Injecting Semantics into Diagnosis of Discrete-Event Systems

Gianfranco Lamperti [1], **Marina Zanella** [1]

[1] *Dipartimento di Ingegneria dell'Informazione, Brescia, 25123, Italy*
lamperti@ing.unibs.it, zanella@ing.unibs.it

ABSTRACT

Most modern approaches to diagnosis of discrete-event systems (DESs) are syntax-oriented. DESs are modeled as networks of communicating automata, where each automaton defines both normal and faulty behavior of one component: the syntax of its regular language. Faulty behavior is associated with a subset of transitions (or events) of the automaton. An evolution of the system (conforming to the observation) is classified as faulty when it involves at least one of such faulty transitions (or events). Consequently, the nature of the system behavior (normal or faulty) strictly conforms to the nature of the behavior of its components. This paper claims that syntax-oriented diagnosis suffers from limited expressiveness when applied to complex DESs. Since a complex DES is topologically organized in a hierarchy of subsystems, different (possibly independent) abstraction levels of diagnosis are required. To overcome the limitations of syntax-oriented diagnosis, a new approach is proposed, based on semantics. A set of semantic rules is specified on a semantic domain (the set of subsystems relevant to diagnosis). Each rule defines the faulty behavior of a subsystem, possibly depending on the behavior of other subsystems. The diagnostic output is a set of candidate diagnoses, which account for the faults of every subsystem in the semantic domain.

1 INTRODUCTION

Most modern approaches to diagnosis of discrete-event systems (DESs) stem from the pioneer work of (Sampath *et al.*, 1996). Basically, the diagnostic approach is to model the DES as a network of communicating automata, with each automaton representing the behavior of one component. The synchronous composition of these automata gives rise to the system model (and, eventually, to the *diagnoser*) which guides the diagnosis of the system based on the given observation. A considerable amount of related research on diagnosis of DESs has been flourishing since then, including (Baroni *et al.*, 1998; 1999; Debouk *et al.*, 2000a; 2000b; Pencolé *et al.*, 2001; Lamperti and Zanella, 2003; Pencolé and Cordier, 2005). Despite the different goals of these works (asynchronism, incrementality, decentralization, distribution, uncertainty, etc.), all of them retained the syntax-oriented nature of diagnosis. Since the system model is a (possibly huge) automaton, the set of possible trajectories of the system (called *histories* from now on) is in fact a regular language (Aho *et al.*, 2006) whose syntax is specified by the communicating automata and the mode in which they are connected to one another in the topological network of the system. Assuming a-posteriori diagnosis, and without loss of generality, the diagnosis problem ultimately reduces to finding out all possible histories of the system which are consistent with the given observation. Depending on the approach, a candidate diagnosis is either the set of faults associated with faulty transitions (or events) of a history, or the history itself. At any rate, the nature of diagnosis is syntax-oriented because it is the syntax (communicating automata and system topology) that defines faults *at component level*. The claim of this paper is that associating faults with transitions (or events) at component level is too a restrictive approach when complex DESs are involved. This claim is consistent with the approach in (Jéron *et al.*, 2006), where the notion of *supervision pattern* is proposed, which allows for a flexible specification of the diagnosis problem, and for an elegant and uniform solution of several classes of problems defined in the literature, including permanent faults, intermittent faults, multiple faults, and sequences of faults. A comparison with (Jéron *et al.*, 2006) is outlined in Section 5.

2 COMPLEX DISCRETE-EVENT SYSTEMS

A *complex DES* is organized as a hierarchy of subsystems, where the root corresponds to the whole system, leaves to components, and intermediate nodes to subsystems. Since in syntax-oriented diagnosis faults are defined at component level, there is no possibility to provide a hierarchy of diagnoses adhering to the hierarchy of the system. Trivially, a subsystem is faulty iff it includes a faulty component. While this approach may be adequate for simple systems, it is not at all satisfactory when applied to complex systems. In fact, the behavior of a complex system can be regarded as

normal even when some components in it are faulty (we call this scenario *positive paradox*). Conversely, the system can be regarded as faulty even when all its components behave normally (*negative paradox*).

2.1 Positive Paradox

To understand positive paradox, consider Example 1.

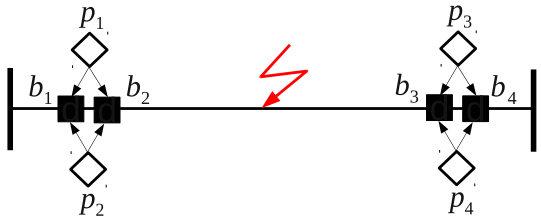

Figure 1: Protected power transmission line.

Example 1. Shown in Fig. 1 is a simplified representation of a power transmission line. The line is protected on both sides by a redundant architecture involving two protections and two breakers (for instance, in the left-hand side these are p_1, p_2, and b_1, b_2, respectively). 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) architecture would incorporate a single protection and a single breaker for each side of the line. Therefore, when one of the two devices is faulty, the line cannot be isolated. By contrast, in the redundant architecture shown in Fig. 1, it suffices the normal behavior of one protection and one breaker (for each side) to guarantee the isolation of the line. Consider the following scenario. On the left-hand side, p_1 and b_1 are faulty, while p_2 and b_2 are normal. On the right-hand side, p_4 is faulty, while p_3, b_3, and b_4 are normal. What is the actual diagnosis of the system? In the syntax-based approach, this is the set of components $\{p_1, b_1, p_4\}$. Note how, owing to redundancy, the behavior of the whole system is in fact normal, as the line is isolated, despite the faulty behavior of a number of components. Now consider a second scenario, where the faulty components are b_1, b_2, and p_4. In this case, the left-hand side fails to open, thereby causing the failure of the protected line. The relevant syntax-oriented diagnosis is $\{b_1, b_2, p_4\}$. Albeit the two diagnoses differs in one component only (p_1 vs. b_2), the behavior of the protected line in the first scenario is normal, while it is faulty in the second one. However, the given diagnoses do not explicitly account for such distinction. More generally, we can consider several subsystems of the protected line and require for each of them a relevant diagnosis. For instance, we can define the following hierarchy: Σ is the whole protected line, σ_ℓ is the protection hardware on the left-hand side, and σ_r is the protection hardware on the right-hand side. In a semantics-oriented setting, the diagnosis of the first scenario is $\{p_1, b_1, p_4\}$, while in the second scenario the diagnosis is $\{b_1, b_2, p_4, \sigma_\ell, \Sigma\}$. Comparing the two diagnoses, we conclude that in the first scenario three components are faulty but their misbehavior is not propagated to higher subsystems. Instead, in the second scenario, besides the three faulty components, σ_ℓ and Σ are faulty too. □

