Inference of fault signatures of discrete-event systems from event logs

Cody Christopher[1] and **Yannick Pencolé**[2] and **Alban Grastien**[1]
[1]CSIRO, Data61 and Australian National University, Canberra, Australia
e-mail: cody.christopher@data61.csiro.au, alban.grastien@data61.csiro.au
[2]LAAS, CNRS, Univ. Toulouse, Toulouse, France
e-mail: yannick.pencole@laas.fr

Abstract

In this paper, we propose a method to diagnose faults in a discrete event system that only relies on past observed logs and not on any behavioural model of the system. Given a set of tagged logs produced by the system, the first objective is to extract from them a set of fault signatures. These fault signatures are represented with a set of critical observations that are the support of the diagnosis method. We first propose a method to compute the fault signatures from an initial log journal and follow with detail on how the signatures can then be updated when new logs are available.

1 Introduction

This paper addresses the problem of fault diagnosis in dynamical discrete event systems (DES) such as communication protocols [1], automated production systems [2], and business work-flows [3], *etc.* The system generates a sequence of events, some of which are observable whilst others are not. Depending on the type of events that is produced (normal/abnormal events) or depending on the way the events are produced (event patterns [4; 5]), the system may change from a *normal mode* to an *abnormal mode*. Given a flow of observations, the problem consists of online analysis of the observation flow and retrieving the current mode of the system. We typically call the generic function that performs the observation and mode analysis of the system a *diagnoser*. The general question is typically about how we go designing such a diagnoser for any given system. In addressing this problem, there are two main difficulties: the *acquisition of knowledge* about the behaviour of the system and the *computational complexity* of the diagnosis problem based on the acquired knowledge.

The first method of acquiring knowledge consists in developing a behavioural model of the system with help of experts. The result is a predictive model that describes how the system behaves (faultless model or not). The main advantage of such a method is that the model describes the physical nature of the underlying system so the knowledge about the system is rich and precise (*white-box model*) [6; 7]. The drawback of this approach is the complexity and cost in the design of such a model. As soon as the system consists of complex parts of varying origin, acquiring knowledge from experts might become prohibitive (high costs, confidentiality reasons, *etc*).

The contrasting method of knowledge acquisition is to use a *data-driven approach*. In this context, the system is known only based on the sequences of observable events that it or similar systems have produced in the past (a *black-box model* consisting of a set of *event logs*) [8]. In the era of BigData, systems now can indeed generate tremendous amounts of data (multiple set of sensors, cyber-physical systems, internet of things). For BigData to succeed we require that the logs obtained are meaningful enough to extract some crucial pieces of information. In the context of health monitoring, the relevant information we seek regards the relationship between the actual current flow of observations and some faulty operation modes. The challenge of a data-driven method is in how we extract this information from event logs and deduce *fault mode signatures* [9].

The second difficulty surrounds the complexity of the diagnosis problem. In both approaches (model-based or data-based), a diagnoser has to complete two tasks—to analyse the current flow of observed events that the system generates and to retrieve the set of possible fault modes that may have occurred. In a typical white-box model the diagnoser is an algorithm that relies on a finite state machine (or set of FSMs) that uses observable events to update the current diagnosis. This approach is advantageous in that it retains the precision of the model, however the time and space complexity of such a diagnoser can be an obstacle.

One approach to reducing the complexity of the algorithm is to consider that many observations generated by the system are not relevant from a diagnosis point of view. Under this assumption, only a subset of observable events within the observation flow are relevant and sufficient to perform diagnosis. Such an approach relies on a *recogniser* [10] that reads the observations flow but only retains events that are part of a given *observable pattern*, often called a *chronicle*[1]. Once a pattern has been recognised, the diagnoser updates its belief about the current mode of the system with the information associated with the recognised pattern.

In this paper, we propose using a *data-driven* approach to compute *observable event patterns* as a solution to the diagnosis problem of DES. Our objectives are:

1. to design a diagnoser of the system without the need for a model that would require expertise, simply based on data logs produced by the system;
2. to abstract away irrelevant information to provide a

[1]The term chronicle usually refers to event patterns where two events are constrained with time intervals $[\alpha, \beta]$. In this work, we do not deal with time intervals, only with event sequence.

diagnoser that performs more efficiently by simply recognising fault signatures represented by *critical observations* [11] and emit the associated diagnosis;

3. to continuously update the set of *critical observations* when more logs are available such that we capture more fault signatures and therefore improve the diagnoser.

The paper is organised as follows. Section 2 introduces the problem formally along with necessary notation. Section 3 recalls the theory of critical observations and describes a method to compute them from a fixed set of event logs. Section 4 then provides an incremental version of this method that updates the set of critical observations when new logs are available. Section 5 finally presents an illustrative example about the proposed method.

2 Problem statement

This section formally introduces the problem that is addressed in this paper.

