Context-Sensitive Diagnosis of Discrete-Event Systems

Gianfranco Lamperti, Marina Zanella
University of Brescia, Italy
gianfranco.lamperti@ing.unibs.it, marina.zanella@ing.unibs.it

Abstract

Since the seminal work of Sampath et al. in 1996, despite the subsequent flourishing of techniques on diagnosis of discrete-event systems (DESs), the basic notions of fault and diagnosis have been remaining conceptually unchanged. Faults are defined at component level and diagnoses incorporate the occurrences of component faults within system evolutions: diagnosis is *context-free*. As this approach may be unsatisfactory for a complex DES, whose topology is organized in a hierarchy of abstractions, we propose to define different diagnosis rules for different subsystems in the hierarchy. Relevant fault patterns are specified as regular expressions on patterns of lower-level subsystems. Separation of concerns is achieved and the expressive power of diagnosis is enhanced: each subsystem has its proper set of diagnosis rules, which may or may not depend on the rules of other subsystems. Diagnosis is no longer anchored to components: it becomes *context-sensitive*. The approach yields seemingly contradictory but nonetheless possible scenarios: a subsystem can be normal despite the faulty behavior of a number of its components (positive paradox); also, it can be faulty despite the normal behavior of all its components (negative paradox).

1 Introduction

A wide variety of techniques for diagnosis of discrete-event systems (DESs) [Cassandras and Lafortune, 1999] have been proposed in the last years, including [Baroni *et al.*, 1999; Debouk *et al.*, 2000; Lamperti and Zanella, 2003; Pencolé and Cordier, 2005; Qiu and Kumar, 2006]. However, after the seminal work of [Sampath *et al.*, 1996], the basic notions of fault and diagnosis have been remaining conceptually unchanged. Typically, a fault is 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. The claim of this paper is that anchoring faults at component level is too restrictive an approach when complex DESs are involved.

2 Context-Sensitive Diagnosis

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 level, 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 a 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, 1956] 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 depends on the context, production rules in BNF notation are unable to force all the syntax constraints of the language (for example: the list of actual parameters in a function call must match, in number and types, the list of formal parameters). In compilers, the check of these (context-sensitive) constraints are generally left to semantic analysis.

Loosely speaking, the idea of context-sensitivity can be profitably injected into diagnosis of DESs in three ways:

- 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;
- The fault of a component is not necessarily ascribed to one transition: it can be the result of several transitions;
- 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*.

2.1 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 of Fig. 1 is a simplified representation of a power transmission line. The line is protected on both sides by a redundant protection hardware, called $\mathcal{W}$ and $\mathcal{W}'$, respectively, involving one protection and two breakers. For instance, $\mathcal{W}$ incorporates protection p, and breakers b_1 and b_2. When a lightening strikes the line, a short circuit may occur on the latter. The protection system is designed to open the breakers in order to isolate the line, which eventually causes the extinction of the short. To this end, when detecting a short circuit, a protection is expected to trigger both breakers to open. A protection is faulty when either it does not detect the short or, after the short detection, it does not trigger the breakers. A breaker is faulty when, after receiving the triggering command from the protection, it does not open. A minimal (non redundant) protection hardware would incorporate a single protection and a 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 and of *one* breaker (for each side) to guarantee the isolation of the line. Consider the following two scenarios.

1. In $\mathcal{W}$, b_1 is faulty, while p and b_2 are normal. In $\mathcal{W}'$, b_1' is faulty, while p' and b_2' are normal. The diagnosis is $\delta_1 = \{b_1, b_1'\}$. 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 $\mathcal{W}$, b_1 is faulty, while p and b_2 are normal. In $\mathcal{W}'$, p' is faulty, while b_1' and b_2' are normal. The diagnosis is $\delta_2 = \{b_1, p'\}$. 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 (p' vs. b_1'), 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 an abstraction hierarchy, and require for each of them a diagnosis. Such a hierarchy is displayed in the right of Fig. 1, where $\mathcal{L}$ denotes the whole protected line. According to such a hierarchy, the diagnosis of the first scenario is $\{b_1, b_1'\}$, while in the second scenario the diagnosis is $\{b_1, p', \mathcal{W}', \mathcal{L}\}$. Comparing the two diagnoses, we conclude that in the first scenario two components

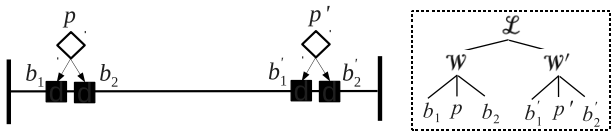

Figure 1: Protected power transmission line (left) and relevant abstraction hierarchy (right).

are faulty but their misbehavior is not propagated to higher subsystems. By contrast, in the second scenario, besides two faulty components (b_1 and p'), $\mathcal{W}'$ and $\mathcal{L}$ are faulty too. ◇

2.2 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:

- A software system can be faulty even if all its software components are bug-free;
- A marriage may fail even if the behavior of each partner is unfailing on its own;
- A human society can be bound to dictatorship notwithstanding all its democratic institutions.

