) The alphabet of a component is the set of transitions of its model.
2) Let $\mathbb{N}$ be the set of names used for semantic patterns in $\bar{\mathcal{P}}$, where $(\bar{\xi}, \bar{\mathcal{P}}) \in \mathcal{S}$; let $\mathbb{I}$ be the subset of names in $\mathbb{N}$ that are also in the interface space, namely, $\mathbb{I} = \mathbb{N} \cap \mathbf{I}$; each symbol in the alphabet $\mathcal{A}$ of $\bar{\xi}$ is a subset of $\mathbb{I}$, called an *interface symbol*, in other words, $\mathcal{A}(\bar{\xi})$ is the powerset of $\mathbb{I}$, namely, $\mathcal{A}(\bar{\xi}) = 2^{\mathbb{I}}$.
3) Let $(\xi, \{\xi_1, \ldots, \xi_m\}) \in \mathcal{H}$, $(\xi, \mathcal{P}) \in \mathcal{S}$, $P_j \in \mathcal{P}$, $P_j = (N_j, E_j)$; the alphabet $\mathcal{A}$ of E_j is the union of the alphabets of the subcontexts (possibly components) of ξ and the semantic-pattern names defined up to $P_{j-1} \in \mathcal{P}$

$$\mathcal{A}(E_j) = \left(\bigcup_{i=1}^{m} \mathcal{A}(\xi_i)\right) \cup \left(\bigcup_{i=1}^{j-1} \{N_i\}\right). \tag{2}$$

The *plain form* of the regular expression E_j is the iterated macrosubstitution of each name in $\mathbf{N}$ (involved in E_j) by the corresponding regular expression.

The syntax of the regular expression on alphabet $\mathcal{A}$ is defined inductively as follows (assuming x and y to be regular expressions denoting languages $L(x)$ and $L(y)$, respectively):

1) ε denotes the language $\{\varepsilon\}$ (where ε is the *null* symbol);
2) if $a \in \mathcal{A}$ then a denotes the singleton language $\{a\}$;
3) (x) denotes $L(x)$ (parentheses are allowed as usual);
4) $x^?$ denotes $L(x) \cup \{\varepsilon\}$ (optionality);
5) x^* denotes $\bigcup_{i=0}^{\infty}(L(x))^i$ (iteration zero or more times);
6) x^+ denotes $\bigcup_{i=1}^{\infty}(L(x))^i$ (iteration one or more times);
7) xy denotes $L(x)L(y)$ (concatenation);
8) $x \mid y$ denotes $L(x) \cup L(y)$ (alternative);
9) $x \,\&\, y$ is a shorthand for $(xy \mid yx)$ (free concatenation).

We assume that the plain form of each regular expression is nonempty.

Example 6: A ruler $\mathcal{R}$ for protection hardware $\mathcal{W}$ defined in Example 3 is specified as follows. Context domain: $\Xi = \{\xi_l, \xi_r, \xi_{\mathcal{W}}\}$, where both ξ_l and ξ_r include just breakers l and r, respectively. The context hierarchy $\mathcal{H}$ is outlined in Fig. 5. $\mathbf{N} = \mathbf{F} = \{nol, ncl, nor, ncr, fop, fcp, fdw, fcw\}$ (explanation is in Table IV). $\mathbf{I} = \{nol, ncl, nor, ncr\}$. $\mathcal{S}$ is defined in Table IV. In the specification, bold symbols denote singletons: for instance, **nol** is a shorthand for the interface symbol $\{nol\}$ (in our example, all interface symbols are singletons).

For breakers l and r, two semantic patterns are defined, involving one transition, corresponding to failure either to open or to close. For protection hardware $\mathcal{W}$, four semantic patterns are defined. The first two patterns, *fop* and *fcp*, are relevant to the misbehavior of the protection device (by sending breakers the wrong command). Semantic pattern *fdw* indicates a failure in disconnecting the left-hand-side of the line: either the protection device sends the wrong command to breakers (*fop*) or it sends the correct command (p_1), yet

TABLE IV
SEMANTIC PATTERNS

Context	*Semantic patterns*	*Explanation*
ξ_l	$nol = b_{3l}$	Breaker l does not open
	$ncl = b_{4l}$	Breaker l does not close
ξ_r	$nor = b_{3r}$	Breaker r does not open
	$ncr = b_{4r}$	Breaker r does not close
$\mathcal{W}$	$fop = p_3$	Prot. device p fails to open
	$fcp = p_4$	Prot. device p fails to close
	$fdw = fop \mid p_1$ **(nol & nor)**	Prot. hw $\mathcal{W}$ fails to disconnect
	$frw = fcp \mid p_2$ **(ncl | ncr)**	Prot. hw $\mathcal{W}$ fails to reconnect

both breakers keep being closed (**nol** & **nor**). Finally, semantic pattern *frw* indicates a failure in reconnecting the left-hand-side of the line: either the protection device sends the wrong command to breakers (*fcp*) or it sends the correct command (p_2), yet either l or r keeps being open (**ncl** | **ncr**).

V. HISTORY PROJECTION

As a consequence of context-sensitivity, in order to define a diagnosis we need to introduce the notion of history projection. Intuitively, the projection of a history h on a context ξ, written $h[\xi]$, is the mode in which h is perceived by ξ in terms of interface symbols (if ξ includes other contexts) and/or component transitions (if ξ includes components).

If ξ includes components only, $h[\xi]$ is simply the subsequence of transitions in h which belong to such components. Instead, when ξ embodies other contexts, $h[\xi]$ is bound to include interface symbols of such contexts. Each of these symbols corresponds to one or more matchings of relevant semantic patterns.

We provide the precise definition of $h[\xi]$ in operational terms. Let $\xi_1, \ldots, \xi_n$ be the set of contexts involved (either directly or indirectly) in the context hierarchy rooted in ξ (including the latter). Let $\mathbb{C}_\xi$ denote the set of components involved in that hierarchy. The operational specification, called projection, generates all projections $h[\xi_1], \ldots, h[\xi_n]$ in one run as detailed below.

1. **procedure** *Projection*(h, ξ)
2. **input**
3. h: a history for a system Σ,
4. ξ: a context in $\mathcal{H}$ (context hierarchy of Σ);
5. **output**
6. The history projections $h[\xi_1], \ldots, h[\xi_n]$ for ξ and all descendant contexts of ξ;
7. **begin** ⟨Projection⟩
8. Initialize each $h[\xi_i]$, $i \in [1..n]$, to the empty sequence;
9. **foreach** $T \in h$ such that $C_T \in \mathbb{C}_\xi$ **do**
10. Let ξ_p be the parent context of C_T in the hierarchy;
11. Shift T to $h[\xi_p]$;
12. **repeat**
13. Let $\mathbb{I}$ be the set of symbols $N \in \mathbf{I}$ (interface) such that (N, E) is a semantic pattern for ξ_p **and** a suffix of $h[\xi_p]$ matches E (in plain form);
14. **if** $\mathbb{I} \neq \emptyset$ **and** $\xi_p \neq \xi$ **and** ξ_p has a parent ξ_p^* **then**
15. Append $\mathbb{I}$ to $h[\xi_p^*]$;
16. $\xi_p := \xi_p^*$
17. **endif**
18. **until** $\mathbb{I} = \emptyset$ **or** $\xi_p = \xi$
19. **endfor**
20. **end** ⟨Projection⟩.

TABLE V
TRACING OF Projection$(h, \mathcal{R})$

i	h	$h[\xi_l]$	$h[\xi_r]$	$h[\mathcal{W}]$
0	$p_1, b_{3r}, b_{3l}, p_4, b_{3r}, b_{1l}$			
1	$b_{3r}, b_{3l}, p_4, b_{3r}, b_{1l}$			p_1
2	$b_{3l}, p_4, b_{3r}, b_{1l}$		b_{3r}	p_1, **nor**
3	p_4, b_{3r}, b_{1l}	b_{3l}	b_{3r}	p_1, **nor**, **nol**
4	b_{3r}, b_{1l}	b_{3l}	b_{3r}	p_1, **nor**, **nol**, p_4
5	b_{1l}	b_{3l}	b_{3r}, b_{3r}	p_1, **nor**, **nol**, p_4, **nor**
6		b_{3l}, b_{1l}	b_{3r}, b_{3r}	p_1, **nor**, **nol**, p_4, **nor**

Example 7: Based on $Bsp(\Sigma, \Sigma_0)$ outlined in Fig. 3, consider history $h = [p_1, b_{3r}, b_{3l}, p_4, b_{3r}, b_{1l}]$. With reference to the context hierarchy displayed in Fig. 5, traced in Table V is the computation of $h[\mathcal{W}]$ by procedure *Projection*. For each transition considered in the main loop, identified by $i \in [1..6]$, configurations of h, $h[\xi_l]$, $h[\xi_r]$, and $h[\mathcal{W}]$ are represented as follows:

0) initialization (line 8);
1) p_1 shifted to $h[\mathcal{W}]$; $\mathbb{I} = \emptyset$;
2) b_{3r} shifted to $h[\xi_r]$; $\mathbb{I} = \{nor\}$, **nor** appended to $h[\mathcal{W}]$;
3) b_{3l} is shifted to $h[\xi_l]$; $\mathbb{I} = \{nol\}$, **nol** appended to $h[\mathcal{W}]$;
4) p_4 shifted to $h[\mathcal{W}]$; $\mathbb{I} = \emptyset$;
5) b_{3r} shifted to $h[\xi_r]$; $\mathbb{I} = \{nor\}$, **nor** appended to $h[\mathcal{W}]$;
6) b_{1l} shifted to $h[\xi_l]$; $\mathbb{I} = \emptyset$.

The configurations of the last line are the actual projections of history h on clusters ξ_l, ξ_r, and $\mathcal{W}$. In particular, we have $h[\mathcal{W}] = [p_1, \mathbf{nor}, \mathbf{nol}, p_4, \mathbf{nor}]$. Instead, as expected, the projections on clusters ξ_l and ξ_r include the subsequence of transitions relevant to components l and r, respectively.