2.1 Notation

The present work takes place in the context and standard framework of discrete event systems (DES) [12; 6]. A DES is a system that generates *runs*. A system *run* is a finite sequence of events (e), $w = e_1 e_2 \dots e_k$ and Σ denotes the set of events that the system could possibly generate in its runs. The symbol ε denotes the empty sequence. By definition if $e_1 e_2 \dots e_k, k > 1$ is a run of the system, then $e_1 e_2 \dots e_{k-1}$ is also a run of the system. The run ε is a run of any system. It follows that the set of runs of a DES is a prefix-closed language, $\mathcal{S} \subseteq \Sigma^\star$, where $\Sigma^\star$ is the Kleene closure of Σ.

In this framework, we make the assumption that the set of events of the underlying system is partitioned into *observable* events Σ_o—events that are recorded—and unobservable events Σ_u—those that are not. The observation flow o generated by run $w = e_1 e_2 \dots e_k$ will be hereafter called the *trace* of w: it is the projection (P) of w on the set of observable events (i.e., all unobservable events of the run are deleted):

$$o = P_{\Sigma_\text{o}}(w) = \begin{cases} \varepsilon & \text{if } k = 0 \\ e_1 P_{\Sigma_\text{o}}(e_2 \dots e_k) & \text{if } k > 0 \text{ and } e_1 \in \Sigma_\text{o} \\ P_{\Sigma_\text{o}}(e_2 \dots e_k) & \text{otherwise.} \end{cases}$$

The Σ-language of a trace o, denoted $\mathcal{L}_o$, is the set of finite sequences of events from Σ that could produce the observed sequence:$\mathcal{L}_o = P^{-1}_{\Sigma_\text{o}}(o) = \{w \in \Sigma^\star \mid P_{\Sigma_\text{o}}(w) = o\}$.

2.2 Event logs

In this paper, we consider that the language $\mathcal{S}$ of the system is unknown. The only available information we start from is a set of *logs*. A log results from a past run of the system (or a past run of a system that is identical). It is simply the record of the trace of the run with information about the fault class.

Definition 1 (Log)**.** *A log l is a tuple (o, f) such that:*

- *there exists a run $w \in \mathcal{S}$ such that $o = P_{\Sigma_\text{o}}(w)$;*
- *f is a tag of the fault mode and the run w leads the system to be in this fault mode f.*

The notion of fault mode here is generic. First, we denote N as the nominal mode as a specific fault mode. Usually, if a run is in a fault mode $f \neq N$, it means that a fault event $f \in \Sigma$ has occurred in the run. However, a fault in DES can also be the occurrence of a more complex behaviour (denoted pattern in [4; 5]) and f then means that such a pattern has occurred in the run. In the following and without loss of generality, we suppose that if the system is in a fault mode f it means that a fault event $f \in \Sigma_u$ is part of that run of the system. With slight abuse of notation we write $f \in w$ to indicate "f appears in w" (i.e. $w \in \Sigma^\star f \Sigma^\star$).

Whereas it is pretty easy to record the trace o, the association between the trace and its fault class f is not straightforward. There are basically two ways to get this fault class: by expertise, and by data classification. In the case of expertise, the considered trace has already been analysed by a repair agent that tagged the trace with what was actually found within the system (the set of components replaced for instance). The second is to use automated classification techniques to automatically tag the set of traces depending on some distance criteria. This is out of the scope of this paper.

2.3 Diagnosis problem and implementation

We recall the classical diagnosis problem in a DES and describe the proposed diagnoser that solves an approximation of this problem.

As previously introduced, the set of unobservable events of the system includes a subset of fault events, $\Sigma_\text{f} \subseteq \Sigma_\text{u}$. A set $\delta \subseteq \Sigma_\text{f}$ of faults is *consistent* with the system $\mathcal{S}$ and the trace o if there exists a run $w \in \mathcal{S}$ that would produce this trace ($P_{\Sigma_\text{o}}(w) = o$) and that exhibits exactly these faults ($w \cap \Sigma_\text{f} = \delta$). The diagnosis of trace o, denoted $\Delta(o)$, is the collection of all consistent sets of faults:

$$\Delta(o) = \left\{ \delta \subseteq \Sigma_\text{f} \;\middle|\; \begin{array}{l} \exists w \in \mathcal{S}. \\ P_{\Sigma_\text{o}}(w) = o \;\wedge\; \delta = w \cap \Sigma_\text{f} \end{array} \right\} \quad (1)$$

We find it more convenient to define the diagnosis in terms of emptiness of languages. Let $\mathcal{L}_\delta$ be the language that represents all sequences that contain exactly δ:

$$\mathcal{L}_\delta = \{w \in \Sigma^\star \mid w \cap \Sigma_\text{f} = \delta\} = \bigcap_{f \in \delta} \Sigma^\star f \Sigma^\star \cap \bigcap_{f \in \Sigma_\text{f} \setminus \delta} (\Sigma \setminus \{f\})^\star$$

That is, $\mathcal{L}_\delta$ represents the set of all runs containing all of the faults of δ, intersected with all possible runs where the faults not in δ never occur—the result is a set of all runs where the only faults that occur are those in δ. With $\mathcal{L}_\delta$ defined, we can equivalently express the diagnosis as an emptiness of languages problem:

$$\delta \in \Delta(o) \iff \mathcal{S} \cap \mathcal{L}_o \cap \mathcal{L}_\delta \neq \emptyset. \quad (2)$$

The classical definition of the diagnosis $\Delta(o)$ relies on $\mathcal{S}$. In a classical model-based approach, this language $\mathcal{S}$ is assumed to be known. However, in the presented framework, we do not know $\mathcal{S}$ but only a set of logs $L = \{l_1, \dots, l_n\}$.