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 lightening, 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 protections and breakers was normal, the behavior of the line is actually faulty. ◇

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 real 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.

3 Active Systems

Active systems [Lamperti and Zanella, 2003] are a special class of asynchronous DESs. An active system is a network of components that are connected to one another through links. Each component is modeled as a communicating automaton [Brand and Zafiropulo, 1983], that reacts to events either coming from the external world or from neighboring components through links, and is endowed with input and output terminals. A transition from one state to another is triggered by an event ready at an input terminal: such 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. However, a strong assumption for diagnosis of active systems is the unavailability of the global model as, in real applications, the generation of the global

model is impractical. 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 *history* in $Bsp(\Sigma, \Sigma_0)$ is the list of component transitions marking the arcs of a path rooted in Σ_0.

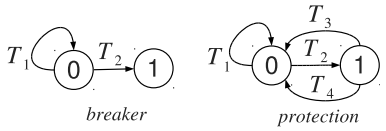

Figure 2: Behavioral models of components.

Example 3 With reference to Example 1, shown in Fig. 2 are the behavioral models of breaker and protection. The automaton of the breaker includes two states, 0 (closed) and 1 (open), and two transitions, which are both triggered by a tripping event sent by the protection. While T_2 moves the breaker to open (correct behavior), T_1 leaves the breaker *stuck to closed*. The automaton of the protection involves two states, 0 (idle) and 1 (awaken), and four transitions. Transition T_2 is triggered by the occurrence of a short circuit on the line and awakes the protection. The actual tripping command to breakers is generated by T_4. However, the protection may return to state 0 without sending any command to breakers (T_3). The protection may even not be able to detect the short circuit (T_1). Based on Fig. 1, the active system $\mathcal{W}$ consists of protection p and breakers b_1 and b_2. The behavior space of $\mathcal{W}$ is outlined in Fig. 3, where arcs are marked by component transitions (labels in bold will be explained shortly). ◇

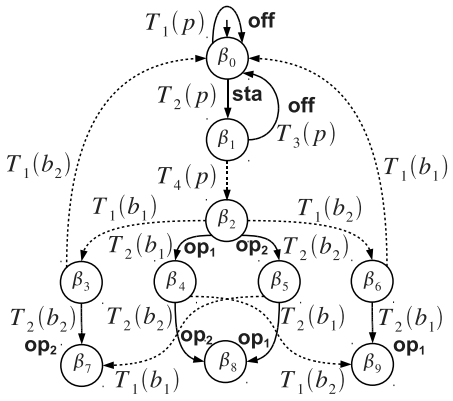

Figure 3: Behavior space $Bsp(\mathcal{W}, \mathcal{W}_0)$.

4 Diagnosis Problem

When reacting, an active system performs a list of transitions (history), that moves the system from the initial state to a final state. Since a number of such transitions are perceived to the observer as visible labels, such a history generates a list of labels, the *trace* of the history. A *diagnosis problem* $\wp$ for a system Σ is a 4-tuple

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

where Σ_0 is the initial state of the system, $\mathcal{V}$ the *viewer*, $\mathcal{O}$ the *observation*, and $\mathcal{R}$ the *ruler*. 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 observable labels and arcs denote (partial) temporal precedence between nodes.

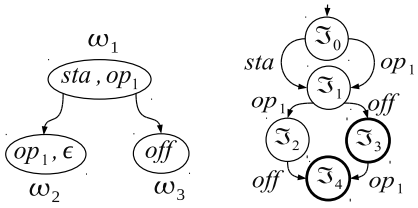

Figure 4: Observation $\mathcal{O}$ and index space $Isp(\mathcal{O})$.

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: $T_2(b_1)$: op_1, $T_2(b_2)$: op_2, $T_1(p)$: *off*, $T_2(p)$: *sta*, $T_3(p)$: *off*. Displayed in the left of Fig. 4 is an observation $\mathcal{O}$ relevant to viewer $\mathcal{V}$, which is composed of three nodes and two arcs. ◇

An observation implicitly embodies several *candidate traces*, denoted $\|\mathcal{O}\|$, each one obtained by picking up one label from each node of the observation without violating the temporal constraints imposed by arcs. Among such candidates is the (unknown) trace actually generated by the system history. For practical reasons, an *index space* of the observation $\mathcal{O}$ is generated, denoted $Isp(\mathcal{O})$. 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 $\|\mathcal{O}\|$, in other words, the set of paths in $Isp(\mathcal{O})$ equals the set of candidate traces of $\mathcal{O}$.