VI. PROBLEM SOLUTION

We now provide the definition of solution of a diagnosis problem. The actual mode in which the solution is generated by the diagnosis engine is described in Section VIII.

A. Diagnosis

First, we need to introduce the notion of a (candidate) diagnosis. Let $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ be a diagnosis problem. Assume ruler $\mathcal{R} = (\Xi, \mathcal{H}, \mathcal{S})$ and $\|E\|$ denoting the language of (the plain form of) a regular expression E. The diagnosis of a history h based on ruler $\mathcal{R}$, written $h_{\{\mathcal{R}\}}$, is the set of fault symbols defined as follows:

$$h_{\{\mathcal{R}\}} = \{F | F \in \mathbf{F},\ (F, E) \in \mathcal{P}, (\xi, \mathcal{P}) \in \mathcal{S}, \\ e \in \|E\|,\ e \sqsubseteq h[\xi]\}. \quad (3)$$

In other words, the diagnosis $h_{\{\mathcal{R}\}}$ is the set of faults F involved in semantic-pattern list $\mathcal{P}$ of a context ξ, such that there exists a string e, in the language of the (plain form of) regular expression E relevant to F, that is a (contiguous) substring ($\sqsubseteq$) of the projection of history h on ξ.

TABLE VI
HISTORIES, PROJECTIONS, AND DIAGNOSES

h	$h[\xi_l]$	$h[\xi_r]$	$h[\mathcal{W}]$	$h_{\{\mathcal{R}\}}$
$h_1 \cdots h_4$	b_{1l}, b_{4l}	b_{3r}, b_{5r}	p_1, **nor**, p_2, **ncl**	{*nor*, *ncl*, *frw*}
h_5	b_{1l}, b_{6l}	b_{3r}, b_{3r}	p_1, **nor**, p_4, **nor**	{*nor*, *fcp*, *frw*}

Example 8: Let $\wp(\mathcal{W}) = (\mathcal{W}_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ be the diagnosis problem for protection hardware $\mathcal{W}$ defined in Example 3, where viewer $\mathcal{V}$ and observation $\mathcal{O}$ are defined in Example 4, while ruler $\mathcal{R}$ is specified in Example 6. The behavior space $Bsp(\mathcal{W}, \mathcal{W}_0)$ is outlined in Fig. 3, with $\mathcal{W}_0 = 0$ being the initial state. Consider history $h = [p_1, b_{3r}, b_{3l}, p_4, b_{3r}, b_{1l}]$ defined in Example 7. According to (3) and based on the semantic patterns specified in Table IV, to determine the diagnosis of h based on $\mathcal{R}$, we need to perform pattern matching on the projections of h on contexts ξ_l, ξ_r, and $\mathcal{W}$.

Considering $h[\xi_l] = [b_{3l}, b_{1l}]$, since $[b_{3l}] \sqsubseteq h[\xi_l]$, we have $nol \in h_{\{\mathcal{R}\}}$. For $h[\xi_r] = [b_{3r}, b_{3r}]$, since $[b_{3r}] \sqsubseteq h[\xi_r]$, we have $nor \in h_{\{\mathcal{R}\}}$. Finally, for $h[\mathcal{W}] = [p_1, \mathbf{nor}, \mathbf{nol}, p_4, \mathbf{nor}]$, since $[p_4] \sqsubseteq h[\mathcal{W}]$ and $[p_1, \mathbf{nor}, \mathbf{nol}] \sqsubseteq h[\mathcal{W}]$ (with $[p_1, \mathbf{nor}, \mathbf{nol}]$ being a string in the language of $p_1(\mathbf{nol} \,\&\, \mathbf{nor})$), both fault labels *fcp* and *fdw* are in $h_{\{\mathcal{R}\}}$.

In summary, $h_{\{\mathcal{R}\}} = \{nol, nor, fcp, fdw\}$, meaning that both breakers fail to open, the protection device sends the wrong command (open) to breakers, and the protection hardware fails to disconnect the left-hand side of the line.

B. Solution

The solution of a diagnosis problem is a set of candidate diagnoses, where each candidate is a (possibly empty) set of fault symbols. By denoting the trace of a history h (the sublist of visible transitions in h) based on viewer $\mathcal{V}$ with $h_{[\mathcal{V}]}$, we can now define the solution of a diagnosis problem. The solution of the diagnosis problem $\wp(\Sigma)$, written $\Delta(\wp(\Sigma))$, is the set of candidate diagnoses of the histories of the behavior space $Bsp(\Sigma, \Sigma_0)$, based on ruler $\mathcal{R}$, that are consistent with observation $\mathcal{O}$

$$\Delta(\wp(\Sigma)) = \{\, h_{\{\mathcal{R}\}} | h \in Bsp(\Sigma, \Sigma_0), h_{[\mathcal{V}]} \in \|\mathcal{O}\| \,\}. \quad (4)$$

In other words, the solution of a diagnosis problem $\wp(\Sigma)$ is the set of diagnoses of histories h of Σ whose trace is a candidate trace in $\mathcal{O}$.

Example 9: Let $\wp(\mathcal{W}) = (\mathcal{W}_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ be the diagnosis problem for the protection hardware $\mathcal{W}$ defined in Example 3, where viewer $\mathcal{V}$ and observation $\mathcal{O}$ are defined in Example 4, while ruler $\mathcal{R}$ is specified in Example 6. The behavior space $Bsp(\mathcal{W}, \mathcal{W}_0)$ is outlined in Fig. 3. The solution of $\wp(\mathcal{W})$ can be determined based on (4). First, we have to select the histories in $Bsp(\mathcal{W}, \mathcal{W}_0)$ whose trace is in $\|\mathcal{O}\|$, in other words, whose trace is a path in the index space of $\mathcal{O}$ (Fig. 4). Among the six candidate traces in $Isp(\mathcal{O})$, generated by paths from the initial state $\Im_0$ to a final state (either $\Im_3$ or $\Im_5$), only one trace, namely $[awk, opl, ide]$, is consistent with $Bsp(\mathcal{W}, \mathcal{W}_0)$. This single trace is generated by five different histories: 1) $h_1 = [p_1, b_{3r}, b_{1l}, p_2, b_{4l}, b_{5r}]$; 2) $h_2 = [p_1, b_{3r}, b_{1l}, p_2, b_{5r}, b_{4l}]$; 3) $h_3 = [p_1, b_{1l}, b_{3r}, p_2, b_{4l}, b_{5r}]$; 4) $h_4 = [p_1, b_{1l}, b_{3r}, p_2, b_{5r}, b_{4l}]$; and 5) $h_5 = [p_1, b_{1l}, b_{3r}, p_4, b_{3r}, b_{6l}]$. According to the context hierarchy outlined in Fig. 5, the projections on such histories on contexts ξ_l, ξ_r, and $\xi_{\mathcal{W}}$ can be computed as in Example 7, giving rise to the results displayed in Table VI. Since histories $h_1 \cdots h_4$ share the same projections, the corresponding diagnosis (based on ruler $\mathcal{R}$) is the same, namely, $\delta_1 = h_{1\{\mathcal{R}\}} = \{nor, ncl, frw\}$. Instead, $\delta_2 = h_{5\{\mathcal{R}\}} = \{nor, fcp, frw\}$. Consequently, the solution of the diagnosis problem includes two candidate diagnoses, specifically, $\Delta(\wp(\mathcal{W})) = \{\delta_1, \delta_2\}$, corresponding to these two scenarios:

1) δ_1: Breaker r fails to open, breaker l fails to close, and protection hardware $\mathcal{W}$ fails to reconnect the line;
2) δ_2: Breaker r fails to open, protection device sends the wrong command to breakers, and $\mathcal{W}$ fails to reconnect the line.

Albeit the solution includes two candidate diagnoses, since $\delta_1 \cap \delta_2 = \{nor, frw\}$, certainly r fails to open and $\mathcal{W}$ fails to reconnect the line.

VII. SEMANTIC SPACE

In the diagnosis process a distinction is made between online and offline tasks. An offline task is accomplished before the system becomes operating.[6] By contrast, an online task is performed while the system is operating, typically under stringent time constraints. For instance, the modeling of the system is an offline task, while the task performed by the diagnosis engine, which is in charge of solving the diagnosis problem, is carried out online.

Since diagnosing active systems (and, more generally, DESs) is computational expensive, it is convenient to maximize the amount of processing performed offline [32], [33]. This provides the following two advantages.

1) Specific properties of the system, such as diagnosability, can be checked (and possibly changed) before the system becomes operating.
2) The speed of the diagnosis engine increases, as what is performed offline is not to be performed online.

Typically, within the quadruple defining a diagnosis problem, only the temporal observation $\mathcal{O}$ is available on-line, while the viewer and the ruler are defined at system-modeling time (offline). In particular, the semantic patterns can be analyzed and compiled into data structures which allow for better (online) performances of the diagnosis engine.

As known, any regular expression can be transformed into an equivalent deterministic finite automaton [34].

The compilation process involves the following steps.

1) For each semantic pattern (F_i, E_i) such that $F_i \in \mathbf{F}$, E_i is unfolded into its plain form and an equivalent deterministic automaton A_i is generated, where final states are marked by label F_i.
2) For each semantic pattern (I_j, E_j) such that $I_j \in (\mathbf{I} - \mathbf{F})$, E_j is unfolded into its plain form and an equivalent

[6]Like in compilers, where tasks such as type-checking are performed before the software becomes operating (at compile-time rather than at runtime).