Assumption 1. *Any log $l = (o, f)$ of L is correct:*

$$(f \neq N \Rightarrow (\exists w \in \mathcal{S}, (P_{\Sigma_\text{o}}(w) = o) \wedge (w \cap \Sigma_\text{f} = \{f\}))) \wedge$$
$$(f = N \Rightarrow (\exists w \in \mathcal{S}, (P_{\Sigma_\text{o}}(w) = o) \wedge (w \cap \Sigma_\text{f} = \emptyset))).$$

The design of the proposed diagnoser only relies on the logs L and is based on critical observations (introduced in § 3.2). A critical observation θ corresponds to an observable language resulting from the abstraction of irrelevant observable events. The objective is to derive from L a set of critical observations $\{\theta_1, \dots, \theta_m\}$ that represent fault signatures. This means that each θ_i is associated with a fault $\delta_i \subseteq \Sigma_\text{f}$ (note that a δ_i can be associated with multiple θ_i). The proposed diagnoser, denoted Δ_L, then works as follows.

Definition 2 (Δ_L diagnoser)**.** *Let o be a trace of the system.*

$$\delta \in \Delta_L(o) \iff ((\{\exists \theta_i \ni o\} \Rightarrow \delta = \delta_i\}). \\ \wedge (\{\forall \theta_i, o \notin \theta_i\} \Rightarrow \delta = \varnothing))$$

Intuitively speaking, the diagnoser Δ_L records a new trace o of the system and checks whether it can find any critical observations θ in its database that o matches (in other words, the diagnoser checks whether θ is *recognised* in o). If o matches θ then the diagnoser returns the fault class δ associated to θ. In the case where o is not recognised by any of the critical observations then Δ_L returns the unknown diagnosis $\varnothing$.

3 Off-line inference of critical observations

The aim of this section is to describe the method that computes a set of critical observations from a given set of logs L. This method first relies on the computation of a specific representation of the log set L as a *log tree*.

3.1 Log tree

A log tree is a representation of a given set of logs L as a finite state machine.

Definition 3 (Log Tree)**.** *The log tree (L-tree) $LgT(L)$ is a tree:*

$$LgT(L) = (Q, \Sigma, \delta, q_0)$$

such that

- *Q is a finite set of nodes;*
- *Σ is a finite alphabet composed of the set of observable events Σ_o and the set of fault events Σ_f;*
- *$\delta : Q \times \Sigma \to Q$ is a transition function (defined below);*
- *q_0 is the root node.*

Let δ^* denote the transitive closure of δ, that is $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, e.w) = \delta^*(\delta(q, e), w)$ for $e \in \Sigma$ and $w \in \Sigma^*$. The transition function δ is defined by the following four conditions.

1. q_0 has no predecessor: $\forall q \in Q, \nexists e : \delta(q, e) = q_0$.
2. For any node $q \in Q \setminus \{q_0\}$, $|\{q', \exists e, \delta(q', e) = q\}| = 1$ and $\exists w \in \Sigma^*, \delta^*(q_0, w) = q$ (q always has one predecessor q' and is reachable from q_0: $LgT(L)$ is a tree.)
3. For any log $l = (o = (e_1 \dots e_n), N)$, there is a transition sequence such that $\exists q_n \in Q, q_n = \delta^*(q_0, o)$.
4. For any log $l = (o = (e_1 \dots e_n), f)$, let $o_1.e$ be the minimal prefix of o ($o = o_1 e o_2$) such that $o_1.e$ is not the prefix of any other log $l' = (o'', f'), f' \neq f$ or $l' = (o'', N)$, then there is a sequence $q_{o_1} = \delta^*(q_0, o_1)$ followed by $q_o = \delta^*(q_{o_1}, f e o_2)$.

Consider for example, the following log set L: $(cccd, N)$, (aba, f_1), (aba, N), $(ccae, f_2)$, $(dbbdb, N)$, $(abebda, f_1)$, $(ababcc, f_2)$, its log tree is represented in Figure 1.

Any normal log is associated with a unique path from the root node of the tree. Any fault log, like $(abebda, f_1)$ is also represented with a unique path. In the example, the prefix ab is the longest prefix of $abebda$ that is common with a log of another class (normal log aba), which is why the log $abebda$ is represented by the path abf_1ebda. Note also that in this example the log (aba, f_1) is represented by the same path of the log (aba, N) as the longest prefix of aba common to the modes f_1 and N is the sequence itself.