Example 5 Shown in the right of Fig. 4 is the index space of observation $\mathcal{O}$, where $\Im_3$ and $\Im_4$ are the final states. ◇

The ruler $\mathcal{R}$ is a 5-tuple

$$\mathcal{R} = (\mathcal{D}, \mathcal{H}, \mathcal{N}, \mathcal{F}, \mathcal{S}) \tag{2}$$

where $\mathcal{D}$ is the *diagnosis domain*, $\mathcal{H}$ the *abstraction hierarchy*, $\mathcal{N}$ the *name space*, $\mathcal{F}$ the *fault space*, and $\mathcal{S}$ the *pattern specification*. More specifically, the diagnosis domain $\mathcal{D} = \{\sigma_1, \ldots, \sigma_n\}$ is the set of subsystems of Σ that are stated as relevant to the output of the diagnosis task. The abstraction hierarchy $\mathcal{H}$ defines a partition of each $\sigma_i \in \mathcal{D}$, $i \in [1\,..\,n]$, in terms of relevant subsystems, namely $(\sigma_i, \{\sigma_{i_1}, \ldots, \sigma_{i_{n_i}}\})$. The name space $\mathcal{N}$ is the set of identifiers used in $\mathcal{S}$, while $\mathcal{F} \subseteq \mathcal{N}$ is the set of *faults*. Finally, the pattern specification $\mathcal{S}$ is a set of pairs $(\sigma, \mathcal{P})$, where $\sigma \in \mathcal{D}$ and $\mathcal{P} = [P_1, \ldots, P_k]$ is the *pattern list*, with each pattern $P_i \in \mathcal{P}$, $P_i = (N_i, E_i)$, being the association between a name $N_i \in \mathcal{N}$ and a regular expression E_i. The alphabet of E_i is defined as follows. Let $(\sigma, \{\sigma_1, \ldots, \sigma_m\}) \in \mathcal{H}$. The *alphabet* $\mathcal{A}$ of σ, where $(\sigma, \mathcal{P}) \in \mathcal{S}$, is the union of the names relevant to patterns in $\mathcal{P}$. The alphabet of a component C is the set of transitions relevant to the model of C. The alphabet $\mathcal{A}$ of E_i is defined

as the union of the alphabets of the subsystems of σ and the pattern names defined up to P_{i-1}:

$$\mathcal{A}(E_i) = \mathcal{A}(\sigma_1) \cup \cdots \cup \mathcal{A}(\sigma_m) \cup \{N_1, \ldots, N_{i-1}\}. \quad (3)$$

In other words, the regular expression E_i is defined on the pattern names relevant to the subsystems of σ and those defined before P_i in the same pattern list $\mathcal{P}$ involving E_i. The *plain form* of E_i is the macro-substitution of each name in $\mathcal{N}$ by the corresponding regular expression. As such, the plain E_i is defined on the transitions of the components incorporated in σ. 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 $L(y)$):

- ϵ denotes the language $\{\epsilon\}$ (where ϵ is the *null* symbol);
- If $a \in \mathcal{A}$ then a denotes $\{a\}$;
- (x) denotes $L(x)$ (parentheses are allowed as usual) ;
- $x^?$ denotes $L(x) \cup \{\epsilon\}$ (optionality);
- x^* denotes $\bigcup_{i=0}^{\infty}(L(x))^i$ (iteration zero or more times);
- x^+ denotes $\bigcup_{i=1}^{\infty}(L(x))^i$ (iteration one or more times);
- xy denotes $L(x)L(y)$ (concatenation);
- $x \mid y$ denotes $L(x) \cup L(y)$ (alternative);
- $x \,\&\, y$ stands for $(xy \mid yx)$ (free concatenation).

Table 1: Pattern specification for system $\mathcal{W}$.

Subsystem σ	**Pattern list** $\mathcal{P}$
b_1	$stc(b_1) = T_1(b_1)$
b_2	$stc(b_2) = T_1(b_2)$
p	$deaf(p) = T_1(p)$ $lazy(p) = T_2(p)T_3(p)$ $fail(p) = deaf(p) \mid lazy(p)$
$\mathcal{W}$	$stc(b_{12}) = stc(b_1)\ \&\ stc(b_2)$ $nop(\mathcal{W}) = fail(p) \mid stc(b_{12})$