What makes semantic diagnosis possible in the previous example is some *inference rule* that establishes when a (sub)system is to be considered faulty based on the behavior of some other subsystems.

Example 2. The set of inference rules for Example 1 is the following:

$$\begin{aligned} \sigma_\ell &\leftarrow (p_1 \wedge p_2) \vee (b_1 \wedge b_2) \\ \sigma_r &\leftarrow (p_3 \wedge p_4) \vee (b_3 \wedge b_4) \\ \Sigma &\leftarrow \sigma_\ell \vee \sigma_r \end{aligned} \tag{1}$$

where the head (left-hand side) of each rule is the identifier of a subsystem, while the tail (right-hand side) is a formula of predicate calculus involving Boolean variables whose value is true iff the corresponding subsystem is faulty. For instance, the last rule establishes that Σ is faulty when either σ_ℓ or σ_r is faulty. □

2.2 Negative Paradox

Now we turn our attention to the negative paradox: a complex system can be faulty despite the normal behavior of all its components. A software system can be faulty even if all its software components are bug-free. A society can be bound to dictatorship notwithstanding all its democratic institutions. This apparent paradox stems from complexity itself: generally speaking, the behavior of a complex system is uncertain in nature and cannot be completely foreseen based on the behavior of its components.

Considering to the domain of formal languages (to which regular languages belong), we can instantiate the negative paradox borrowing the notion of semantic analysis performed in compilers (Aho *et al.*, 2006). Typically, the syntax of a programming language is defined by a context-free grammar in BNF notation.

Example 3. To specify the *if-then* statement of an imperative language, the following syntax rule can be defined:

$$\textit{if-stat} \rightarrow \textbf{if}\ \textit{expression}\ \textbf{then}\ \textit{block}$$

where *expression* and *block* are nonterminal symbols (language abstractions) representing a Boolean condition and a list of statements, respectively. However, since the grammar is context-free, a sentence of the language can be correct from the syntax viewpoint, but faulty in its semantics. For instance, we can write:

$$\textbf{if}\ \ v\ \ \textbf{then}\ \ x = y$$

where v is a variable. If v is not of Boolean type (e.g., a vector of strings), the above sentence is faulty despite its being syntactically correct. □

The point is, there exist several language constraints that cannot be captured by the syntax. For example, it is impossible to force the calling of a function having the actual parameters that comply (in number, order, and type) with the corresponding formal parameters. Thinking of it in terms of diagnosis, we conclude that a

sentence may be faulty even if all its components (subsentences) are correct on their own. In terms of the regular language of a DES, this translates to the claim that a sentence (history) of the DES can be faulty even if all its subsentences (histories of components) are normal. To cope with negative paradoxes of DESs, we need to enrich the specification of the faulty behavior by means of some *fault patterns*, each one relevant to a subsystem of the semantic domain. A fault pattern is a regular expression on the alphabet composed of the transitions of components included in the subsystem. Since a regular expression can be always represented as a deterministic finite automaton, fault patterns are in fact automata with one initial state and a set of final states. Each path (phrase) within the automaton represents a possible way in which the subsystem may misbehave. Each fault pattern is associated with a *fault*. If the subsystem performs a history belonging to the regular language of the fault pattern, it is considered as faulty in terms of the associated fault. Once fault patterns are defined (this is performed off-line, at system specification), solving a diagnosis problem (based on the actual temporal observation) requires some sort of *pattern recognition*, aimed at uncovering fault patterns in the reconstructed system behavior. Therefore, two phases are envisaged for the diagnostic process:

- *Semantic specification* (off-line) : semantic rules are defined (pattern rules and inference rules);
- *Semantic analysis* (on-line) : fault patterns are detected and, possibly, inference rules applied for each subsystem in the semantic domain.

Besides, after semantic specification, a preprocessing activity is performed (off-line), which generates further graph-based information as a surrogate of semantic rules, in order to speed up the (on-line) semantic analysis when solving the diagnosis problem.

3 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 to the external 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 4-tuple

$$\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{S}) \tag{2}$$

where Σ_0 is the initial state of the system, $\mathcal{V}$ the *viewer*, $\mathcal{O}$ the *observation*, and $\mathcal{S}$ the *semantics*. The viewer maps each component transition to a visible label, thereby establishing how transitions are visible. However, if the label is ϵ (null label), it means that the transition is invisible. The observation is a directed acyclic graph where nodes are marked by observable labels and arcs denote (partial) temporal precedence between nodes.

Example 4. Displayed on the left-hand side of Fig. 2 is an observation $\mathcal{O}$ composed of three nodes and two arcs. Node ω_2 is uncertain because it includes two labels, y and ϵ. □

An observation implicitly embodies several *candidate traces*, denoted $\|\mathcal{O}\|$, each one obtained by picking up a label from each node of the observation without violating the temporal constraints imposed by arcs.