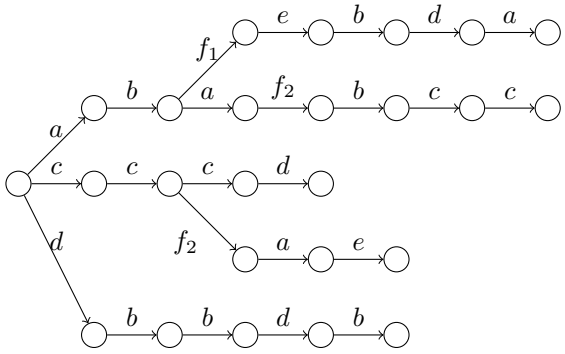

Figure 1: Log tree of $L = \{(cccd, N), (aba, f_1), (aba, N), (ccae, f_2), (dbbdb, N), (abebda, f_1), (ababcc, f_2)\}$.

Proposition 3.1. *For any set of logs L, the log-tree $LgT(L)$ is unique.*

Proof. By construction. □

Based on the log-tree $LgT(L)$, for any fault f present in the log L we can define the language $\mathcal{L}_{L,f}$ as the set of words $e_1, \dots, e_n$ represented by a path $e_1, \dots, e_{i-1}, f, e_i, \dots, e_n$ from q_0. Any word of $\mathcal{L}_{L,f}$ is thus a sequence of observations that *always* results in the diagnosis of fault f according to the log L. Back to Figure 1, $\mathcal{L}_{L,f_1} = \{abe, abeb, abebd, abebda\}$. Note that aba is not part of $\mathcal{L}_{L,f_1}$ even if the logs of L contain (aba, f_1) as (aba, N) is also part of L. The sequence aba leads to an ambiguity according to L.

3.2 Theory of sub-observations

Our objective now is to generate fault signatures from the log-tree $LgT(L)$, that gathers a subset of traces of the system. We propose updating the theory of critical observations introduced in [11] to infer fault signatures as critical observations. Before the notion of critical observations can be introduced, we first define the idea of a *sub-observation*.

A sub-observation, in a practical sense, is intended as a relaxation of the information given in some trace o of the system. Sub-observations allow for the abstraction ('hiding') of events such that only the most relevant information is present. In the following we distinguish between 'hard' and 'soft' events — A *hard event* is a singleton observable event, $x \in \Sigma_o$, and represents the firm occurrence of an event, and a *soft event* is a subset of observable events, $y \subseteq \Sigma_o$, that any number (incl. zero) of which may have occurred along with any number of unobservable events (Σ_u), and these events may occur multiple times.

Definition 4 (Sub-observation)**.** *A sub-observation, θ, is an abstraction over a trace that represents an intentional relaxation of the concrete knowledge in the trace, and is a strict time-ordered alternating sequence of soft and hard events, commencing and ending with a soft event: $\theta = y_0 x_1 y_1 \dots x_n y_n$.*

A sub-observation θ implicitly defines, and thus can be expressed as, a language over the alphabet $\Sigma = \Sigma_o \cup \Sigma_u$:

$$\mathcal{L}_\theta = (y_0 \cup \Sigma_u)^* x_1 (y_1 \cup \Sigma_u)^* \dots x_n (y_n \cup \Sigma_u)^*.$$

A given trace o of the system can be abstracted in several manners (choices between hard and soft events), so a trace

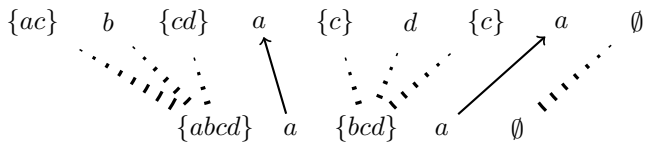

Figure 2: Two sub-observations with a map m satisfying $\preceq$

o is associated with a space of sub-observations, denoted $\mathbb{O}(o)$. Now, if we consider the set of possible traces of Σ_o^*, it can be associated with the global space of sub-observations:

$$\mathbb{O} = \bigcup_{o \in \Sigma_o^*} \mathbb{O}(o).$$

Into the space $\mathbb{O}$, we inject any trace o to a most refined sub-observation, denoted $sub(o)$ where the hard events are the events of o and the soft events are empty.

Definition 5. *The function $sub(\cdot)$ generates the most-refined (or least abstract) sub-observation in $\mathbb{O}(o)$ from a given trace o, by inserting empty soft events at either end of the trace, and between hard events:*

$$o = e_1 \dots e_n$$
$$sub(o) = \emptyset e_1 \emptyset \dots e_n \emptyset \in \mathbb{O}(o) \subseteq \mathbb{O}$$

Space $\mathbb{O}$ gathers the set of all possible sub-observations that are more or less abstracted. We equip $\mathbb{O}$ with a partial order, denoted $\preceq$, to allow us to distinguish whether a given sub-observation is 'more' or 'less' abstract than some other sub-observation. $\preceq$ is defined as follows.