Example 6 A ruler $\mathcal{R}$ for system $\mathcal{W}$ defined in Example 3 is specified as follows. Diagnosis domain: $\mathcal{D} = \{\mathcal{W}, p, b_1, b_2\}$. The abstraction hierarchy $\mathcal{H}$ adheres to the tree outlined in the right of Fig. 1. $\mathcal{N} = \{stc(b_1), stc(b_2), deaf(p), lazy(p), fail(p), stc(b_{12}), nop(\mathcal{W})\}$. $\mathcal{F} = \{stc(b_1), stc(b_2), deaf(p), lazy(p), nop(\mathcal{W})\}$. $\mathcal{S}$ is defined as shown in Table 1. For breakers b_1 and b_2, only one pattern is defined (*stc*, stuck to closed), involving transition T_1. For protection p, three patterns are listed: *deaf* (p does not perceive the short circuit), *lazy* (although perceiving the short, p does not trip breakers), and *fail* (p is either *deaf* or *lazy*). For the protection hardware $\mathcal{W}$, pattern $stc(b_{12})$ indicates both breakers stuck to closed. Finally, pattern *nop* (no operation) specifies when the protection hardware is faulty as a whole (when either p fails or both breakers are stuck to closed). ◇

5 Preprocessing

Once fault patterns are specified in terms of regular expressions, some (off-line) preprocessing on them is worthwhile to speed up the (on-line) diagnosis engine (which is in charge of solving the diagnosis problem). As known, [Aho *et al.*, 2006], any regular expression can be automatically transformed into an equivalent deterministic finite automaton. Specifically, two steps are performed:

1. For each pattern (F_i, E_i) such that $F_i \in \mathcal{F}$, E_i is unfolded into its plain form (to include component transitions only) and an equivalent deterministic automaton A_i is generated, where final states are marked by F_i.
2. The so-generated deterministic automata $A_1, \ldots, A_n$ *within the same pattern list* $\mathcal{P}$ (and, therefore, relevant to the same subsystem σ) are merged to yield a new deterministic automaton called the *pattern space* of σ, $Pts(\sigma)$, where each final state S is marked by the set of faults associated with states identifying S.[1]

The merging of automata in step 2 is performed as follows:

- Each automaton A_i is extended by inserting an empty transition from each non-initial state to the initial state.
- An empty transition is inserted from each initial state of each A_i to the initial state of each A_j, $j \neq i$.
- The so-obtained nondeterministic automaton is transformed into an equivalent deterministic automaton, in fact $Pts(\sigma)$, having as initial state the ϵ-closure of the set of initial states of $A_1, \ldots, A_n$. Each final state S of $Pts(\sigma)$ is marked by the union of faults associated with the states in S that are final in the corresponding A_i.

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

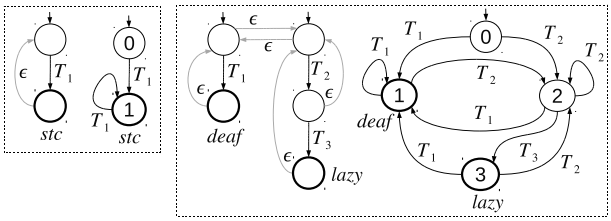

Figure 5: Generation of pattern spaces $Pts(b)$ and $Pts(p)$.

Example 7 Shown in Fig. 5 is the generation of the pattern space of a generic breaker b (left) and of protection p (right). Considering b, based on Table 1, fault pattern *stc* is defined as the single transition T_1. The corresponding deterministic automaton generated by step 1 is represented by the two states and the arc marked by T_1. Step 2 is performed by inserting the empty transition (gray arc). $Pts(b)$ is outlined next (second automaton in the figure).[2] The final node is marked by fault *stc*. Likewise, the generation of $Pts(p)$ is outlined in the right of Fig. 5. For space reasons, we have omitted the generation of $Pts(\mathcal{W})$ (whose final nodes are marked by fault *nop*). ◇

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

[2] States are identified by numbers, having however different meaning with respect to the homonym states of the models in Fig. 2.

6 Problem Solution

The solution of a diagnosis problem $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$, written $\Delta(\wp(\Sigma))$, is a set of *candidate diagnoses*, where each candidate is a (possibly empty) set of faults. To define precisely $\Delta(\wp(\Sigma))$, we need to introduce a few notations. The trace of a history h of Σ (based on viewer $\mathcal{V}$) is written $h_{[\mathcal{V}]}$. The *projection* of h on a subsystem σ, written $h_{[\sigma]}$, is the sublist of transitions of h relevant to components in σ. The language of a regular expression E is denoted by $\|E\|$.

Now, assume ruler $\mathcal{R} = (\mathcal{D}, \mathcal{H}, \mathcal{N}, \mathcal{F}, \mathcal{S})$. The diagnosis of a history h based on $\mathcal{R}$ is:

$$h_{[\mathcal{R}]} = \{\ F \mid F \in \mathcal{F},\ (F, E) \in \mathcal{P},\ (\sigma, \mathcal{P}) \in \mathcal{S},\ e \in \|E\|,\ e \subseteq h_{[\sigma]}\ \}. \quad (4)$$

That is, the diagnosis $h_{[\mathcal{R}]}$ is the set of faults F involved in a pattern list $\mathcal{P}$ of subsystem σ, such that there exists a string e, in the language of the regular expression E relevant to F, that is a sublist of the projection on σ of a history h. The *solution* of the diagnosis problem $\wp(\Sigma)$ is:

$$\Delta(\wp(\Sigma)) = \{\ h_{[\mathcal{R}]} \mid h \in Bsp(\Sigma, \Sigma_0),\ h_{[\mathcal{V}]} \in \|\mathcal{O}\|\ \}. \quad (5)$$

That is, the solution of $\wp(\Sigma)$ is the set of diagnoses of histories h whose trace, $h_{[\mathcal{V}]}$, is a candidate trace in $\mathcal{O}$.