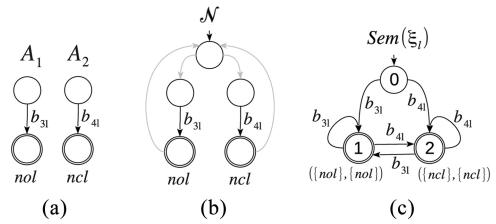

Fig. 6. Compilation of semantic space for breaker l. Deterministic automata A_1 and A_2 (a). Nondeterministic automaton $\mathcal{N}$ (b). Semantic space $Sem(\xi_l)$ (c).

deterministic automaton A_j is generated, where final states are marked by label I_j.

3) The so-generated deterministic automata $A_1, \ldots, A_n$ within the same list of semantic patterns $\mathcal{P}$ (and, as such, relevant to the same context ξ) are merged to yield a new deterministic automaton called the *semantic space* of ξ, $Sem(\xi)$, where each final state S is marked by a pair $(\mathbb{I}, \mathbb{F})$, where $\mathbb{I}$ and $\mathbb{F}$ are the set of interface labels and the set of fault labels, respectively, that are associated with the states identifying S.[7]

The merging of automata in step 3 is performed as follows.

1) A nondeterministic automaton $\mathcal{N}$ is created by generating its initial state S_0 and one empty transition from S_0 to each initial state of A_i, $i \in [1..n]$.
2) In each A_i, $i \in [1..n]$, an empty transition from each noninitial state to S_0 is inserted.
3) $\mathcal{N}$ is determinized, thereby obtaining $Sem(\xi)$, with each final state S of $Sem(\xi)$ being marked by pair $(\mathbb{I}, \mathbb{F})$, where $\mathbb{I}$ and $\mathbb{F}$ are the interface labels and the fault labels, respectively, obtained by the union of the interface labels and fault labels, respectively, that are associated with states in S that are final in the corresponding A_i.

The rationale of the sequence of steps above is that, during the reconstruction of the system behavior, the diagnosis engine is supposed to uncover the matching of several (possibly overlapping) semantic patterns. This means that after the matching of any transition, the same semantic pattern, or even a different one, may possibly start.

Example 10: With reference to ruler $\mathcal{R}$ specified in Example 6, shown in Fig. 6 is the generation of semantic space $Sem(\xi_l)$ for the context involving breaker l. Based on Table IV, semantic patterns relevant to fault labels *nol* and *ncl* are defined by single transitions b_{3l} and b_{4l}, respectively. For step 1 of the compilation process, the corresponding deterministic automata A_1 and A_2 are represented in Fig. 6(a), with each one being composed of two states (initial and final) and one transition, which is marked by b_{3l} and b_{4l}, respectively. Step 2 is not applicable, as fault labels *nol* and *ncl* are also interface labels. For step 3, the nondeterministic automaton $\mathcal{N}$ is outlined in Fig. 6(b), where empty transitions are represented by gray arcs. Finally, the actual semantic space $Sem(\xi_l)$ is outlined in Fig. 6(c), where each final state is marked by a pair $(\mathbb{I}, \mathbb{F})$, with (incidentally) $\mathbb{I} = \mathbb{F}$.

[7] A state of the determinized automaton is identified by a subset of the states of the corresponding nondeterministic automaton [34].

TABLE VII
TRANSITION FUNCTION OF SEMANTIC SPACE $Sem(\mathcal{W})$

State	p_1	p_2	p_3	p_4	**nol**	**nor**	**ncl**	**ncr**	$\mathbb{F}$
0	1	2	3	4					
1	1	2	3	4	5	6			
2	1	2	3	4			7	7	
3	1	2	3	4					{*fop*, *fdw*}
4	1	2	3	4					{*fcp*, *frw*}
5	1	2	3	4		8			
6	1	2	3	4	8				
7	1	2	3	4					{*frw*}
8	1	2	3	4					{*fdw*}

The generation of the semantic space $Sem(\xi_r)$ for breaker r gives rise to an identical automaton, where labels b_{3l}, b_{4l}, *nol*, and *ncl* are replaced by b_{3r}, b_{4r}, *nor*, and *ncr*, respectively.

The generation of the semantic space for protection hardware $\mathcal{W}$ is carried out in a similar way, based on the semantic patterns specified in the bottom section of Table IV. For space reasons, we only report in Table VII the tabular representation of the resulting automaton $Sem(\mathcal{W})$. Each row defines the transition function for states $0, 1, \ldots, 8$, where $3, 4, 7, 8$ are final. According to the semantic patterns defined in Table IV, the alphabet of $Sem(\mathcal{W})$ is $\{p_1, p_2, p_3, p_4, \mathbf{nol}, \mathbf{nor}, \mathbf{ncl}, \mathbf{ncr}\}$. If defined, the state reached by pair (S, a), where S is a state and a a symbol in the alphabet, is indicated in the cell corresponding to row S and symbol a; for instance, the state reached by (2, **ncl**) is 7. The last column outlines for each final state the associated set of fault symbols; for instance, state 4 is marked by set {*fcp*, *frw*} of fault symbols.

Proposition 1: No transition enters the initial state of a semantic space.

Proof: In the last step of the construction of a semantic space, the deterministic automata $A_1, \ldots, A_n$ are merged by creating the initial state S_0, and by inserting an empty transition from S_0 to the initial state of each A_i, $i \in [1..n]$, and from each noninitial state of each A_i to S_0. As such, S_0 is entered by empty-transitions only. Based on the algorithm for determinization of finite automata [34], starting from the initial state S_0^d of the deterministic automaton, obtained as the ε-closure of S_0, each new state is generated as the ε-closure of a nonempty subset of states of the nondeterministic automaton. Hence, to generate S_0^d, we need a nonempty transition entering S_0, which is not the case. ■

Proposition 2: If s is a string in the language of semantic space $Sem(\xi)$, generated by a path from the initial state to a final state S_f, then the interface symbol associated with S_f is the set of symbols $N \in \mathbf{I}$ such that (N, E) is a semantic pattern relevant to ξ, and the language of E (in plain form) includes a string e which is a suffix of s.

Proof: The specification of $Sem(\xi)$ requires three steps. In steps 1 and 2, a deterministic automaton A_i, $i \in [1..n]$, is generated for each semantic pattern (N_i, E_i) relevant to ξ, where $N_i \in \mathbf{F} \cup \mathbf{I}$, with each final state of A_i being marked by label N_i. This means that each A_i recognizes the language of E_i (in plain form). Then, in step 3, automata $A_1, \ldots, A_n$ are merged into a nondeterministic automaton $\mathcal{N}$, for which an initial state S_0 is created and connected to the initial state

of each A_i by an empty transition (see example in Fig. 6). Furthermore, each noninitial state of each A_i is connected to S_0 by an empty transition. Based on this construction, each string s in the language of $\mathcal{N}$ is generated by a set $\boldsymbol{\pi}$ of paths from S_0 to a final state. Due to the way in which automata A_i are merged into $\mathcal{N}$, each path in $\boldsymbol{\pi}$ ending at a final state marked by symbol N_i necessarily has a suffix which is a string in the language of A_i, and hence in the language of E_i (in plain form). Eventually, $Sem(\xi)$ is obtained by determinization of $\mathcal{N}$, with the latter sharing the same language of the former, where each final state S_f is marked by a pair $(\mathbb{I}, \mathbb{F})$, with $\mathbb{F} \subseteq \mathbf{F}$ and $\mathbb{I} \subseteq \mathbf{I}$, where, in particular, $\mathbb{I}$ is the set of symbols in $\mathbf{I}$ marking the final states of $\mathcal{N}$ within S_f. Therefore, since $Sem(\xi)$ is the result of the determinization of $\mathcal{N}$, each string s in the language of $Sem(\xi)$ is generated by *one* path π, from the initial state to a final state S_f. In $\mathcal{N}$, the same string s is generated by a set $\boldsymbol{\pi}$ of paths from S_0 to the final states of $\mathcal{N}$ included in S_f. Consequently, the interface symbol associated with S_f is the set of symbols $N \in \mathbf{I}$ such that (N, E) is a semantic pattern relevant to ξ, and the language of E (in plain form) includes a string e which is a suffix of s. ■

Corollary 1: If s is a string in the language of semantic space $Sem(\xi)$, generated by a path from the initial state to a final state S_f, then the set of faults associated with S_f is the set of symbols $N \in \mathbf{F}$ such that (N, E) is a semantic pattern relevant to ξ, and the language of E (in plain form) includes a string e which is a suffix of s.

Proof: By analogy with Proposition 2, as the way the set of symbols in $\mathbf{F}$ is associated with final states of the semantic space is the same as that for the set of symbols in $\mathbf{I}$. ■

VIII. Diagnosis Engine

The diagnosis engine takes as input a diagnosis problem $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ and outputs the solution $\Delta(\wp(\Sigma))$, as defined in Eqn. (4). In doing so, it needs to build the behavior of Σ that conforms with the observation, namely, $Bhv(\wp(\Sigma))$. We assume that the semantic spaces generated offline by preprocessing (see Section VII) be available to the engine. Since pattern recognition is to be performed, states of $Bhv(\wp(\Sigma))$ will incorporate information not only on the observation but also on the semantic spaces to maintain the state of the matching.