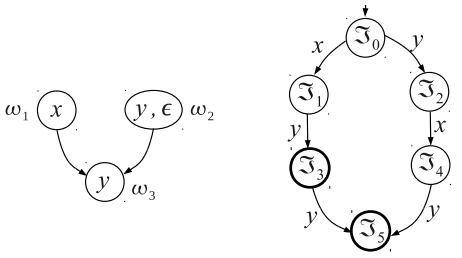

Figure 2: Observation $\mathcal{O}$ and index space $Isp(\mathcal{O})$.

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 $Isp(\mathcal{O})$ is $\|\mathcal{O}\|$, in other words, the set of paths in $Isp(\mathcal{O})$ equals the set of candidate traces of $\mathcal{O}$.

Example 5. Shown on the right-hand side of Fig. 2 is the index space of observation $\mathcal{O}$ (displayed on the left-hand side). The initial state of $Isp(\mathcal{O})$ is $\Im_0$, while two final states (in bold) are included: $\Im_3$ and $\Im_5$. The language of $Isp(\mathcal{O})$ is $\{xy, xyy, yxy\}$, in fact, $\|\mathcal{O}\|$. □

The semantics $\mathcal{S}$ is a pair $(\mathcal{D}, \mathcal{R})$, where $\mathcal{D}$ is the *semantic domain*, that is, the set of subsystems of Σ which are relevant to diagnosis, while $\mathcal{R}$ is the sequence of *semantic rules*. Each rule $R \in \mathcal{R}$ is a triple $(\sigma, \mathcal{P}, \mathcal{F})$, where $\sigma \in \mathcal{D}$ and $\mathcal{F}$ is a *fault*. $\mathcal{P}$ is either a *fault pattern* or a *fault inference*, the latter being a formula of predicate calculus. Correspondingly, R is either a *pattern rule* or an *inference rule*.

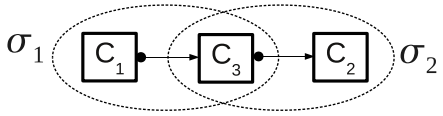

Figure 3: System Σ, including subsystems σ_1 and σ_2.

Example 6. Outlined in Fig. 3 is the topology of a system Σ embodying components C_1, C_2, and C_3. The system is composed of two overlapping subsystems, σ_1 and σ_2, sharing component C_3. We define the semantic domain $\mathcal{D} = \{\Sigma, \sigma_1, \sigma_2\}$. □

3.1 Pattern Rules

In a pattern rule $(\sigma, \mathcal{P}, \mathcal{F})$, pattern $\mathcal{P}$ is specified as a regular expression on the alphabet of transitions of components in subsystem σ.[1] Since a regular expression can be represented by a deterministic automaton with the same language, for processing reasons, $\mathcal{P}$ is translated (off-line) to the equivalent automaton.

Example 7. With reference to the semantic domain defined in Example 6 (Fig. 3), we specify one fault pattern for σ_1, one for σ_2, and two fault patterns for system Σ. For the sake of simplicity, we assume that

[1] More precisely, the fault pattern can be specified based on a *regular definition* (Aho *et al.*, 2006), namely a list of pairs (N, E), where N is a name and E is a regular expression possibly involving names of previous pairs.

models of components C_1, C_2, and C_3 involve just one transition each, namely T_1, T_2, and T_3, respectively. Consequently, the alphabet of the regular expression of σ_1, σ_2, and Σ are $\{T_1, T_3\}$, $\{T_2, T_3\}$, and $\{T_1, T_2, T_3\}$, respectively. For σ_1, the rule is $(\sigma_1, \mathcal{P}_a, a)$, where $\mathcal{P}_a = T_3T_3T_3^*$ (notice how $\mathcal{P}_a$ involves T_3 only). For σ_2, the rule is $(\sigma_2, \mathcal{P}_b, b)$, where $\mathcal{P}_b = T_3T_2$. For Σ, the rules are $(\Sigma, \mathcal{P}_c, c)$, where $\mathcal{P}_c = T_1T_2T_2^*$, and $(\Sigma, \mathcal{P}_d, d)$, where $\mathcal{P}_d = T_1T_3T_2$. The equivalent deterministic automata are displayed in Fig. 4 (where states are identified by capital letters and final states are marked by faults). □

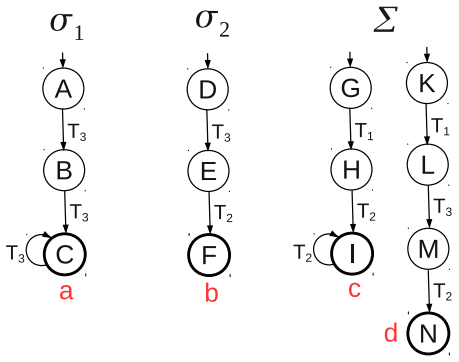

Figure 4: Fault patterns for σ_1, σ_2, and Σ.

Once patterns are represented as deterministic automata, some preprocessing on them is worthwhile. For each (sub)system in the semantic domain, a *pattern space* is generated by merging all the fault patterns relevant to the (sub)system. To this end, each automaton is extended by new empty transitions exiting each non-initial state and entering each initial state of all the automata relevant to the same (sub)system. The purpose of this transformation is to allow the matching of different patterns (of the same (sub)system) exploiting a single automaton only. Specifically, each empty transition entering the initial state of a pattern captures the fact that any new pattern can start at any pattern-matching state, as patterns are in general overlapping.