Example 8 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, where $\mathcal{W}_0 = \beta_0$ is the initial state and visible transitions are marked by relevant labels (in bold). The solution of $\wp(\Sigma)$ can be determined based on eqn. (5). 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). Of the six paths in $Isp(\mathcal{O})$ (from the initial state $\Im_0$ to a final state (either $\Im_3$ or $\Im_4$), only $[sta, off]$ is consistent with $Bsp(\mathcal{W}, \mathcal{W}_0)$, that is generated by history $h = [T_2(p), T_3(p)]$. Then, we have to determine the set δ of faults, where each fault F is associated with a regular expression E for subsystem σ such that a path in $\|E\|$ is contained in the projection of h on σ. Based on the pattern specification in Table 1, of the five pattern names in $\mathcal{F}$, namely $stc(b_1)$, $stc(b_2)$, $deaf(p)$, $lazy(p)$, and $nop(\mathcal{W})$, only $lazy(p)$ and $nop(\mathcal{W})$ meet such a condition. In fact, the only string in the regular expression of $lazy(p)$ equals $h_{[p]} = h$. Besides, unfolding the regular expression associated with $nop(\mathcal{W})$ yields the plain expression:

$$T_1(p) \mid T_2(p)T_3(p) \mid T_1(b_1)T_1(b_2) \mid T_1(b_2)T_1(b_1)$$

which includes $h_{[\mathcal{W}]} = h$. Thus, based on eqn. (5), the solution of $\wp(\mathcal{W})$ is: $\Delta(\wp(\mathcal{W})) = \{\, lazy(p),\ nop(\mathcal{W})\,\}$. ◇

7 Diagnosis Engine

The diagnosis engine is required to take as input a diagnosis problem $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ and to output the solution $\Delta(\wp(\Sigma))$, as defined in eqn. (5). In doing so, it needs to reconstruct the behavior of Σ that conforms to the observation, namely $Bhv(\wp(\Sigma))$. We assume that the pattern spaces generated off-line by preprocessing (see Section 5) are available to the engine. Since a sort of pattern recognition is to be performed, states of $Bhv(\wp(\Sigma))$ will incorporate information not only on the observation but also on the pattern spaces in order to maintain the state of the matching.

Let $\mathbb{B}$ denote the set of states of $Bsp(\Sigma, \Sigma_0)$, $\mathbb{I}$ the set of states of $Isp(\mathcal{O})$, and $\mathbb{P} = \mathbb{P}_1 \times \mathbb{P}_2 \times \cdots \times \mathbb{P}_n$, where $\mathbb{P}_1, \ldots, \mathbb{P}_n$ are the set of states of pattern spaces $Pts(\sigma_1), \ldots, Pts(\sigma_n)$, respectively. The *behavior*

$$Bhv(\wp(\Sigma)) = (\mathbb{S}, \mathbb{T}, S_0, \mathbb{S}_f) \quad (6)$$

is a deterministic automaton such that:

- $\mathbb{S} \subseteq \mathbb{B} \times \mathbb{I} \times \mathbb{P}$ is the set of states;
- $S_0 = (\Sigma_0, \Im_0, \mathbf{P}_0)$ is the initial state, where $\Im_0$ is the initial state of $Isp(\mathcal{O})$, while $\mathbf{P}_0 = (P_{1\,0}, \ldots, P_{n\,0})$ is the record of the initial states of $Pts(\sigma_1), \ldots, Pts(\sigma_n)$;
- $\mathbb{S}_f = \{(\beta, \Im_f, \mathbf{P}) \mid \Im_f \text{ is final in } Isp(\mathcal{O})\}$ is the set of final states;
- $\mathbb{T}$ is the transition function: $(\beta, \Im, \mathbf{P}) \xrightarrow{T} (\beta', \Im', \mathbf{P}') \in \mathbb{T}$, $\mathbf{P} = (P_1, \ldots, P_n)$, $\mathbf{P}' = (P'_1, \ldots, P'_n)$, iff the following conditions hold:
  1. $\beta \xrightarrow{T} \beta'$ is a transition in $Bsp(\Sigma, \Sigma_0)$;[3]
  2. if T is invisible then $\Im' = \Im$ else $\Im'$ is the target state of transition $\Im \xrightarrow{\ell} \Im'$ in $Isp(\mathcal{O})$, where ℓ is the label associated with T in $\mathcal{V}$;
  3. $\mathbf{P}' = (P'_1, \ldots, P'_n)$ is such that $\forall i \in [1\,..\,n]$, P'_i is defined by the following rule:

     if T is relevant to a component in σ_i **then**
     if $P_i \xrightarrow{T} \bar{P}_i \in Pts(\sigma_i)$ **then** $P'_i = \bar{P}_i$ **else** $P'_i = P_{i\,0}$
     else $P'_i = P_i$.