Let $\boldsymbol{B}$ denote the set of states of $Bsp(\Sigma, \Sigma_0)$, $\mathfrak{I}$ the set of states of $Isp(\mathcal{O})$, and $\boldsymbol{P} = \boldsymbol{P}_1 \times \cdots \times \boldsymbol{P}_n$, where $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n$ are the set of states of all semantic spaces $Sem(\xi_1), \ldots, Sem(\xi_n)$, respectively. The behavior of $\wp(\Sigma)$

$$Bhv(\wp(\Sigma)) = (\mathbb{S}, \mathbb{T}, S_0, \mathbb{S}_f) \tag{5}$$

is a deterministic automaton such that:

1) $\mathbb{S} \subseteq \boldsymbol{B} \times \boldsymbol{P} \times \mathfrak{I}$ is the set of states;
2) $S_0 = (\Sigma_0, \mathbb{P}_0, \mathfrak{I}_0)$ is the initial state, where $\mathfrak{I}_0$ is the initial state of $Isp(\mathcal{O})$, and $\mathbb{P}_0 = (P_{1_0}, \ldots, P_{n_0})$ is the tuple of the initial states of $Sem(\xi_1), \ldots, Sem(\xi_n)$;
3) $\mathbb{S}_\mathrm{f} = \{(B_\mathrm{f}, \mathbb{P}, \mathfrak{I}_\mathrm{f})\}$ is the set of final states, where B_f is final in $Bsp(\Sigma, \Sigma_0)$ and $\mathfrak{I}_\mathrm{f}$ is final in $Isp(\mathcal{O})$;
4) $\mathbb{T}$ is the transition function: $(B, \mathbb{P}_0, \mathfrak{I}) \xrightarrow{T} (B', \mathbb{P}', \mathfrak{I}') \in \mathbb{T}$, $\mathbb{P} = (P_1, \ldots, P_n)$, $\mathbb{P}' = (P'_1, \ldots, P'_n)$, iff the following conditions hold:
   a) $B \xrightarrow{T} B'$ is a transition in $Bsp(\Sigma, \Sigma_0)$;[8]
   b) If T is invisible then $\mathfrak{I}' = \mathfrak{I}$ else $\mathfrak{I}'$ equals the target state $\bar{\mathfrak{I}}$ of transition $\mathfrak{I} \xrightarrow{\ell} \bar{\mathfrak{I}}$ in $Isp(\mathcal{O})$, where ℓ is the label associated with T in $\mathcal{V}$;
   c) $\mathbb{P}'=(P'_1, \ldots, P'_n)$ is such that, $\forall i \in [1..n]$, P'_i is defined by the following (possibly recursive) rule:

```
if T is in the alphabet of a context ξ_i then
    P'_i := if P_i --T--> P̄_i ∈ Sem(ξ_i) then P̄_i else P_{i_0}
endif;
if the interface symbol 𝕀 marking P'_i is nonempty and
        ξ_i has a parent ξ_j in the hierarchy then
    reapply the rule replacing T with 𝕀 and i with j
  endif
else
  P'_i := P_i
endif.
```

As such, each state of $Bhv(\wp(\Sigma))$ is a triple involving a state of the behavior space, a state of the index space of $\mathcal{O}$, and a tuple of semantic-space states. A transition marked by T is defined in $Bhv(\wp(\Sigma))$ if a transition marked by T is defined between the corresponding states of the behavior space. The index $\mathfrak{I}'$ of the new state differs from the index $\mathfrak{I}$ in the old state only if T is visible (according to viewer $\mathcal{V}$). Finally, $\mathbb{P}'$ is obtained from $\mathbb{P}$ by performing a state change in the semantic space of the context ξ_i including the component relevant to T: if the new state is marked by a nonempty interface symbol, a state change is possibly propagated to the ancestors of ξ_i.

A. Behavior Building

In this subsection we provide a pseudo-coded algorithm (listed below), called *Build*, for the construction of the behavior $Bhv(\wp(\Sigma))$, as defined above. The algorithm takes as input a diagnosis problem $\wp(\Sigma)$ and two additional data structures: $Isp(\mathcal{O})$, the index space of the observation, and Π, the semantic spaces generated (offline) from $\mathcal{R}$.

Considering the body of build (lines 9–48), the set of states $\mathbb{S}$ and the set of transitions $\mathbb{T}$ are initialized (lines 9–11). Then, a loop is iterated until all states in $\mathbb{S}$ have been processed (lines 12–46). At each iteration, starting from the initial state, an unprocessed state $(B, \mathbb{P}, \mathfrak{I})$ is chosen from $\mathbb{S}$ (line 13) and its transitions are determined (lines 14–44). Each transition T triggerable from B is first checked against the observation, and the new index-space state $\mathfrak{I}'$ is computed (lines 15–22). Then, the new state B' is computed based on B and T (lines 23–26). Finally, the new tuple $\mathbb{P}'$ of semantic-space states is determined (lines 27–40). To this end, the state of the semantic space ξ_i including T in its alphabet is possibly updated. Besides, if the new state P_i is marked by an interface symbol $\mathbb{I} \neq \emptyset$, then the semantic-space change is possibly propagated to the ancestors of ξ_i. Eventually, the set of final states is determined (line 47), and all (spurious) states and transitions,

[8]Since $Bsp(\Sigma, \Sigma_0)$ is assumed to be unavailable, operationally this means that T is triggerable from state B.

which are not within a path from the initial state to a final state, are removed (line 48).

1. **function** $Build(\wp(\Sigma), Isp(\mathcal{O}), \Pi)$
2. **input**
3. $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$: a diagnosis problem,
4. $Isp(\mathcal{O})$: the index space of temporal observation $\mathcal{O}$,
5. $\Pi = (Sem(\xi_1), \ldots, Sem(\xi_n))$: the tuple of semantic spaces by $\mathcal{R}$;
6. **output**
7. The behavior $Bhv(\wp(\Sigma)) = (\mathbb{S}, \mathbb{T}, S_0, \mathbb{S}_f)$;
8. **begin** $\langle Build \rangle$
9. $S_0 := (\Sigma_0, \mathbb{P}_0, \Im_0)$, where $\Im_0$ is the initial state of $Isp(\mathcal{O})$, and $\mathbb{P}_0$ the tuple of initial states of semantic spaces in Π;
10. $\mathbb{S} := \{S_0\}$;
11. $\mathbb{T} := \emptyset$;
12. **repeat**
13. Choose an unmarked state $S = (B, \mathbb{P}, \Im)$ in $\mathbb{S}$;
14. **foreach** component transition T which is triggerable in B **do**
15. Let (T, ℓ) be the association in viewer $\mathcal{V}$;
16. **if** $\ell = \varepsilon$ **then**
17. $\Im' := \Im$
18. **elsif** $\Im \xrightarrow{\ell} \bar{\Im}$ is a transition in $Isp(\mathcal{O})$ **then**
19. $\Im' := \bar{\Im}$
20. **else**
21. **continue**
22. **endif;**
23. $B' := B$;
24. Let C_T be the component relevant to transition T;
25. Set state in B' relevant to C_T with the state reached by T;
26. Insert into links of B' the output events of T;
27. $\mathbb{P}' := \mathbb{P}$;
28. **if** C_T is within the context hierarchy $\mathcal{H}$ **then**
29. Let ξ_i be the context whose alphabet includes symbol T;
30. $\mathbb{I} := T$;
31. **repeat**
32. Let P_i be the state in $\mathbb{P}'$ corresponding to $Sem(\xi_i)$;
33. Let P_{i_0} be the initial state of semantic space $Sem(\xi_i)$;
34. $P_i :=$ **if** $P_i \xrightarrow{\mathbb{I}} \bar{P}_i \in Sem(\xi_i)$ **then** $\bar{P}_i$ **else** P_{i_0} **endif**;
35. Let $\mathbb{I}'$ be the interface symbol marking P_i in $Sem(\xi_i)$;
36. **if** $\mathbb{I}' \neq \emptyset$ **and** ξ_i has a parent ξ_j in $\mathcal{H}$ **then**
37. $\xi_i := \xi_j$; $\mathbb{I} := \mathbb{I}'$
38. **endif**
39. **until** $\mathbb{I}' = \emptyset$ **or** ξ_i has no parent in $\mathcal{H}$
40. **endif**;
41. $S' := (B', \mathbb{P}', \Im')$;
42. **if** $S' \notin \mathbb{S}$ **then** Insert S' into $\mathbb{S}$ **endif**;
43. Insert transition $S \xrightarrow{T} S'$ into $\mathbb{T}$;
44. **endfor**;
45. Mark S
46. **until** all states in $\mathbb{S}$ are marked;
47. $\mathbb{S}_f := \{S_f | S_f \in \mathbb{S}, S_f = (B_f, \mathbb{P}, \Im_f)$, both B_f and $\Im_f$ are final$\}$;
48. Remove from $\mathbb{S}$ and $\mathbb{T}$ all states and transitions which are not connected to a state in $\mathbb{S}_f$
49. **end** $\langle Build \rangle$.

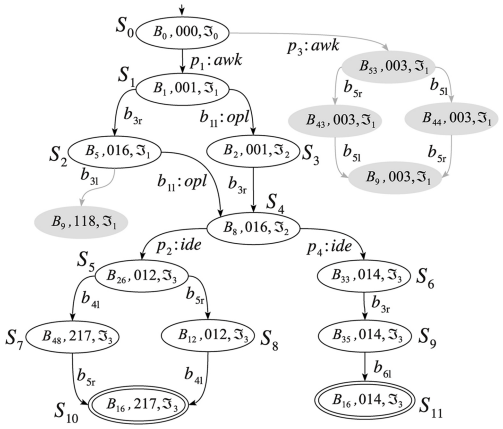

Fig. 7. Behavior $Bhv(\mathcal{P}(\mathcal{W}))$.

Example 11: Consider problem $\wp(\mathcal{W}) = (\mathcal{W}_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ defined in Example 9. Depicted in Fig. 7 is the relevant behavior $Bhv(\wp(\mathcal{W}))$, where observable component-transitions are associated with corresponding viewer labels. According to the definition, the initial state is $S_0 = (B_0, \mathbb{P}_0, \Im_0)$, where $\Im_0$ is the initial state of $Isp(\mathcal{O})$ (Fig. 4), while $\mathbb{P}_0 = (0, 0, 0)$ is the tuple of initial states of semantic spaces $Sem(\xi_l)$, $Sem(\xi_r)$, and $Sem(\mathcal{W})$, respectively. The following conditions hold.

1) $Bsp(\wp(\Sigma))$ includes a transition $B_0 \xrightarrow{p1} B_1$ which is visible via label *awk* of viewer $\mathcal{V}$.
2) A transition $\Im_0 \xrightarrow{awk} \Im_1$ is included in $Isp(\mathcal{O})$.
3) Component transition p_1 is in the alphabet of $Sem(\mathcal{W})$,
4) Transition $0 \xrightarrow{p1} 1$ is included in $Sem(\mathcal{W})$.