Definition 6. *The relation $\preceq$ over $\mathbb{O}$ is defined such that $\theta' \preceq \theta$ if and only if there exists a mapping function m:*

$$\begin{aligned}
&\text{Given } |\theta'| = n, |\theta| = n' \\
&m : \{0, \dots, n+1\} \to \{0, \dots, n'+1\} \text{ such that} \\
&m(i) < m(i+1),\ m(0) = 0,\ m(n+1) = n'+1 \\
&x'_i = x_{m(i)} \\
&y'_i \supseteq \bigcup_{m(i) \le j \le m(i+1)-1} y_j \cup \bigcup_{m(i) < j < m(i+1)} x_j
\end{aligned}$$

The relation $\preceq$ is provably a partial order over $\mathbb{O}$.

In words: $\theta' \preceq \theta$ if there exists some m that maps the hard events in θ' to an equivalent sequence in θ, retaining the ordering of both, and each y'_i in θ' captures the union of all intervening events – y_j (inclusive) and x_j (exclusive), for j ranging between $m(i)$ and $m(i+1)-1$. As an example, take $\theta = \{ac\}\, b\, \{cd\}\, a\, \{c\}\, d\, \{c\}\, a\emptyset$ and $\theta' = \{abcd\}\, a\, \{bcd\}\, a\emptyset$. The hard events in θ' are matched to x_2 and x_4 in θ, and each y'_i 'consumes' the other information. Specifically, $m(1) = 2, m(2) = 4$, satisfies the constraints for $\theta' \preceq \theta$. We illustrate this in Figure 2. One can read θ' as a regular expression: $[abcd\Sigma_u]^*(a)[bcd\Sigma_u]^*(a)$.

The fundamental principle of a sub-observation θ is that it implicitly represents the set of traces for which it is a more abstract form of:

$$\psi(\theta) = \{o \in \Sigma_o^* \mid \theta \preceq sub(o)\}$$

Therefore, $\theta' \preceq \theta \Rightarrow \psi(\theta') \supseteq \psi(\theta)$. To summarise, the notion of sub-observations leads to the definition of a formal abstraction space:

Definition 7. *The set of sub-observations over the observable alphabet Σ_o is a partial-order set $\langle \mathbb{O}, \preceq, sub \rangle$.*

3.3 Critical observations and log-based diagnosis

In [11], critical observations are computed with respect to the model of the system $\mathcal{S}$. Given a trace o, a critical observation is a sub-observation θ that is precise enough to infer the original diagnosis of o given $\mathcal{S}$ and that is also maximally abstracted. That is, the sub-observation θ cannot further be abstracted without compromising the known diagnosis of the original observation o.

In this paper, we take $\mathcal{S}$ to be unknown, with $LgT(L)$ available as a partial model of the system. We now adapt the definitions of sufficiency and criticality in [11].

Definition 8. *The diagnoses of a sub-observation θ is the union of the diagnoses of the fault traces of the logs L for which θ is the more abstract form of:*

$$\begin{aligned}
\Delta_L(\theta) &= \bigcup_{o \in \left(\psi(\theta) \cap \bigcup_{f \in L} \mathcal{L}_{L,f}\right)} \Delta_L(o) \\
&= \bigcup_{o \in \left(\psi(\theta) \cap \bigcup_{f \in L} \mathcal{L}_{L,f}\right)} \{f\}
\end{aligned}$$

Therefore, if $\theta \preceq sub(o)$ then $\delta \in \Delta_L(\theta)$ if $\delta \in \Delta_L(o)$.

Definition 9 (Sufficient)**.** *Given a log $o \in \mathcal{L}_{L,f}$, a sub-observation $\theta \preceq sub(o)$ is* sufficient *for o if $\Delta(\theta) = \{f\}$.*

Sufficiency, then, is the formalisation of the property that information lost to abstraction does not affect the known diagnosis by making other potential diagnoses feasible (that is, others logs $l' \in L$ with $f \neq f'$ are not captured by θ):

$$\Delta_L(\theta) \setminus \{f\} = \varnothing \tag{3}$$

Observe that by Definitions 7, and 8, the naïve sub-observation, $sub(o)$, satisfies the criteria to be sufficient for o, and means that at least one solution can be found:

$$\begin{aligned}
\Delta_L(sub(o)) &= \bigcup_{o \in \left(\psi(sub(o)) \cap \bigcup_{f \in L} \mathcal{L}_{L,f}\right)} \Delta_L(o) \\
&= \Delta_L(o) = \{f\}
\end{aligned}$$

We are then left to search for a 'maximally abstract', or *critical*, sub-observation, and define that as follows:

Definition 10 (Critical)**.** *Given a log $o \in \mathcal{L}_{L,f}$, a sub-observation $\theta \preceq sub(o)$ is* critical *for o if it is sufficient for o and there is no strict sub-observation of θ that is also sufficient:*

$$\forall \theta' \preceq \theta \text{ if } (\Delta_L(\theta') = \{f\}) \text{ then } (\theta' = \theta)\,. \tag{4}$$