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 record of pattern-space states. A transition marked by T is defined in $Bhv(\wp(\Sigma))$ iff a transition marked by T is defined between the corresponding states of the behavior space. The index $\Im'$ of the new state differs from the index $\Im$ in the old state only if T is visible (according to viewer $\mathcal{V}$). Finally, considering $\mathbf{P}'$ in the new state, three cases are possible:

(a) T is relevant to a component within σ and there exists a transition in $Pts(\sigma_i)$ exiting P_i, marked by T, and entering $\bar{P}_i$: P'_i equals $\bar{P}_i$;

(b) T is relevant to a component within σ and there not exists a transition in $Pts(\sigma_i)$ exiting P_i and marked by T: P'_i equals the initial state $P_{i\,0}$ of $Pts(\sigma_i)$;

(c) T is not relevant to a component within σ: P'_i equals P_i.

The rationale of case (b) is that the current pattern recognition fails to match any fault in σ_i, thereby causing the recognition process to start anew, at the initial state $P_{i\,0}$. As to case (c), state P_i does not change because transition T is relevant to a component outside system σ_i, thereby being irrelevant to pattern recognition within $Pts(\sigma_i)$.

[3] In practice, since $Bsp(\Sigma, \Sigma_0)$ is assumed to be unavailable, it means that T is triggerable from state β.

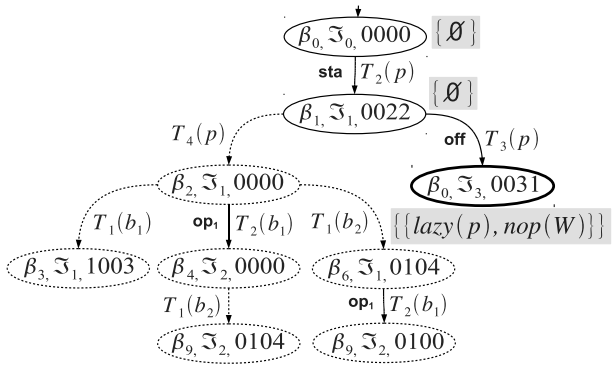

Figure 6: Behavior $Bhv(\wp(\mathcal{W}))$.

Example 9 Consider problem $\wp(\mathcal{W}) = (\mathcal{W}_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ defined in Example 8. Depicted in Fig. 6 is the relevant behavior $Bhv(\wp(\mathcal{W}))$ (node decoration will be explained shortly). According to the definition, the initial state is $(\beta_0, \Im_0, \mathbf{P_0})$, where $\Im_0$ is the initial state of $Isp(\mathcal{O})$ (Fig. 4), while $\mathbf{P_0} = (0, 0, 0, 0)$ is the record of the initial states of $Pts(b_1)$, $Pts(b_1)$, $Pts(p)$, and $Pts(\mathcal{W})$, respectively. Since the following conditions hold:

- $Bsp(\wp(\Sigma))$ (Fig. 3) includes a transition $\beta_0 \xrightarrow{T_2} \beta_1$ which is visible via label *sta* of viewer $\mathcal{V}$,
- A transition $\Im_0 \xrightarrow{sta} \Im_1$ is included in $Isp(\mathcal{O})$,
- Transition $T_2(p)$ is relevant neither to b_1 nor to b_2,
- $P_0 \xrightarrow{T_2(p)} P_2$ is included in both $Pts(p)$ and $Pts(\mathcal{W})$,

a transition $(\beta_0, \Im_0, 0000) \xrightarrow{T_2(p)} (\beta_1, \Im_1, 0022)$ is generated in $Bhv(\wp(\mathcal{W}))$. The construction of the behavior continues until no new node is generated by the application of the transition function. In Fig. 6, state $(\beta_0, \Im_3, 0031)$ is final (in bold), as $\Im_3$ is final in $Isp(\mathcal{O})$. The dashed 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 eventually removed, leaving $Bhv(\wp(\mathcal{W}))$ with three states only. $\diamond$

Once reconstructed the behavior $Bhv(\wp(\Sigma))$, the diagnosis engine is required to generate the problem solution $\Delta(\wp(\Sigma))$ defined in eqn. (5) by means of a sound and complete (possibly efficient) technique. This is carried out by decorating the nodes of the behavior with sets of faults as follows.