A new transition $(B_0, 000, \Im_0) \xrightarrow{p1} (B_1, 001, \Im_1)$ is created in $Bhv(\wp(\mathcal{W}))$. Consider transition $S_5 \xrightarrow{b_{4l}} S_7$, where $\mathbb{P} = (0, 1, 2)$ while $\mathbb{P}' = (2, 1, 7)$. In fact, based on Fig. 6, $Sem(\xi_l)$ includes transition $0 \xrightarrow{b_{4l}} 2$, whose target state 2 is marked by interface symbol *ncl*. Since, based on Table VII, *ncl* is in the alphabet of $Sem(\mathcal{W})$ and a transition $2 \xrightarrow{ncl} 7$ exists in the latter, the semantic-space state relevant to $\mathcal{W}$ is updated too. The construction of the behavior continues until no new node is generated by the application of the transition function. In Fig. 7, final states are S_{10} and S_{11}, as B_{16} is final in $Bsp(\wp(\mathcal{W}))$, and $\Im_3$ is final in $Isp(\mathcal{O})$. The gray part of the graph is inconsistent because it is not encompassed by any path from the initial state to a final state. Hence, it is removed, leaving $Bhv(\wp(\mathcal{W}))$ with twelve states, namely $S_0 \cdots S_{11}$.

B. Behavior Decoration

Once reconstructed the behavior $Bhv(\wp(\Sigma))$, the diagnosis engine is required to generate the problem solution $\Delta(\wp(\Sigma))$

defined in (4) by means of a sound and complete technique. This is carried out by decorating the nodes of the behavior with sets of sets of faults (candidate diagnoses), resulting in the decorated behavior of $\wp(\Sigma)$, denoted $Bhv^*(\wp(\Sigma))$.

To this end, we introduce the notion of the *fault set* of a tuple of semantic-space states $\mathbb{P}$, written $\mathcal{F}(\mathbb{P})$, which is the union of the faults associated with each state of $\mathbb{P}$ in the corresponding semantic space. We denote with $\Delta(S)$ the set of sets of faults decorating state S in $Bhv(\wp(\Sigma))$. Each state in $Bhv(\wp(\Sigma))$ is decorated with a set of sets of faults based on the following two rules:

(1) for the initial state $S_0 = (\Sigma_0, \Im_0, \mathbb{P}_0)$, $\Delta(S_0) = \{\emptyset\}$;
(2) for each $S \xrightarrow{T} S'$ in $Bhv(\wp(\Sigma))$, where $S = (B, \mathbb{P}, \Im)$, $S' = (B', \mathbb{P}', \Im')$, if $\delta \in \Delta(S)$ then $(\delta \cup \mathcal{F}(\mathbb{P}')) \in \Delta(S')$.

Operationally, the decoration algorithm starts by marking the initial state with the singleton $\{\mathcal{F}(\mathbb{P}_0)\} = \{\emptyset\}$ and all other states with $\emptyset$. Then, starting from the decoration of the initial state, it continuously applies the second rule for each transition exiting a state S whose decoration has changed. The rationale of rule (2) is as follows: if the current decoration of a state S of $Bhv(\wp(\Sigma))$ includes the set of faults δ and the reached state S' involves the tuple $\mathbb{P}'$ of semantic-space states with associated faults $\mathcal{F}(\mathbb{P}')$, then the set of faults δ' associated with S' will be the extension of δ by δ', as the latter is the set of faults relevant to the set of semantic patterns recognized in $\mathbb{P}'$. The algorithm stops when the decoration becomes stable (the application of rule (2) will no longer produce further changes).

A pseudo-coded implementation of the decorating algorithm, called decorate, is given below. It makes use of the auxiliary (recursive) procedure *Dec* (lines 6–21). This takes as input a state S of the behavior and a set $\mathcal{D}$ of diagnoses, and propagates $\mathcal{D}$ to the neighboring states of S based on decoration rule (2). In doing so, if new diagnoses $\mathcal{D}^+$ are generated for a neighboring state S', then *Dec* is recursively called on S' and $\mathcal{D}^+$. *Dec* stops when no further diagnosis is generated. Within the body of decorate (lines 23–26), the decorated behavior is initialized as a copy of the behavior (line 23). According to decoration rule (1), the initial state is marked by $\{\emptyset\}$, while all other states are marked by $\emptyset$ (lines 24–25). The actual decoration based on rule (2) is performed by calling $Dec(S_0, \{\emptyset\})$ at line 26.

1. **function** *Decorate*($Bhv(\wp(\Sigma))$)
2. **input**
3. $Bhv(\wp(\Sigma))$: the behavior;
4. **output**
5. $Bhv^*(\wp(\Sigma))$: the decorated behavior;
6. **auxiliary procedure** $Dec(S, \mathcal{D})$
7. **input**
8. S: a state of $Bhv(\wp(\Sigma))$,
9. $\mathcal{D}$: a set of candidate diagnoses;
10. **side effects**
11. Change in decoration of state S is propagated to other states;
12. **begin**⟨Dec⟩
13. **foreach** transition $S \xrightarrow{T} S'$ **do**
14. Let Δ' be the current decoration $\Delta(S')$;
15. Extend the decoration $\Delta(S')$ by $\{\delta' | \delta \in \mathcal{D}, \delta' = \delta \cup \mathcal{F}(\mathbb{P}')\}$;
16. $\mathcal{D}^+ := \Delta(S') - \Delta'$;
17. **if** $\mathcal{D}^+ \neq \emptyset$ **then**
18. $Dec(S', \mathcal{D}^+)$
19. **endif**
20. **endfor**
21. **end**⟨Dec⟩
22. **begin** ⟨Decorate⟩
23. $Bhv^*(\wp(\Sigma)) = Bhv(\wp(\Sigma))$;
24. Mark initial state S_0 with the singleton $\{\emptyset\}$;
25. Mark all other (non initial) states with the empty set $\emptyset$;
26. $Dec(S_0, \{\emptyset\})$
27. **end** ⟨Decorate⟩.

Example 12: Consider the behavior $Bhv(\wp(\mathcal{W}))$ depicted in Fig. 7. Based on the semantic spaces of l and r (Fig. 6), and $\mathcal{W}$ (Table VII), fault sets relevant to states of $Bhv(\wp(\mathcal{W}))$ are outlined in Table VIII (where states with the same set of faults are grouped together). Since, incidentally, $Bhv(\wp(\mathcal{W}))$ is acyclic, the decorations are generated top-down, from the initial state to final states. Initially, S_0 is marked by $\{\emptyset\}$ and all other states by $\emptyset$. Then, the call to $Dec(S_0, \{\emptyset\})$ gives rise to a tree of recursive calls. For instance, a path of the tree from S_0 to S_{10} gives rise to the following stream of computation.

1) $\Delta(S_1)$ is extended to $\{\emptyset\}$.
2) $\Delta(S_2)$ is extended to $\{nor\}$.
3) $\Delta(S_4)$ is extended to $\{\{nor\}\}$.
4) $\Delta(S_5)$ is extended to $\{\{nor\}\}$.
5) $\Delta(S_7)$ is extended to $\{\{nor, ncl, frw\}\}$.
6) $\Delta(S_{10})$ is extended to $\{\{nor, ncl, frw\}\}$.

Eventually, focusing on final states, S_{10} is decorated by $\{\{nor, ncl, frw\}\}$, while S_{11} by $\{\{nor, fcp, frw\}\}$.

TABLE VIII
FAULT SETS RELEVANT TO STATES OF $Bhv(\mathcal{P}(\mathcal{W}))$ (FIG. 7)

$S = (B, \Im, \mathbb{P})$	$\mathcal{F}(\mathbb{P})$
S_0, S_1, S_3	$\emptyset$
S_2, S_4, S_5, S_8	$\{nor\}$
S_7, S_{10}	$\{nor, ncl, frw\}$
S_6, S_9, S_{11}	$\{nor, fcp, frw\}$

C. Solution Generation

Given the decorated behavior $Bhv^*(\wp(\Sigma))$, the solution of the diagnostic problem $\wp(\Sigma)$ can be straightforwardly determined as the union of the decorations of final states

$$\Delta(\wp(\Sigma)) = \{\delta | \delta \in \Delta(S_f), S_f \text{ is final in } Bhv^*(\wp(\Sigma))\}. \quad (6)$$

Example 13: With reference to the behavior $Bhv(\wp(\mathcal{W}))$ in Fig. 7 and the relevant decorations determined in Example 12, the solution of $\wp(\mathcal{W})$ is the union of the decorations of final states S_{10} and S_{11}, namely

$$\Delta(\wp(\mathcal{W})) = \{\{nor, ncl, frw\}, \{nor, fcp, frw\}\}.$$

As a matter of fact, the set of candidate diagnoses equals the solution of the same problem determined in Example 9

based on the formal definition of diagnosis-problem solution stated in (4).

The equality of the solutions for the same diagnosis problem determined in Example 9, based on (4), and Example 13, based on (6), is not a coincidence, as formally proven by Theorem 1 in Section IX.

IX. Correctness

We formally prove the soundness and completeness of the diagnosis technique in determining the solution of a diagnosis problem. Theorem 1 states how the actual solution of $\wp(\Sigma)$ can be distilled from the decorated behavior $Bhv^*(\wp(\Sigma))$.

Theorem 1: Let $Bhv^*(\wp(\Sigma))$ be a decorated behavior. The union of the set of sets of faults decorating the final states of $Bhv^*(\wp(\Sigma))$ equals the solution of $\wp(\Sigma)$.

Proof: Grounded on Lemmas 1–7, where $\mathcal{B}^s$ and $\mathcal{B}^v$ denote $Bsp(\Sigma, \Sigma_0)$ and $Bhv^*(\wp(\Sigma))$, respectively. ■

Lemma 1: If history $h \in \mathcal{B}^v$ then $h \in \mathcal{B}^s$.