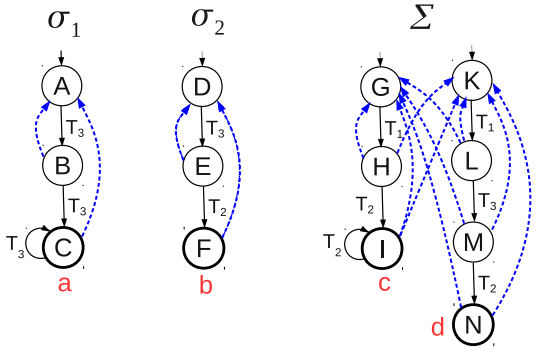

Figure 5: Patterns extended with empty transitions.

Example 8. The extended patterns (automata) relevant to Example 7 (Fig. 4) are displayed in Fig. 5. Notice how empty transitions (represented by dashed lines) are directed towards the initial states of the automata relevant to the same (sub)system. □

Since the introduction of empty transitions gives rise to a nondeterministic automaton, the latter is eventually transformed into an equivalent deterministic automaton. According to the standard determinization algorithm (Aho *et al.*, 2006), each state of the deterministic automaton is identified by a subset of the states of the nondeterministic automaton. The deterministic automaton resulting from the determinization of the fault patterns (extended with empty transitions) relevant to (sub)system σ is called the *pattern space* of σ, written $Pts(\sigma)$. Each final state S of $Pts(\sigma)$ is marked by the faults associated with the states identifying S that are final in the corresponding fault pattern.

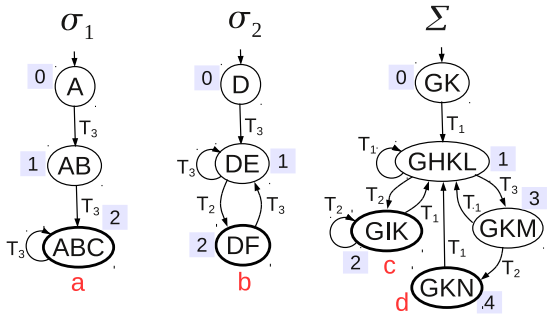

Figure 6: Pattern spaces $Pts(\sigma_1)$, $Pts(\sigma_2)$, and $Pts(\Sigma)$.

Example 9. Depicted in Fig. 6 are the pattern spaces generated from the nondeterministic automata displayed in Fig. 5, namely $Pts(\sigma_1)$, $Pts(\sigma_2)$, and $Pts(\Sigma)$. The initial state of each pattern space is identified by the initial states of the nondeterministic automata of the same (sub)system. Since final states in the fault patterns are C, F, I, and N, the relevant faults a, b, c, and d, respectively, are associated with the final states of the pattern spaces that include either C, F, I, or N. For subsequent easy referencing, states of pattern spaces are identified by numbers. □

The final goal of fault-pattern preprocessing is building the pattern space (deterministic automaton) relevant to the semantic domain $\mathcal{D}$, namely $Pts(\mathcal{D})$. Notice that $Pts(\mathcal{D})$ does not coincide with $Pts(\Sigma)$, as the latter only refers to the fault patterns defined for system Σ, while the former accounts for the fault patterns of all the (sub)systems in the semantic domain $\mathcal{D}$ of $\wp(\Sigma)$. Each state S of $Pts(\mathcal{D})$ is a record of states $(S_1, S_2, \ldots, S_n)$, where each S_i, $i \in [1\,..\,n]$, is a state in the pattern space $Pts(\sigma_i)$. The initial state of $Pts(\mathcal{D})$ is the record of initial states of the pattern spaces $Pts(\sigma_i)$. Denoting with $\mathcal{A}_i$ the *alphabet* of σ_i, a transition

$$(S_1, S_2, \ldots, S_n) \xrightarrow{T} (S'_1, S'_2, \ldots, S'_n)$$

is defined as follows (assuming $i \in [1\,..\,n]$):

- T is the label marking (at least) one transition exiting (at least) one state S_i in $Pts(\sigma_i)$.
- If $T \notin \mathcal{A}_i$, then $S'_i = S_i$ (as the state of the pattern space $Pts(\sigma_i)$ does not change).

- If $S_i \xrightarrow{T} \bar{S}_i \in Pts(\sigma_i)$, then $S'_i = \bar{S}_i$ (as the state of the pattern space $Pts(\sigma_i)$ changes).
- If $T \in \mathcal{A}_i$ and $S_i \xrightarrow{T} \bar{S}_i \notin Pts(\sigma_i)$, then S'_i is the initial state of $Pts(\sigma_i)$ (as the pattern recognition relevant to $Pts(\sigma_i)$ fails, thereby such recognition must be restarted from the initial state of $Pts(\sigma_i)$).

A final state of $Pts(\mathcal{D})$ is such that it includes at least one state S_i which is final in $Pts(\sigma_i)$. Each final state $(S_1, S_2, \ldots, S_n)$ is marked by the union of the set of faults associated with the final states S_i in $Pts(\sigma_i)$.

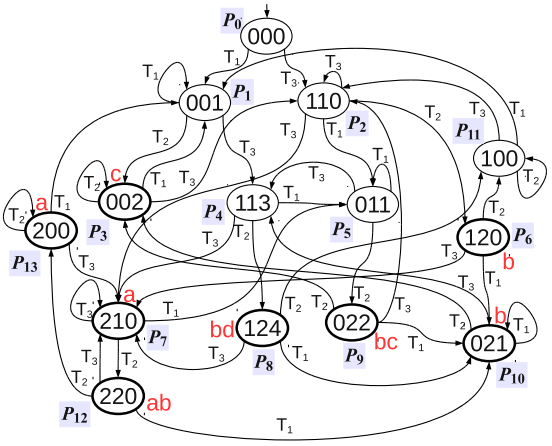

Figure 7: Pattern space $Pts(\mathcal{D})$.