Noting that $\preceq$ is only a partial order, it is possible that there could be several critical sub-observations. Back to Definition 2, the full log-based diagnoser Δ_L can then be defined as follows. Let $\Theta = \{\theta_1, \dots, \theta_m\}$ be the set of critical observations that have been inferred, for any further sequence o produced by the system, Δ_L returns:

$$\Delta_L(o) = \bigcup_{\theta \in \Theta, o \in \psi(\theta)} \Delta_L(\theta).$$

The log-based diagnoser Δ_L may not be accurate as it is based on a partial knowledge of the underlying system. For a given new trace o, it returns a set of diagnoses $\delta \in \Delta_L(o)$ based on the fact there exist some logs o' with $\theta \preceq sub(o')$ and $\theta \preceq sub(o)$ for which δ is certainly the diagnosis of o' according the available set of logs L.

Procedure FINDCRITICALOBSERVATIONS
input: logtree $LgT(L)$
input: log $o \in \mathcal{L}(L, f)$
output: critical observation
diag $:= f$; $\theta := sub(o)$
candidates := *children*(θ)
while *candidates* $\neq \emptyset$ **do**
  $\theta' :=$ pop(*candidates*)
  if $\Delta_L(\theta') =$ *diag* **then**
    $\theta := \theta'$
    candidates := *children*(θ)
  end if
end while
return θ

Figure 3: Finding a critical observation

Theorem 3.2. *For any fault f, Δ_L converges to the model-based diagnoser Δ with respect to the size of L:*

$$(\exists o \in \Sigma_o^*, \{f\} = \Delta(o)) \Rightarrow (\lim_{|L|\to\infty} \Delta_L(o) = \{f\})$$

Proof. (sketch) By convergence, we mean here that the larger L is, the more likely $\Delta_L(o) = \Delta(o) = \{f\}$. As L gets bigger, it captures more and more behaviours from the underlying system $\mathcal{S}$. If $\mathcal{S}$ is such that $\exists o \in \Sigma_o^*, \{f\} = \Delta(o)$ (i.e. the fault f can be diagnosed with certainty by observing o), then it is guaranteed, if L is big enough, that Δ_L will know a critical observation θ such that $\theta \preceq sub(o)$ a nd $\Delta_L(\theta) = \{f\}$ and no other available critical observations $\theta' \neq \theta$ will be such that $o \in \psi(\theta')$. □

3.4 Inferring the Critical Observations Set

The remaining question is now how to infer a set of critical observations from $LgT(L)$. From [11] we also take two other results:

1. The finiteness of the set of sub-observations $\mathbb{O}(o)$, and
2. The monotonicity of Definition 9 – given θ_1, θ_2 such that $\theta_1 \preceq \theta_2 \preceq sub(o)$, if θ_1 is sufficient for o, then so is θ_2.

These together grant there are no unreachable sufficient sub-observations, and that at least one critical sub-observation exists at a finite depth, k, of abstraction away from $sub(o)$.

To define a valid search space for algorithmic use, we must first define explicitly the children (successors) of a sub-observation:

Definition 11. *A* child *of sub-observation θ is a strict sub-observation θ' of θ such that no sub-observation sits 'between' θ' and θ.*

$$\theta' \in children(\theta) \iff (\theta' \prec \theta) \wedge (\nexists \theta'' \in \mathbb{O}.\ \theta' \prec \theta'' \prec \theta)\,.$$

This provides a practical characterisation of criticality: a sufficient sub-observation θ is critical if and only if none of its children are sufficient. This then provides all the conditions necessary to define a terminating search algorithm (such as that given in [11], along with child generation, which as been adapted for this setting) presented in Figure 3.

We can now create the diagnoser Δ_L that makes use of a set of critical observations, Θ, derived from $LgT(L)$. We construct Θ as follows. Taking L, construct the log tree $LgT(L)$ as defined in § 3.1. Then compute the critical observations $\{\theta_o\}$ for each trace with a fault mode that is not nominal: $o \in \bigcup_{f \neq N} \mathcal{L}_{L,f}$, and set $\Theta = \bigcup_o \{\theta_o\}$.

By further extending Algorithm 3 to maintain all currently sufficient children in the queue, and to save all valid critical observations for a given trace rather than just the first, we can guarantee a complete set as L approaches the underlying model ($L \to L_\infty \approx \mathcal{S}$). This extension does not ultimately affect the complexity of the algorithm, as it is equivalent to the worst-case exhaustive search under the greedy approach. In [11] it was demonstrated that the equivalent of Algorithm 3 had complexity $O(n^2m^2)$ in the number of calls to the model based diagnoser Δ. We provably retain the same bound but in calls to $\Delta_L(\cdot)$ at worse for each trace, and can be dramatically improved with heuristics that allow us to recall and then avoid the types of abstraction that caused pruning earlier in the search.