The *fault set* of $\mathbf{P}$, written $\delta_{\mathbf{P}}$, is the union of the faults associated with each state of $\mathbf{P}$ in the corresponding pattern 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:

- The initial state $S_0 = (\Sigma_0, \Im_0, \mathbf{P_0})$ is first decorated by the singleton $\Delta(S_0) = \{\delta_{\mathbf{P}_0}\}$.
- For each transition $S \xrightarrow{T} S'$ in $Bhv(\wp(\Sigma))$, where $S = (\beta, \Im, \mathbf{P})$, $S' = (\beta', \Im', \mathbf{P}')$, the decoration of S' is defined by the following set-containment relationship:

$$\Delta(S') \supseteq \{\, \delta' \mid \delta \in \Delta(S),\ \delta' = \delta \cup \delta_{\mathbf{P}'} \,\}. \qquad (7)$$

The *decoration algorithm* starts by marking the initial state with a single diagnosis $\delta_{\mathbf{P}_0}$ including the set of faults associated with the initial state $\mathbf{P_0}$. 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 (7) 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 record $\mathbf{P}'$ of pattern-space states with associated faults $\delta_{\mathbf{P}'}$, 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 patterns recognized in state $\mathbf{P}'$. The algorithm stops when the decoration becomes stable (the application of (7) will no longer produce any changes). Theorem 1 states how the actual solution of $\wp(\Sigma)$ can be distilled from the decorated behavior.

Theorem 1 *Let $Bhv(\wp(\Sigma))$ be the behavior of a diagnosis problem $\wp(\Sigma)$. The union of the set of sets of faults decorating the final states of $Bhv(\wp(\Sigma))$ is the solution of $\wp(\Sigma)$.*

Proof. (*Sketch*) The proof is grounded on Lemma 1.1.

Lemma 1.1 *Let π be a path in $Bhv(\wp(\Sigma))$, from the initial state to a final state. Let h be the list of component transitions marking the arcs of π. Let δ be the union of the fault sets relevant to states of π. Then, h is a history in $Bsp(\Sigma, \Sigma_0)$ such that $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$ and $h_{[\mathcal{R}]} = \delta$.*

Proof. The fact that h is a history in $Bsp(\Sigma, \Sigma_0)$ such that $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$ derives from the mode in which fields β and $\Im$ are generated, specifically, from conditions 1 and 2 of transition-function of $Bhv(\wp(\Sigma))$.

To prove $h_{[\mathcal{R}]} = \delta$, on the one hand, assume $F \in \delta$, that is, $F \in \delta_{\mathbf{P}}$, where $(\beta, \Im, \mathbf{P})$ is a state in π. As such, F is a fault associated with a (final) state of a pattern space $Pts(\sigma)$. Since $Pts(\sigma)$ is constructed by merging the deterministic automata relevant to the unfolded regular expressions involved in fault patterns, based on rule 3 of transition-function definition for $Bhv(\wp(\Sigma))$, it means that a string $e \in \|E\|$ has been recognized, where (F, E) is a fault pattern relevant to subsystem σ, and $e \subseteq h_{[\sigma]}$. Hence, based on eqn. (4), $F \in h_{[\mathcal{R}]}$.

On the other hand, assume $F \in h_{[\mathcal{R}]}$, where h is a history in $Bhv(\wp(\Sigma))$. Based on eqn. (4), F is the fault associated with an expression E, relevant to a subsystem σ, such that there exists a string e in $\|E\|$ that is a sublist of the projection of h on σ. In other words, the projection of h on σ contains a list of transitions that is a string relevant to the regular expression associated with F. Consider condition 3 in the definition of the transition function for $Bhv(\wp(\Sigma))$, and the corresponding rule for the specification of $\mathbf{P}'$. If T is relevant to a component in σ then the state of the prefix space $Pts(\sigma)$ in $\mathbf{P}$ changes either to the target state of the transition marked by T in $Pts(\sigma)$ or to the initial state of $Pts(\sigma)$, depending on whether or not such a transition exists in $Pts(\sigma)$. Since the language of $Pts(\sigma)$ contains the language of expression E (relevant to fault F), the string e will be matched within $Pts(\sigma)$, where the state reached upon such recognition is marked (at least) by F. Hence, $F \in \delta$. Since $F \in \delta \Leftrightarrow F \in h_{[\mathcal{R}]}$, we conclude $h_{[\mathcal{R}]} = \delta$. This terminates the proof of Lemma 1.1.

To prove Theorem 1, denoting with S_{f} a final state of $Bhv(\wp(\Sigma))$, we have to show $\delta \in \Delta(S_{\mathrm{f}}) \Leftrightarrow \delta \in \Delta(\wp(\Sigma))$.