Proof: This derives from the fact $\mathcal{B}^v$ differs from $\mathcal{B}^s$ in the additional fields $\Im$ and $\mathbb{P}$, which are irrelevant to the triggering of transitions. By induction on h, starting from the initial state, each new transition applicable in $\mathcal{B}^v$ is applicable in $\mathcal{B}^s$ too. ■

Lemma 2: If history $h \in \mathcal{B}^v$ then $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$.

Proof: Recall that $h_{[\mathcal{V}]}$ is the sequence of observable labels associated with visible transitions in viewer $\mathcal{V}$. Based on the definition of $\mathcal{B}^v$, $h_{[\mathcal{V}]}$ belongs to the language of $Isp(\mathcal{O})$, which equals $\|\mathcal{O}\|$. Thus, $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$. ■

Lemma 3: If history $h \in \mathcal{B}^s$, $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$, then $h \in \mathcal{B}^v$.

Proof: By induction on h, starting from the initial state, each new transition T applicable in $\mathcal{B}^s$ is applicable in $\mathcal{B}^v$ too. In fact, if T is invisible, no further condition is required. If T is visible, based on the assumption $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$ and on the fact that the language of $Isp(\mathcal{O})$ equals $\|\mathcal{O}\|$, the label associated with T in viewer $\mathcal{V}$ matches a transition in $Isp(\mathcal{O})$. ■

Lemma 4: Let π be a path in $\mathcal{B}^v$, from the initial state to $S = (B, \mathbb{P}_S, \Im)$, with $\mathbb{P}_S = (P_1, \ldots, P_n)$. Let h be the sequence of transitions marking arcs in π. Let $S_i = (B_i, \mathbb{P}_{S_i}, \Im_i)$, $i \in [1..n]$, be the state in $\mathcal{B}^v$ corresponding to the nearest ancestor of S in π such that the ith element of $\mathbb{P}_{S_i}$ is the initial state of $Sem(\xi_i)$. Let h_i be the prefix of h ending at S_i. Then, each P_i in $\mathbb{P}_S$ is the recognition state of $h[\xi_i] - h_i[\xi_i]$, where the sequence difference denotes the suffix of $h[\xi_i]$ obtained by removing from the latter its prefix $h_i[\xi]$.

Proof: (*Sketch*) By induction on h. In the following, for $k \geq 0$, $\pi(k)$ denotes the prefix of π up to the kth arc, while $h(k)$ denotes the prefix of h composed of k transitions.[9]

(*Basis*) For $h(0) = [\,]$, Lemma 4 is trivially fulfilled.

(*Induction*) If Lemma 4 holds for $h(k)$ then it holds for $h(k+1)$ too. Let T_{k+1} be the last transition in $h(k+1)$. Two cases are possible, depending on whether or not there exists a context ξ whose alphabet includes T_{k+1}. If not, $\mathbb{P}'$ equals $\mathbb{P}$, thereby making Lemma 4 still true. Instead, if ξ does exist (this being the parent of the component performing transition T_{k+1}), then, based on the recursive rule introduced in the definition of $Bhv(\wp(\Sigma))$ (specifically, for the computation of $\mathbb{P}'$, point 3), the new state P' of $Sem(\xi)$ is either $\bar{P}$, if $Sem(\xi)$ includes $P \xrightarrow{T_{k+1}} \bar{P}$, or the initial state of $Sem(\xi)$, if such a transition does not exist.[10] In either case, $\bar{P}$ is the recognition state of $h(k+1)[\xi_i] - h_i[\xi_i]$. Afterward, if the interface symbol $\mathbb{I}$ marking the new state P' is not empty and ξ has a parent ξ_p in the hierarchy, then the rule is reapplied, with T_{k+1} and ξ being substituted by $\mathbb{I}$ and ξ_p, respectively. In fact, if $\mathbb{I}$ is not empty, P' is a final state of $Sem(\xi)$. Consequently, based on Proposition 2, the induction assumption, and line 13 of *Projection* specification (see Section V), $\mathbb{I}$ is the set of symbols $N \in \mathbf{I}$ such that (N, E) is a semantic pattern for ξ_p, and a suffix of $h(k+1)[\xi_p] - h_i[\xi_p]$ matches E (in plain form). The possible reiteration of the rule mimics the operational definition of projection (Section V), specifically, the loop within lines 12–18. Eventually, the condition stated by Lemma 4 is still true when h is replaced by $h(k+1)$. ■

Lemma 5: Let π be a path in $\mathcal{B}^v$ ending at a final state. Let h be the sequence of transitions marking arcs in π. Let $\pi_{\{\mathcal{R}\}}$ denote the set of faults relevant to π, namely

$$\pi_{\{\mathcal{R}\}} = \bigcup_{(B,\Im,\mathbb{P})\in\pi} \mathcal{F}(\mathbb{P}). \tag{7}$$

Then, $h_{\{\mathcal{R}\}} = \pi_{\{\mathcal{R}\}}$.

Proof: (*Sketch*) According to Section VIII-B, $\mathcal{F}(\mathbb{P})$ is the union of the faults associated with each state P_i of tuple $\mathbb{P}$ in $Sem(\xi_i)$, $i \in [1..n]$. We denote the subset of $\mathcal{F}(\mathbb{P})$ relevant to the ith state of $\mathbb{P}$ by $\mathcal{F}(P_i)$. We denote the prefix of h up to the kth transition by $h(k)$. Hence, $h(k)_{\{\mathcal{R}\}}$ denotes the diagnosis of $h(k)$ based on $\mathcal{R}$. Likewise, $\pi(k)$ denotes the prefix of path π up to the kth arc. Hence, $\pi(k)_{\{\mathcal{R}\}}$ denotes the set of faults in $\pi(k)$. The proof is by induction on h.

(*Basis*) $h(0)_{\{\mathcal{R}\}} = \pi(0)_{\{\mathcal{R}\}} = \emptyset$. In fact, $h(0) = [\,]$, hence, based on (3), $h(0)_{\{\mathcal{R}\}} = \emptyset$. On the other side, $\pi(0)$ is only composed of the initial state of $\mathcal{B}^v$, namely, $(B_0, \mathbb{P}_0, \Im_0)$, where $\mathcal{F}(\mathbb{P}_0) = \emptyset$, hence $\pi(0)_{\{\mathcal{R}\}} = \emptyset$.

(*Induction*) If Lemma 5 holds for $h(k)$ then it holds for $h(k+1)$ too. In other words, we assume that substituting h with $h(k)$ (and hence π with $\pi(k)$) Lemma 5 is true. Based on this assumption, we have to show that Lemma 5 is still true when h is substituted by $h(k+1)$ (and hence π with $\pi(k+1)$). To this end, let T_{k+1} be the last transition in $h(k+1)$, and $(B, \mathbb{P}, \Im) \xrightarrow{T_{k+1}} (B', \mathbb{P}', \Im')$ the corresponding arc in $\pi(k+1)$. Based on Lemma 4, each P_i in $\mathbb{P}$ is the recognition state of the suffix $h(k)[\xi_i] - h_i(k)[\xi_i]$, where $h_i(k)$ is the prefix of $h(k)$ ending at a state where the ith semantic-space state is the initial state of $Sem(\xi_i)$. Considering each ξ_i, $i \in [1..n]$, comparing $h(k)$ and $h(k+1)$, two cases are possible: either $h(k+1)[\xi_i] = h(k)[\xi_i]$ or $h(k+1)[\xi_i]$ is the extension of $h(k)[\xi_i]$ by a new symbol in the alphabet of ξ_i, namely $\mathbb{I}$.

If $h(k+1)[\xi_i] = h(k)[\xi_i]$ then, on the one hand, no state change is performed in $Sem(\xi_i)$, in other terms, the ith element of tuple $\mathbb{P}'$ equals the corresponding element in $\mathbb{P}$, that is, $P'_i = P_i$. Hence, $\mathcal{F}(P'_i) = \mathcal{F}(P_i)$, in other words, the

[9] In other words, $h(k)$ is the prefix of h corresponding to $\pi(k)$.

[10] According to Proposition 1, the initial state P_0 of a semantic space cannot be entered by a transition. Thus, setting $P' = P_0$ makes the nearest ancestor of S in h, namely S_i, to the the actual initial recognition state of a string in the language of the corresponding regular expression.

contribution of P'_i to $\mathcal{F}(\mathbb{P}')$ is empty. On the other hand, since $h(k+1)[\xi_i] = h(k)[\xi_i]$, no additional match can fulfill $e \sqsubseteq h(k+1)[\xi]$ in (3). Hence, the contribution of the new transition T_{k+1} to $h(k+1)_{\{\mathcal{R}\}}$ is empty. Therefore, both in $h(k+1)_{\{\mathcal{R}\}}$ and $\pi(k+1)_{\{\mathcal{R}\}}$, no additional fault relevant to ξ_i is generated.

If instead $h(k)[\xi_i]$ is extended into $h(k+1)[\xi_i]$ by a new symbol $\mathbb{I}$ (in the alphabet of ξ_i), then two cases are possible, depending on whether a transition $P_i \xrightarrow{\mathbb{I}} \bar{P}_i$ is matched in $Sem(\xi_i)$ or not.

If matched, two cases are possible, depending on whether or not $\bar{P}_i$ is final in $Sem(\xi_i)$.

If $\bar{P}_i$ is not final then $\mathcal{F}(\bar{P}_i) = \emptyset$, and no contribution to $\mathcal{F}(\mathbb{P}')$ comes from P'_i. On the other hand, since $\bar{P}_i$ is not final, the extension of $h(k+1)[\xi_i]$ by $\mathbb{I}$ does not provoke any new match relevant to ξ_i in (3), in other words, no further fault relevant to ξ_i is generated.