As is typical in diagnosis problems we often prefer *minimal* diagnoses (with respect to some notion of minimality: cardinality, set inclusion, etc.). Whether this type of minimality property can be extended to Θ is an open question, as without computing the full set of critical observations for any given observation, the completeness of Θ can not necessarily be guaranteed.

In particular, consider the following scenario: l_1 and l_2 are both associated with the same fault mode f, but produce different critical observations, θ_1 and θ_2 such that $(\theta_1 \not\preceq \theta_2) \wedge (\theta_2 \not\preceq \theta_1)$, with $o_1, o_2 \in \psi(\theta_1)$ and $o_2 \in \psi(\theta_2)$. Under a set-inclusion minimality, we would remove θ_2 from Θ as it would be seen as superfluous. Suppose then that logs l_3 and l_4 which both matched θ_2 are queried on Δ_L. As $\Theta = \theta_1$ in this case unfortunately we provide an inaccurate diagnosis.

4 Online inference of the critical observations

We now discuss the problem of inferring an update of the set of critical observations. Given a log tree $LgT(L)$ and a new available log $l = (o, f)$, the objective is to update the set of critical observations to keep the consistency between the diagnoser Δ_L and its current new knowledge. In other words, if L' denotes $L \cup \{l\}$, the objective of the critical observation update is to actually update Δ_L to $\Delta_{L'}$ (see Definition 2).

To intuitively explain the objective of the incremental method that is proposed here, consider that the diagnoser Δ_L is available (relying for instance on the method that is detailed in Section 3) and in charge of analysing the new sequence of observations o (that is part of the new available log $l = (o, f)$).

The diagnoser Δ_L might simply return $\varnothing$, in this case it means that the sequence o belongs to a type of observable sequences that is unknown from the log L or ambiguous. It might also return a diagnosis $\delta = f'$ due to the fact that Δ_L *knows* some critical observations that are associated with fault f'. What happens now if the tag of the new log is not f'? Based on the current knowledge, the set of critical observations of Δ_L must be revised. In this example, the critical observation of Δ_L leading to an inconsistency with respect to the new log l is either too abstracted (not only it captures behaviours of fault mode f, but also it captures newly discovered behaviours of fault mode f') or simply removed (diagnosability issues between fault mode f' and fault mode f, the system generates the same observable behaviour for both modes).

To perform online inference of critical observations, we first update the log-tree from $LgT(L)$ to the new log tree $LgT\left(L \cup \{(o, f)\}\right)$ (see for instance the update of the log-

tree from Figure 1 to Figure 5). After updating the log tree we can proceed to updating Θ, the current set of critical observations. First observe that the critical observations in Θ can never be made more abstract in the presence of new logs.

Theorem 4.1 (Most Abstract)**.** *Each $\theta \in \Theta$ is maximally abstract and cannot be made more abstract.*
Formally, $\forall \theta \in \Theta$ that matches some subset of the logs of a class f, $\lambda \subseteq \mathcal{L}_{L,f}$, θ cannot be made more abstract. That is, if $\exists \theta' \preceq \theta$ such that $\forall (o, f) \in \lambda, (\theta' \preceq sub(o)) \wedge (\Delta(\theta') = f)$, then $\theta' = \theta$.

Adding new logs can only cause sub-observations to become refined again. By construction, θ is as abstract as possible. Should a new log, $l = (o, t)$, become available that matches a critical observation θ associated with fault mode f, there are two cases to consider:

1. That o is tagged with the fault class of θ, $t = f$
2. That o is tagged with a different class, $t \neq f$.

Lemma 4.2 (Sketch)**.** *If $l = (o, t)$ is tagged with the same fault class of θ ($t = f$), then θ remains critical. Assume there exists a sufficient $\theta' \prec \theta_f$ matching l and all prior logs $l' \in \lambda$. However, θ is by construction critical (no sufficient children) prior to the update and matches l, inducing a contradiction. Therefore no such θ' exists, as if it did then $\theta = \theta'$ before the update, and θ_f maintains criticality.*

Lemma 4.3 (Sketch)**.** *If $l = (o, t)$ has a different class tag to θ, $t \neq f$, then it is the case that θ is no longer sufficient by definition 9. Assume there exists some sufficient $\theta' \preceq \theta$. However, by monotonicity, θ is therefore sufficient, inducing a contradiction. Therefore no such θ' exists, and some less abstract parent of θ is the nearest sufficient sub-observation.*

Proof of Theorem 4.1. Lemmas 4.2 and 4.3 together show that Θ is maximally abstract at all times, and can only be made more concrete when correcting for a conflict. □

Given Theorem 4.1, we know that repairing Θ involves the finding of less abstract sub-observations. However, backtracking in this search space is not trivial, as for any given sub-observation θ there is no unique parent (the successor function $children(\theta)$ is not invertible).

There are two possible solutions to this problem. The naïve approach involves the recomputation of all conflicting critical observations, restarting the computation of these observations from the top (at $sub(o)$ for all corresponding traces). This approach is quite expensive however, and ignores the works already done in finding the prior result. The alternative is to remember the search path through $\mathbb{O}$, trading off the time-cost for a space-cost (linear in the size of Θ).