Assume $\delta \in \Delta(S_f)$. Based on the (two) decoration rules for $Bhv(\wp(\Sigma))$, δ is obtained by (at least) one history h by extending the initial set of faults $\delta_{\mathbf{P}_0}$ by the set $\delta_{\mathbf{P}'}$ associated with each state $S' = (\beta', \Im', \mathbf{P}')$ encompassed by h. In other words, δ is the union of the fault sets relevant to states of h. Hence, by virtue of Lemma 1.1, $\delta = h_{[\mathcal{R}]}$ and, based on eqn. (5), $\delta \in \Delta(\wp(\Sigma))$.

Assume $\delta \in \Delta(\wp(\Sigma))$. Based on eqn. (5), $\delta = h_{[\mathcal{R}]}$. By virtue of Lemma 1.1, δ equals the union of the faults sets relevant to the states encompassed by h in $Bhv(\wp(\Sigma))$. Following the same reasoning on the decoration rules above, this union is in fact an element of the decoration of the (final) state S_f of $Bhv(\wp(\Sigma))$ reached by h, hence, $\delta \in \Delta(S_f)$. □

Example 10 With reference to the behavior $Bhv(\wp(\mathcal{W}))$ depicted in Fig. 6, the processing performed by the decoration algorithm is very simple, as the behavior is linear.[4] According to the first decoration rule, the initial state is marked by the singleton $\{\emptyset\}$, as $\delta_{0000} = \emptyset$. Applying the second rule, state $(\beta_1, \Im_1, 0022)$ is decorated by $\{\emptyset\}$, as $\delta_{0022} = \emptyset$. For the final state $S_f = (\beta_0, \Im_3, 0031)$, the decoration is the singleton $\{\{lazy(p), nop(\mathcal{W})\}\}$, which, based on Theorem 1, equals the solution $\Delta(\wp(\mathcal{W}))$, as confirmed by Example 8. ◇

8 Related Work

This paper is built upon [Lamperti and Zanella, 2010]. A related approach is proposed in [Jéron *et al.*, 2006], where the notion of *supervision pattern* is introduced, which allows for a flexible specification of the diagnosis problem, and for an 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, [Jéron *et al.*, 2006] 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 abstraction hierarchy, where diagnosis rules are defined for each subsystem in the hierarchy.

9 Conclusion

Context-sensitive diagnosis is bound to enhance the expressive power of diagnosis of complex DESs. However, we do not consider the proposed notation for pattern specification as final: different formalisms can be envisaged to fit different classes of DESs. Maybe, the general approach is of benefit to diagnosis of complex systems other than DESs, from static systems to more general dynamic systems [Struss, 1997].

[4] Generally speaking, based on eqn. (7), the decoration algorithm is required to visit the nodes of the behavior several times.

References

[Aho *et al.*, 2006] A. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers – Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, second edition, 2006.

[Baroni *et al.*, 1999] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110(1):135–183, 1999.

[Brand and Zafiropulo, 1983] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of ACM*, 30(2):323–342, 1983.

[Cassandras and Lafortune, 1999] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, volume 11 of *The Kluwer International Series in Discrete Event Dynamic Systems*. Kluwer Academic Publisher, Boston, MA, 1999.

[Chomsky, 1956] N. Chomsky. Three models for the description of language. *Transactions on Information Theory*, 2(3):113–124, 1956.

[Debouk *et al.*, 2000] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 10:33–86, 2000.

[Jéron *et al.*, 2006] T. Jéron, H. Marchand, S. Pinchinat, and M.O. Cordier. Supervision patterns in discrete event systems diagnosis. In *Seventeenth International Workshop on Principles of Diagnosis – DX'06*, pages 117–124, Peñaranda de Duero, E, 2006.

[Lamperti and Zanella, 2003] G. Lamperti and M. Zanella. *Diagnosis of Active Systems – Principles and Techniques*, volume 741 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publisher, Dordrecht, NL, 2003.

[Lamperti and Zanella, 2010] G. Lamperti and M. Zanella. Injecting semantics into diagnosis of discrete-event systems. In *21st International Workshop on Principles of Diagnosis – DX'10*, pages 233–240, Portland, OR, 2010.

[Pencolé and Cordier, 2005] Y. Pencolé and M.O. Cordier. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164:121–170, 2005.

[Qiu and Kumar, 2006] W. Qiu and R. Kumar. Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 36(2):384–395, 2006.

[Sampath *et al.*, 1996] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.

[Struss, 1997] P. Struss. Fundamentals of model-based diagnosis of dynamic systems. In *Fifteenth International Joint Conference on Artificial Intelligence – IJCAI'97*, pages 480–485, Nagoya, J, 1997. Morgan Kaufmann, S. Francisco, CA.