Example 10. Shown in Fig. 7 is the pattern space $Pts(\mathcal{D})$ obtained from the pattern spaces $Pts(\sigma_1)$, $Pts(\sigma_2)$, and $Pts(\Sigma)$ displayed in Fig. 6. Each state of $Pts(\mathcal{D})$ is a triple of numbers identifying a state in $Pts(\sigma_1)$, $Pts(\sigma_2)$, and $Pts(\Sigma)$, respectively. Accordingly, the initial state is 000. Since the transitions exiting the initial states of $Pts(\sigma_1)$, $Pts(\sigma_2)$, and $Pts(\Sigma)$ are marked by T_1 and T_3, we have two transitions exiting the initial state of $Pts(\mathcal{D})$, one marked by T_1 and the other by T_3. Considering T_1, the target state is 001. In fact, since T_1 is in the alphabet of $Pts(\sigma_1)$ but does not mark any transition exiting the initial state of $Pts(\sigma_1)$, it follows that S'_1 is the initial state of $Pts(\sigma_1)$, namely 0. Then, since T_1 is not in the alphabet of $Pts(\sigma_2)$, the target state S'_2 keeps being 0. Finally, since T_1 exits the initial state of $Pts(\Sigma)$, S'_3 is the target state 1. Considering T_3, which belongs to the alphabet of all three pattern spaces, the target state is 110, where 1 is the state reached in both $Pts(\sigma_1)$ and $Pts(\sigma_2)$, while 0 is still the initial state of $Pts(\Sigma)$, as no transition exiting the initial state of the latter is marked by T_3. Once generated the complete pattern space $Pts(\mathcal{D})$, the final states are marked by faults as specified above. For instance, state 220 is marked by faults a and b because these faults mark state 2 in pattern spaces $Pts(\sigma_1)$ and $Pts(\sigma_2)$, respectively. □

Identifiers of states of $Pts(\mathcal{D})$ will be referenced in the reconstruction of the system behavior performed by the semantic diagnostic engine, as formalized in Section 4. The reconstruction of the system behavior accounts for pattern-matching by associating with each reconstructed system state a state of the pattern space $Pts(\mathcal{D})$. Since a state of $Pts(\mathcal{D})$ indicates the state of pattern recognition, when a final state of $Pts(\mathcal{D})$ is reached, at least one pattern is recognized.

3.2 Inference Rules

Within an inference rule $(\sigma, \mathcal{P}, \mathcal{F})$, $\mathcal{P}$ is a formula of predicate calculus. Fault $\mathcal{F}$ occurs for subsystem σ iff $\mathcal{P}$ evaluates to true based on the logic values of variables in $\mathcal{P}$. These variables are represented by the identifiers of faults $\mathcal{F}'$ relevant to other pattern and inference rules. A variable $\mathcal{F}'$ is true iff fault $\mathcal{F}'$ occurs in the corresponding subsystem.

Example 11. With reference to the diagnostic domain defined in Example 6 and the fault patterns in Example 7, we specify two inference rules for system Σ, namely $(\Sigma, \mathcal{P}_e, e)$ and $(\Sigma, \mathcal{P}_f, f)$, as follows:

$$\begin{aligned} e &\leftarrow a \wedge b \wedge d \\ f &\leftarrow e \vee c \end{aligned} \tag{3}$$

This means that fault e occurs in Σ if faults a, b, and c occur altogether, while fault f is generated by the occurrence of either e or c. □

In order to prevent circularity, inference rules are supposed to be well-formed: for each variable $\mathcal{F}$ involved in the logic formula of a rule $\mathcal{R}$ there exists a rule $\mathcal{R}' = (\sigma, \mathcal{P}, \mathcal{F})$ defined before $\mathcal{R}$.

A *dependency graph* is defined as follows. Since each inference rule $R = (\sigma, \mathcal{P}, \mathcal{F})$ defines the logic value of fault $\mathcal{F}$ based on the logic values of the faults involved in formula $\mathcal{P}$, there exists a dependency of $\mathcal{F}$ on all the faults involved in $\mathcal{P}$. This can be represented by an *inference graph*, where nodes are faults ($\mathcal{F}$ and those in $\mathcal{P}$) while arcs are directed from each fault in $\mathcal{P}$ to $\mathcal{F}$. The composition of the inference graphs of all rules gives rise to the dependency graph of the semantics $\mathcal{S}$, namely $Dgr(\mathcal{S})$.

Example 12. Considering the inference rules defined in (3), the corresponding dependency graph is displayed in Fig. 8. □

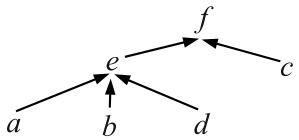

Figure 8: Dependency graph.

A dependency graph can be evaluated on a set of faults δ, denoted $Eval(Dgr(\mathcal{S}), \delta)$, as follows:

- A leaf node $\mathcal{F}$ is assigned a Boolean value based on its membership to δ: if $\mathcal{F} \in \delta$ then $\mathcal{F}$ is assigned to true, otherwise it is assigned to false.
- The assignment of each intermediate node $\mathcal{F}'$ can be performed (based on the corresponding logic formula) iff all its child nodes have been assigned.
- When all nodes have been assigned a value, the result of the evaluation consists of the union of δ and the set of faults assigned to true in $Dgr(\mathcal{S})$.

Example 13. With reference to the dependency graph outlined in Fig. 8, assuming $\delta = \{a, b, d\}$, we have $Eval(Dgr(\mathcal{S}), \delta) = \{a, b, d, e, f\}$, where e and f are the *derived faults*. □

The evaluation of $Dgr(\mathcal{S})$ based on a set of faults δ allows the diagnostic engine to provide the complete set of faults of each candidate diagnosis starting from the faults associated with pattern rules. In fact, uncovering *base faults* associated with patterns requires the diagnostic engine to reconstruct the system behavior, which is not the case for derived faults.