Instead, if $\bar{P}_i$ is final, then $\mathcal{F}(\bar{P}_i)$ is the contribution in $\mathcal{F}(\mathbb{P}')$ relevant to ξ_i. That is, $\mathcal{F}(P'_i) = \mathcal{F}(P_i) \cup \mathcal{F}(\bar{P}_i)$. On the other hand, based on Corollary 1, since $\bar{P}_i$ is final, in (3) additional matches for condition $e \sqsubseteq h(k+1)[\xi]$ hold, precisely, those relevant to faults in $\mathcal{F}(\bar{P}_i)$. In other words, the contribution relevant to ξ_i to $h(k+1)_{\{\mathcal{R}\}}$ equals $\mathcal{F}(\bar{P}_i)$.

If $P_i \xrightarrow{\mathbb{I}} \bar{P}_i$ is not matched in $Sem(\xi_i)$ then $P'_i = P_{i_0}$, and no contribution to $\mathcal{F}(\mathbb{P}')$ is given by $\mathcal{F}(P'_i)$.[11] On the other hand, in (3), the mismatch in $Sem(\xi_i)$ results in the impossibility of further matches for condition $e \sqsubseteq h(k+1)[\xi]$, in other words, no additional diagnosis relevant to ξ_i is generated.

In summary, applying the reasoning on all contexts ξ_i, $i \in [1..n]$, we conclude that $h(k+1)_{\{\mathcal{R}\}} = \pi(k+1)_{\{\mathcal{R}\}}$. ■

Lemma 6: If history $h \in \mathcal{B}^v$ ends at final state S_f, then $h_{\{\mathcal{R}\}} \in \Delta(S_f)$.

Proof: According to Lemma 5, $h_{\{\mathcal{R}\}} = \pi_{\{\mathcal{R}\}}$, where π is the path in $\mathcal{B}^v$ corresponding to h, with $\pi_{\{\mathcal{R}\}}$ being defined in (7). Thus, it suffices to show that $\pi_{\{\mathcal{R}\}} \in \Delta(S_f)$. Based on the two rules for decoration of $\mathcal{B}^v$ (Section VIII-B), the proof is by induction on h. In the following, $h(k)$ denotes the prefix of h up to the kth transition, $\pi(k)$ the prefix of π up to the kth transition, and S_k the last state in $\pi(k)$.

(*Basis*) Based on (7), $\pi(0)_{\{\mathcal{R}\}} = \emptyset$. According to decoration rule (1), $\emptyset \in \Delta(S_0)$. Hence $\pi(0)_{\{\mathcal{R}\}} \in \Delta(S_0)$.

(*Induction*) If $\pi(k)_{\{\mathcal{R}\}} \in \Delta(S_k)$ then $\pi(k+1)_{\{\mathcal{R}\}} \in \Delta(S_{k+1})$. In fact, let T_{k+1} be the transition marking the last arc in $\pi(k+1)$. On the one hand, according to (7), $\pi(k+1)_{\{\mathcal{R}\}} = \pi(k)_{\{\mathcal{R}\}} \cup \mathcal{F}(\mathbb{P}_{k+1})$, with $\mathbb{P}_{k+1}$ being relevant to S_{k+1}. On the other, according to decoration rule (2), for $S_k \xrightarrow{T_{k+1}} S_{k+1}$, if $\delta \in \Delta(S_k)$ then $(\delta \cup \mathcal{F}(\mathbb{P}_{k+1})) \in \Delta(S_{k+1})$. As, by assumption, $\pi(k)_{\{\mathcal{R}\}} \in \Delta(S_k)$, it follows that $\pi(k+1)_{\{\mathcal{R}\}} \in \Delta(S_{k+1})$. ■

Lemma 7: If S_f is a final state in $\mathcal{B}^v$ and $\delta \in \Delta(S_f)$ then there exists a history $h \in \mathcal{B}^v$ ending at S_f such that $h_{\{\mathcal{R}\}} = \delta$.

Proof: Based on the decoration rules for $\mathcal{B}^v$, diagnosis δ is incrementally generated by a path π starting from the empty diagnosis initially associated with S_0, specifically

$$\delta = \bigcup_{(B,\Im,\mathbb{P})\in\pi} \mathcal{F}(\mathbb{P}) \tag{8}$$

which, based on (7), equals $\pi_{\{\mathcal{R}\}}$, which on its turn, according to Lemma 5, equals $h_{\{\mathcal{R}\}}$, with the latter being the history generated by π. On the one hand, in order for h to be a history, π must be finite. If π is infinite then π must include (at least) a cycle in $\mathcal{B}^v$ traversed an infinite number of times. On the other, once a cycle is traversed, all associated $\mathcal{F}(\mathbb{P})$ are inserted into δ: successive iterations of the cycle do not extend δ because of duplicate removals caused by set-theoretic union in decoration rule (2). In other words, δ can be always generated by a finite path π and, hence, h is finite. ■

To prove Theorem 1, we show $\delta \in \Delta(\mathcal{B}^v) \Leftrightarrow \delta \in \Delta(\wp(\Sigma))$. On the one hand, if $\delta \in \Delta(\mathcal{B}^v)$ then, based on Lemmas 1, 2, and 7, there exists history $h \in \mathcal{B}^s$ such that $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$ and $h_{\{\mathcal{R}\}} = \delta$, in other words, based on (4), $\delta \in \Delta(\wp(\Sigma))$. On the other, if $\delta \in \Delta(\wp(\Sigma))$ then, according to (4) and based on Lemmas 3 and 6, there exists a history $h \in \mathcal{B}^v$ ending at final state S_f such that $\delta = h_{\{\mathcal{R}\}}$ and $\delta \in \Delta(S_f)$, in other words, $\delta \in \Delta(\mathcal{B}^v)$. ■

X. Discussion

The contribution of this paper is both scientific and practical. From the scientific viewpoint, it provides diagnosis with context-sensitivity, thereby allowing for viewing a system at different levels of abstractions, the contexts, each one characterized by its own set of faults, which cannot be simply inferred from the faults of lower-level contexts.

From a practical perspective, the proposed method may be applied to real systems in which different levels of diagnosis are required. Consider the operator in a control room responsible for the monitoring of a large power network. When a short circuit occurs on a transmission line and some devices are faulty, the extent of the isolation may involve several lines. In order to minimize the misbehavior, the operator is required to perform some actions on the network within stringent time constraints. These actions are safe only if the operator has a clear picture of what is happening (including the location of the short circuit). Otherwise, an inappropriate action may result in the loss of additional lines, which may lead to a blackout. If the diagnosis provided by the supervision system is at levels of components (breakers and protection devices) and context-free, it may be difficult for the operator deciding the correct actions. By contrast, providing a context-sensitive diagnosis at different levels of abstraction is bound to result in a clear understanding of the risks associated with the actions.

Compared with context-free diagnosis of active systems [30], context-sensitive diagnosis requires more computational resources. In both approaches, the uncertainty of the temporal observation requires the generation of the index space. However, once the index space is available, the field $\Im$ in the nodes of the reconstructed behavior is simply a scalar value (the identifier of a state of the index space).[12] In

[11] Based on Lemma 4, successive states in $Sem(\xi_i)$ represent the recognition of a suffix of $h[\xi_i]$ starting from a successive state in π. At a first sight, this might cause the loss of completeness for $\pi_{\{\mathcal{R}\}}$, specifically for the matches in (3) where string e is extended by $\mathbb{I}$. The fact is, if $\mathbb{I}$ introduces a discontinuity in the matching of e, then no string e including symbol $\mathbb{I}$ (in that position) will match any regular expression E. Thus, completeness is actually preserved.

[12] Remember that a node of the reconstructed behavior is a triple $(B, \mathbb{P}, \Im)$.

context-sensitive diagnosis, in order to trace pattern-matching of semantic spaces, nodes of the reconstructed behavior need the additional tuple $\mathbb{P}$ of semantic-space states. This may result in a considerably larger reconstructed behavior.

However, a strong point of the proposed method is that, unlike [3] and related approaches, no global model of the system must be generated (not even off-line), as no diagnoser is required by the diagnosis engine. As remarked in Section III, the behavior space is introduced for formal reasons only, as the reconstruction of the behavior is carried out based only on the models of components and their connections. Depending on the degree of information in the temporal observation, only a (possibly small) portion of the behavior space is actually generated. This provides support to scalability for relatively large scale systems.

This paper is built upon [6], where pattern stratification is only apparent, as, after macrosubstitution, the regular expression is invariably defined on component transitions. In this paper, instead, pattern stratification is real, since alphabets of regular expressions are stratified based on interface symbols of nested contexts.

A related approach is proposed in [4], where the notion of supervision pattern is introduced, which allows for a flexible specification of the diagnosis problem, and for a uniform solution of different classes of problems. However, three points are to be highlighted. First, supervision patterns are specified by automata. In this paper, instead, fault patterns are specified by regular expressions. Second, a supervision pattern specifies which system evolutions are to be considered as faulty (or, more generally, significant to the supervision process), based on specific occurrences of fault (and repair) events. In this paper, instead, a fault pattern specifies a (complex) fault within a system evolution. Finally, and more importantly, [4] does not provide any hierarchical abstraction to diagnosis: since a diagnosis is an evolution identified by a supervision pattern, the notion of diagnosis invariably refers to the system as a whole. In this paper, instead, the interpretation of the system behavior is based on the context hierarchy, where diagnosis rules are defined for each subsystem in the hierarchy.

This paper also differs both from hierarchical diagnosis of static systems [35], [36], which are based on structural decomposition, without any concern with context-sensitivity, and from diagnosis of hierarchical finite state machines (HFSMs), which are inspired by state-charts [37], [38]. The most important feature of an HFSM is hierarchical state-nesting: if a system is in a nested state (sub-state), it is also in all its surrounding states (super-states). Moreover, transitions are defined at each level of the hierarchy. HFSMs were considered for solving a class of control problems in [39]. Recently, diagnosis of HFSMs has been considered in [40], [41]. However, no patterns are involved and diagnosis is context-free.