3.3 Diagnosis-Problem Solution

Roughly, the solution of a diagnosis problem $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{S})$, written $\Delta(\wp(\Sigma))$, is a set of *candidate diagnoses*, where each candidate diagnosis is a (possibly empty) set of faults specified in the semantic rules.

To define $\Delta(\wp(\Sigma))$, we need to introduce a few notation. The automaton representing all possible histories of Σ starting from Σ_0 is the *behavior space* of Σ, denoted $Bsp(\Sigma, \Sigma_0)$. 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 subsequence of transitions of h relevant to components in σ. The regular language of a fault pattern $\mathcal{P}$ is denoted $\|\mathcal{P}\|$.

Assuming $\mathcal{R}$ to be the semantic rules in $\mathcal{S}$ and $\mathcal{P}$ a fault pattern, the solution of $\Delta(\wp(\Sigma))$ is defined by the following set of diagnoses:

$$\{\ \bar{\delta} \mid h \in Bsp(\Sigma, \Sigma_0), h_{[\mathcal{V}]} \in \|\mathcal{O}\|, \quad \delta = \{\mathcal{F} \mid (\sigma, \mathcal{P}, \mathcal{F}) \in \mathcal{R}, P \in \|\mathcal{P}\|, P \subseteq h_{[\sigma]}\}, \quad \bar{\delta} = \delta \cup Eval(Dgr(\mathcal{S}), \delta)\ \}. \tag{4}$$

In other words, a candidate diagnosis $\bar{\delta}$ is the evaluation of the diagnostic graph $Dgr(\mathcal{S})$ based on δ. The latter is the set of faults $\mathcal{F}$, associated with a fault pattern $\mathcal{P}$ for subsystem σ, such that there exists a path P in $\mathcal{P}$ that is a subsequence of the projection on σ of a history h (formally, $P \subseteq h_{[\sigma]}$) whose trace $h_{[\mathcal{V}]}$ is a candidate trace of observation $\mathcal{O}$.

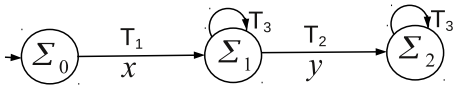

Figure 9: Behavior space $Bsp(\Sigma, \Sigma_0)$.

Example 14. We focus on $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{S})$ for system Σ (Fig. 3). We assume viewer $\mathcal{V}$ such that transitions T_1 and T_2 are visible via labels x and y, respectively, while T_3 is invisible. Observation $\mathcal{O}$ is displayed in Fig. 2, while semantics $\mathcal{S}$ is specified in Sections 3.1 and 3.2. We also assume the behavior space displayed in Fig. 9, where visible transitions are associated with relevant labels. The solution of $\wp(\Sigma)$ can be determined based on (4). First, we have to select the histories in $Bsp(\Sigma, \Sigma_0)$ whose trace is in $\|\mathcal{O}\|$, in other words, whose trace is a path in the index space of $\mathcal{O}$ (Fig. 2). Of the three traces indicated in Example 5, only xy is consistent with $Bsp(\Sigma, \Sigma_0)$. However, trace xy can be generated by an infinite set of histories, actually, all the histories ending in state Σ_2 (with any number of iterations of transition T_3). Then, for the same histories h, we have to determine the set δ of faults, where each fault $\mathcal{F}$ is associated with a fault pattern $\mathcal{P}$ for subsystem σ such that a path in $\mathcal{P}$ is contained in the projection of h on σ. By making a pattern-matching between the behavior space in Fig. 9 and the fault patterns in Fig. 4, we come up with the following set of faults (*base diagnoses*):

- $\delta_1 = \{c\}$, generated by histories $T_1T_2T_3^?$, where operator '?' (applied to T_3) means optionality.
- $\delta_2 = \{a, c\}$, generated by histories $T_1T_2T_3T_3T_3^*$, whose projections on σ_1 and σ_2 are $T_1T_3T_3T_3^*$ and $T_2T_3T_3T_3^*$, respectively.
- $\delta_3 = \{a, b\}$, generated by histories $T_1T_3T_3T_2T_3^*$, whose projections on σ_1 and σ_2 are $T_1T_3T_3T_3^*$ and $T_3T_3T_2T_3^*$, respectively.
- $\delta_4 = \{b, d\}$, by histories $T_1T_3T_2T_3^?$, whose projection on σ_2 is $T_3T_2T_3^?$.
- $\delta_5 = \{a, b, d\}$, by histories $T_1T_3T_2T_3T_3T_3^*$, whose projections on σ_1 and σ_2 are $T_1T_3T_3T_3T_3^*$ and $T_3T_2T_3T_3T_3^*$, respectively.

The actual solution of $\wp(\Sigma)$ is determined by evaluating the dependency graph for each base diagnosis, as shown in Example 12, thus obtaining the following set of candidate diagnoses:

$$\{\{c, f\}, \{a, c, f\}, \{a, b\}, \{b, d\}, \{a, b, d, e, f\}\}$$

where e and f are the derived faults. □

The diagnostic engine is expected to provide the same result by associating the reconstruction of the system behavior with the semantic information incorporated in the pattern space of the diagnosis problem.

4 DIAGNOSTIC ENGINE

The diagnostic engine is required to take as input a diagnosis problem $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{S})$ and to output the relevant solution $\Delta(\wp(\Sigma))$, as defined in (4). In so doing, it needs to reconstruct the behavior of Σ that conforms to observation $\mathcal{O}$. We assume that, as explained in Section 3, the pattern space $Pts(\mathcal{D})$ (generated off-line) is available to the engine. Since a sort of semantic analysis (pattern matching) is to be performed, the states of the reconstructed behavior will incorporate information not only on the observation but also on the fault patterns merged in $Pts(\mathcal{D})$. This is why we call it the *semantic behavior* of the diagnosis problem, written $Sbh(\wp(\Sigma))$.

Formally, let $\mathbb{B}$ denote the set of states of $Bsp(\Sigma, \Sigma_0)$, $\mathbb{I}$ the set of states of $Isp(\mathcal{O})$, and $\mathbb{P}$ the set of states of $Pts(\mathcal{D})$. The semantic behavior

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

is a deterministic automaton such that:

- $\mathbb{S} \subseteq \mathbb{B} \times \mathbb{I} \times \mathbb{P}$ is the set of states;
- $S_0 = (\Sigma_0, \Im_0, P_0)$ is the initial state, where $\Im_0$ the initial state of $Isp(\mathcal{O})$ and P_0 the initial state of $Pts(\mathcal{D})$;
- $\mathbb{S}_f = \{(\beta, \Im_f, P) \mid \Im_f \text{ is final in } Isp(\mathcal{O})\}$ is the set of final states;
- $\mathbb{T}$ is the transition function, defined as follows. $(\beta, \Im, P) \xrightarrow{T} (\beta', \Im', P') \in \mathbb{T}$ iff:
  - $\beta \xrightarrow{T} \beta'$ is a transition in $Bsp(\Sigma, \Sigma_0)$,
  - 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 $\mathcal{V}$,

– if $P \xrightarrow{T} \bar{P}$ is a transition in $Pts(\mathcal{D})$ then $P' = \bar{P}$ else $P' = P_0$.

In other words, each node of $Sbh(\wp(\Sigma))$ is a triple involving a state of the behavior space, a state of the index space of $\mathcal{O}$, and a state of the pattern space of $\mathcal{D}$. A transition marked by T is defined in $Sbh(\wp(\Sigma))$ 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 P' in the new state, two cases are possible: either there exists a transition exiting P in $Pts(\mathcal{D})$ and marked by T or there does not. If it exists, then P' is the state reached by such a transition in $Pts(\mathcal{D})$, otherwise P' equals the initial state P_0 of $Pts(\mathcal{D})$ (restarting the recognition of any pattern, formally, $P' = P_0$).

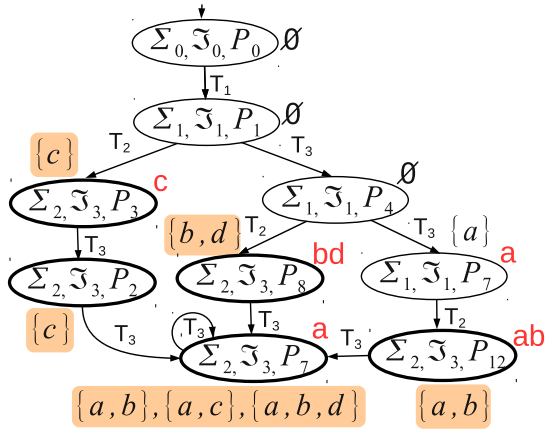

Figure 10: Semantic behavior $Sbh(\wp(\Sigma))$.

Example 15. Consider the diagnosis problem $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{S})$ defined in Example 14. Depicted in Fig. 10 is the relevant semantic behavior $Sbh(\wp(\Sigma))$ (node decoration are explained shortly). According to the definition, the initial state is $(\Sigma_0, \Im_0, P_0)$, where $\Im_0$ is the initial state of $Isp(\mathcal{O})$ (Fig. 2), while P_0 is the initial state of $Pts(\mathcal{D})$ (Fig. 7). Since the following conditions hold:

- Behavior space $Bsp(\wp(\Sigma))$ (Fig. 9) includes a transition $\Sigma_0 \xrightarrow{T_1} \Sigma_1$ which is visible via label x,
- A transition $\Im_0 \xrightarrow{x} \Im_1$ is included in $Isp(\mathcal{O})$,
- A transition $P_0 \xrightarrow{T_1} P_1$ is included in $Pts(\mathcal{D})$,

a transition $(\Sigma_0, \Im_0, P_0) \xrightarrow{T_1} (\Sigma_1, \Im_1, P_1)$ is generated in $Sbh(\wp(\Sigma))$. The construction of $Sbh(\wp(\Sigma))$ continues until no new node is generated by the application of the transition function. The semantic behavior is composed of nine states, five of which are final (in bold in Fig. 10). Furthermore, each state $(\beta, \Im, P)$ such that P is final in $Pts(\mathcal{D})$ is decorated by the set of base faults (from patterns) associated with state P in $Pts(\mathcal{D})$. For instance, $(\Sigma_2, \Im_3, P_8)$ is decorated with faults b and d because such faults are associated with state P_8 in $Pts(\mathcal{D})$ (Fig. 7). □

Once constructed the semantic behavior of $\wp(\Sigma)$, the diagnosis engine is expected to generate the diagnostic solution $\Delta(\wp(\Sigma))$ by means of a sound and complete (possibly efficient) technique. Following the same schema adopted for the definition of $\Delta(\wp(\Sigma))$ given in Section 3.3, this is accomplished in two steps:

- Generation of the set of base diagnoses;
- Completion of each base diagnosis with the relevant derived faults.

The first step is carried out by decorating the nodes of the semantic behavior with sets of base diagnoses as follows. We denote with δ_P the set of base faults associated with state P in $Pts(\mathcal{D})$. We also denote with $\Delta(S)$ the set of base diagnoses decorating state S in $Sbh(\wp(\Sigma))$. Each state in $Sbh(\wp(\Sigma))$ is decorated with a set of faults based on the following rules:

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

$$\Delta(S') \supseteq \{\delta' \mid \delta \in \Delta(S), \delta' = \delta \cup \delta_{P'}\}. \quad (6)$$

The *decoration algorithm* starts by marking the initial state with a single diagnosis δ_{P_0} including the set of base faults associated with the initial state P_0 of $Pts(\mathcal{D})$. 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 (6) is as follows. If the current decoration of a state S of $Sbh(\wp(\Sigma))$ includes diagnosis δ and the reached state S' involves pattern state P' with associated faults $\delta_{P'}$, then the diagnosis 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 P' of $Pts(\mathcal{D})$. The decoration algorithm stops when no further diagnoses are associated with any node (the application of (6) will no longer produce any changes). The set of candidate base diagnoses is the union of the base diagnoses associated with the final nodes.