XI. Conclusion

The state-of-the-art diagnosis techniques for DESs inherit, in one way or another, the approach introduced in the seminal work of [3]: diagnosis is anchored to components, irrespective of the context in which they are embedded, in other words, diagnosis is context-free. When the DES is complex, a shift of modeling is needed, as the DES involves several contexts, each one being qualified by its own diagnosis rules, which may or may not depend on the diagnosis rules of its subcontexts completely. This way, rather being tightly tied to components, faults are anchored to contexts: diagnosis is context-sensitive. Diagnosis rules are specified by semantic patterns, where faults are associated with regular expressions. When the context is made up of components only, the regular expression defines a pattern of transitions for such components. In the general case, when the context is made up of subcontexts, the regular expression defines a pattern of interface symbols associated with its subcontexts, with each interface symbol being associated with a regular expression in the subcontexts. However, we do not consider the proposed notation for semantic-pattern specification as final: different formalisms can be envisaged to fit different classes of DESs. It is our belief that separation of concerns and enhancement in expressive power of context-sensitive diagnosis may be exported to diagnosis of complex systems other than DESs, from static systems to general dynamic systems [42].

References

[1] Enquiry Committee on Grif Disturbance. (2012). *Report of the Enquiry Committee on Grif Disturbance in Northern Region on 30th July 2012 and in Northern, Eastern & North-Eastern Region on 31st Jul. 2012* [Online]. Available: http://www.powermin.nic.in/pdf/GRID_ENQ_REP_16_8_12.pdf.

[2] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems (The Kluwer International Series in Discrete Event Dynamic Systems)*, vol. 11. Boston, MA, USA: Kluwer Academic 1999.

[3] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Failure diagnosis using discrete-event models," *IEEE Trans. Control Syst. Technol.*, vol. 4, no. 2, pp. 105–124, Mar. 1996.

[4] T. Jéron, H. Marchand, S. Pinchinat, and M. Cordier, "Supervision patterns in discrete event systems diagnosis," in *Proc. 17th Int. Workshop Principles Diagnosis DX*, 2006, pp. 117–124.

[5] G. Lamperti and M. Zanella, "Injecting semantics into diagnosis of discrete-event systems," in *Proc. 21st Int. Workshop Principles Diagnosis DX*, 2010, pp. 233–240.

[6] G. Lamperti and M. Zanella, "Context-sensitive diagnosis of discrete-event systems," in *Proc. 22nd IJCAI*, vol. 2. 2011, pp. 969–975.

[7] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, "Diagnosis of large active systems," *Artif. Intell.*, vol. 110, no. 1, pp. 135–183, 1999.

[8] A. Grastien, M. Cordier, and C. Largouët, "Incremental diagnosis of discrete-event systems," in *Proc. 16th Int. Workshop Principles Diagnosis DX*, 2005, pp. 119–124.

[9] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, "Diagnosis of a class of distributed discrete-event systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 30, no. 6, pp. 731–752, Nov. 2000.

[10] Y. Pencolé, "Decentralized diagnoser approach: Application to telecommunication networks," in *Proc. 11th Int. Workshop Principles Diagnosis DX*, 2000, pp. 185–192.

[11] R. Debouk, S. Lafortune, and D. Teneketzis, "Coordinated decentralized protocols for failure diagnosis of discrete-event systems," *J. Discrete Event Dynamic Syst. Theory Applicat.*, vol. 10, nos. 1–2, pp. 33–86, 2000.

[12] Y. Pencolé, M. Cordier, and L. Rozé, "Incremental decentralized diagnosis approach for the supervision of a telecommunication network," in *Proc. 12th Int. Workshop Principles Diagnosis DX*, 2001, pp. 151–158.

[13] R. Debouk, S. Lafortune, and D. Teneketzis, "On the effect of communication delays in failure diagnosis of decentralized discrete event systems," *J. Discrete Event Dynamic Syst. Theory Applicat.*, vol. 13, no. 3, pp. 263–289, 2003.

[14] A. Grastien, M. Cordier, and C. Largouët, "Extending decentralized discrete-event modelling to diagnose reconfigurable systems," in *Proc. 15th Int. Workshop Principles Diagnosis DX*, 2004, pp. 75–80.

[15] Y. Pencolé and M. Cordier, "A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks," *Artif. Intell.*, vol. 164, nos. 1–2, pp. 121–170, 2005.
[16] W. Qiu and R. Kumar, "Decentralized failure diagnosis of discrete event systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 36, no. 2, pp. 384–395, Mar. 2006.
[17] G. Lamperti and M. Zanella, "Diagnosis of discrete-event systems from uncertain temporal observations," *Artif. Intell.*, vol. 137, nos. 1–2, pp. 91–163, 2002.
[18] G. Lamperti and M. Zanella, "Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques," *Artif. Intell.*, vol. 170, no. 3, pp. 232–297, 2006.
[19] X. Zhao and D. Ouyang, "Model-based diagnosis of discrete event systems with an incomplete system model," in *Proc. 18th Eur. Conf. Artif. Intell.*, 2008, pp. 189–193.
[20] G. Lamperti and M. Zanella, "Monitoring of active systems with stratified uncertain observations," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 2, pp. 356–369, Mar. 2011.
[21] R. Kwong and D. Yonge-Mallo, "Fault diagnosis in discrete-event systems: Incomplete models and learning," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 1, pp. 118–130, Feb. 2011.
[22] X. Zhao, D. Ouyang, L. Zhang, X. Wang, and Y. Mo, "Reasoning on partially-ordered observations in online diagnosis of DESs," *AI Commun.*, vol. 25, no. 4, pp. 285–294, 2012.
[23] S. Sohrabi, J. Baier, and S. McIlraith, "Diagnosis as planning revisited," in *Proc. 12th Int. Conf. Knowl. Represent. Reason.*, 2010, pp. 26–36.
[24] A. Grastien, P. Haslum, and S. Thiébaux, "Conflict-based diagnosis of discrete event systems: Theory and practice," in *Proc. 13th Int. Conf. Knowl. Represent. Reason.*, 2012, pp. 489–499.
[25] S. Tripakis, "Fault diagnosis for timed automata," in *Formal Techniques in Real-Time and Fault-Tolerant Systems (LNCS)*, vol. 2469, W. Damm and E.-R. Olderog, Eds. Berlin Heidelberg: Springer-Verlag, 2002, pp. 205–221.
[26] S. Biswas, D. Sarkar, P. Bhowal, and S. Mukhopadhyay, "Diagnosis of delay-deadline failures in real time discrete event models," *ISA Trans.*, vol. 46, no. 4, pp. 569–582, 2007.
[27] E. Gascard and Z. Simeu-Abazi, "Modular modeling for the diagnostic of complex discrete-event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 4, pp. 1101–1123, Oct. 2013.
[28] N. Chomsky, "Three models for the description of language," *Trans. Inf. Theory*, vol. 2, no. 3, pp. 113–124, 1956.
[29] W. Isaacson, *Einstein: His Life and Universe*. New York, NY, USA: Simon & Shuster, 2008.
[30] G. Lamperti and M. Zanella, *Diagnosis of Active Systems: Principles and Techniques*, vol. 741. Dordrecht, NL, USA: Kluwer Academic Publishers, 2003.
[31] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *J. ACM*, vol. 30, no. 2, pp. 323–342, 1983.
[32] G. Lamperti and M. Zanella, "Generation of diagnostic knowledge by discrete-event model compilation," in *Proc. 7th Int. Conf. Knowl. Represent. Reason.*, 2000, pp. 333–344.
[33] G. Lamperti and M. Zanella, "Diagnosis of discrete-event systems by separation of concerns, knowledge compilation, and reuse," in *Proc. ECAI*, 2004, pp. 838–842.
[34] A. Aho, M. Lam, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2006.
[35] L. Chittaro and R. Ranon, "Hierarchical model-based diagnosis based on structural abstraction," *Artif. Intell.*, vol. 155, nos. 1–2, pp. 147–182, 2004.
[36] S. Siddiqi and J. Huang, "Hierarchical diagnosis of multiple faults," in *Proc. IJCAI*, 2007, pp. 581–586.
[37] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.
[38] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, *et al.*, "STATEMATE: A working environment for the development of complex reactive systems," *IEEE Trans. Softw. Eng.*, vol. 16, no. 4, pp. 403–414, Apr. 1990.
[39] H. Brave and M. Heymann, "Control of discrete event sysytems modeled as hierarchical state machines," *IEEE Trans. Autom. Control*, vol. 38, no. 12, pp. 1803–1819, Dec. 1993.
[40] A. Idghamishi and S. Zad, "Fault diagnosis in hierarchical discrete-event systems," in *Proc. 43rd IEEE Conf. Decision Control*, 2004, pp. 63–68.
[41] A. Paoli and S. Lafortune, "Diagnosability analysis of a class of hierarchical state machines," *J. Discrete Event Dyn. Syst. Theory Applicat.*, vol. 18, no. 3, pp. 385–413, 2008.
[42] P. Struss, "Fundamentals of model-based diagnosis of dynamic systems," in *Proc. IJCAI*, 1997, pp. 480–485.

Gianfranco Lamperti (M'00) received the Doctoral degree in Electronic Engineering (Computer Science) from Politecnico di Milano, Milan, Italy, in 1986.

He is currently an Associate Professor of Computer Science with the Department of Information Engineering, University of Brescia, Brescia, Italy. His current research interests include engineering issues in model-based diagnosis of discrete-event systems and automata.

Xiangfu Zhao (M'08) received the Doctoral degree in Computer Science from Jilin University, Jilin, China, in 2009.

He is currently an Associate Professor of Computer Science at Zhejiang Normal University, Zhejiang, China. His current research interests include model-based diagnosis for static systems and discrete-event systems.