Example 16. With reference to the semantic behavior in Fig. 10, consider, for instance, the decoration of state $(\Sigma_2, \Im_3, P_7)$. This decoration includes three diagnoses: $\{a, b\}$, $\{a, c\}$, and $\{a, b, d\}$, each of which is generated by propagating the decoration of the initial state through three different transition paths: $\{a, b\}$ by $T_1T_3T_3T_2T_3$, $\{a, c\}$ by $T_1T_2T_3T_3$, and $\{a, b, d\}$ by $T_1T_3T_2T_3$. The actual set of base diagnoses is the union of the decorations associated with final states (shaded in Fig. 10), namely $\{c\}$, $\{a, c\}$, $\{a, b\}$, $\{b, d\}$, and $\{a, b, d\}$. Notice that this set equals the set of base diagnoses determined in Example 14 in accordance with the definition of diagnosis-problem solution. The actual solution $\Delta(\wp(\Sigma))$ is eventually generated by extending each diagnosis δ by the evaluation of the diagnostic graph relevant to $\mathcal{S}$, namely $Eval(Dgr(\mathcal{S}), \delta)$. Since each evaluation functionally depends on base diagnosis δ, the evaluation of the set of base diagnoses gives rise to the same set of candidate diagnoses determined in Example 14, in fact, the solution of the diagnosis problem $\wp(\Sigma)$. □

5 RELATED WORK

The approach to diagnosis of DESs introduced in this paper shares a conceptual commonality with the approach proposed in (Jéron *et al.*, 2006), namely the idea of pattern.[2] However, a number of differences exist. First, a diagnosis in (Jéron *et al.*, 2006) is a trajectory (history) which involves specific events. In our approach, a diagnosis is the set of faults entailed by a history. Second, a supervision pattern specifies which trajectories are to be considered as faulty (or, more generally, significant to the supervision process), based on specific occurrences of fault (and repair) events. In our approach, a fault pattern specifies a fault, not a relevant history. Besides, the formalisms adopted for the specification of patters are regular expressions and logic formulas rather than automata. Although the two formalisms are equivalent from the expressive-power point of view, using operators of regular expressions and predicate calculus rather then composition of automata (based on set-oriented operators of formal languages) provides a considerable advantage to conciseness, readability, and comprehension of patterns. Third, and more importantly, (Jéron *et al.*, 2006) do not provide any hierarchical abstraction to diagnosis: since a diagnosis is a trajectory identified by a supervision pattern, the notion of diagnosis invariably refers to the system as a whole. In our approach, instead, the interpretation of the system behavior is based on a set of semantic rules that differentiate over a specific hierarchy of subsystems (the semantic domain). The essential question: "*What is the meaning of the occurrence of a string of transitions for a given subsystem?*" is answered based on the semantics defined in the diagnosis problem.

6 CONCLUSION

Confining diagnosis to faults defined at component level is too a limiting approach when complex systems are involved. A complex system is organized in a hierarchy of subsystems, corresponding to different abstraction levels. The behavioral nature of each subsystem does not necessarily depends on the misbehavior of its components. Somewhat paradoxically, a complex system may be normal even when some of its components are faulty. It can also be faulty when all its components are normal. As happens in programming languages, where context-free syntax rules lack expressive power in constraining the correct sentences of the language, the syntax of regular languages specified by communicating automata may be inadequate to capture the faulty behavior of the whole system. Thus, shifting the diagnosis paradigm of DESs from syntax to semantics is bound to considerable advantages. In the approach proposed in this paper, this shift consists in extending the diagnosis problem by a semantics $\mathcal{S} = (\Delta, \mathcal{R})$. For each subsystem in the semantic domain $\mathcal{D}$, a number of semantic rules are defined in $\mathcal{R}$. Pattern rules associate fault patterns with base faults. Inference rules specify derived faults by means of formulas of predicate calculus. The two classes of rules allow for negative and positive paradoxes. However, we do not consider the proposed notation for semantic specification as final. On the contrary, different semantic formalisms can be envisaged to fit different classes of DESs. Even for the single class of active systems, for which the proposed approach has been conceived, additional investigation in semantic-oriented diagnosis seems worthwhile for future research.

[2] The notion of supervision pattern introduced in (Jéron *et al.*, 2006) was subsequently adopted by (Ye *et al.*, 2009) to cope with diagnosability by distributed techniques.

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.*, 1998) P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of active systems. In *Thirteenth European Conference on Artificial Intelligence – ECAI'98*, pages 274–278, Brighton, UK, 1998.

(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.

(Debouk *et al.*, 2000a) 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.

(Debouk *et al.*, 2000b) R. Debouk, S. Lafortune, and D. Teneketzis. A diagnostic protocol for discrete-event systems with decentralized information. In *Eleventh International Workshop on Principles of Diagnosis – DX'00*, pages 41–48, Morelia, MX, 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.

(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.

(Pencolé *et al.*, 2001) Y. Pencolé, M.O. Cordier, and L. Rozé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In *Twelfth International Workshop on Principles of Diagnosis – DX'01*, pages 151–158, San Sicario, I, 2001.

(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.

(Ye *et al.*, 2009) L. Ye, P. Dague, and Y. Yan. A distributed approach for pattern diagnosability. In *20th International Workshop on Principles of Diagnosis – DX'09*, pages 179–186, Stockholm, S, 2009.