For each θ' thrown into the contention by the new log l, we backtrack all stored search paths by undoing abstractions until the sub-observation is sufficient again, and resume the abstraction process from this new point, storing any new critical observations found and associated search paths for future use.

We make use of an algorithm similar to that discussed in § 3.4 for the search resumption, and call this combined procedure $upd(\Theta, \{l\}) = \Theta'$. By using this approach we can guarantee that iteratively updating with new logs over time in this way allows us to converge to the same Θ_{L_∞}, and therefore the same diagnoser Δ_{L_∞}, that would be produced had all logs been available.

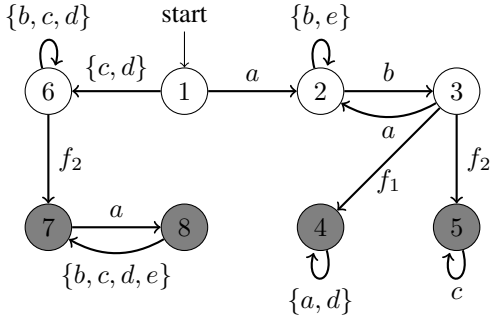

Figure 4: Underlying system of the illustrative example.

Theorem 4.4 (Convergence of Θ)**.** *For any set of logs L, the critical observation set Θ_L converges to Θ_{L_∞} independent of the order of the updates taking L to L_∞. That is, given L and two updates L_1, L_2 with $L_1 \neq L_2$ we have:*

$$upd(upd(\Theta_L, L_1), L_2) = upd(upd(\Theta_L, L_2), L_1)$$

Proof (sketch). For all Θ and any possible update L', collect all $\theta \in \Theta$ that are no longer sufficient (i.e. conflict with L'), and call this set Θ_c. By Theorem 4.1 and the monotonicity of $\mathbb{O}$, we say without loss of generality that either:

1. $\forall \theta_c \in \Theta_c$ there exists some chain of ancestors $\theta'_1, \ldots, \theta'_k$ (possibly zero length) with respect to each trace connected to θ_c, that are not sufficient, ending in some sufficient θ_a such that $\theta_c \preceq \theta'_1 \preceq \ldots \preceq \theta'_k \preceq \theta_a$; or
2. There is no sufficient ancestor.

For the first—in all conflict cases where a sufficient (not necessarily critical) θ_a can be found after any update, that ancestor is a at worst a sufficient sub-observation, and by monotonicity, any $\theta \in \Theta_{L_\infty}$ in the final state must either be one of these ancestors, or descendant on a different branch, which must have been located by the update process, by construction.

For the second—in conflict cases where θ_c cannot be repaired with respect to any matching trace o and is removed, the conflicting logs are by definition still present in L_∞. Whether these conflicting logs are nominal, or tagged with a different mode, they necessarily still supersede in the limit as the size of L approaches infinity, and the conflicting θ_c nor any ancestor exist in Θ_{L_∞}.

The remaining case for consideration is that where the new log fails to match on any $\theta \in \Theta$. In this instance, as there is no updating required, we simply compute the critical observations for this new log and update as per the above. □

5 Illustrative example

We present an illustrative example demonstrating how the fault signatures are inferred from a set of logs. For the sake of illustration, Figure 4 shows the underlying system that generates the described logs. In particular, it must be noticed that this system can generate the set of logs used to construct the log tree in Figure 1. Taking the logs used to build Figure 1, we consider the faulty logs in L to compute an initial Θ as follows:

1. Consider (aba, f_1). In this case, as aba is also a nominal trace, we cannot produce a critical observation that produces f_1, and as such pass over it.

2. Consider $(ccae, f_2)$. We search $\mathbb{O}$ w.r.t. o, and find two critical sub-observations, which are added to Θ:

$$\theta_1 = \{\Sigma_o\}\, c\, \{\Sigma_o\}\, a\, \{\Sigma_o\} \quad \theta_2 = \{\Sigma_o\}\, c\, \{\Sigma_o\}\, e\, \{\Sigma_o\}$$

3. Consider $(abebda, f_1)$. From this log we add many critical sub-observations to Θ:

$$\theta_3 = \{\Sigma_o\}\, a\, \{\Sigma_o\}\, d\, \{\Sigma_o\} \quad \theta_4 = \{\Sigma_o\}\, b\, \{\Sigma_o\}\, e\, \{\Sigma_o\}$$
$$\theta_5 = \{\Sigma_o\}\, d\, \{\Sigma_o\}\, a\, \{\Sigma_o\} \quad \theta_6 = \{\Sigma_o\}\, e\, \{\Sigma_o\}\, b\, \{\Sigma_o\}$$
$$\theta_7 = \{\Sigma_o\}\, e\, \{\Sigma_o\}\, a\, \{\Sigma_o\} \quad \theta_8 = \{\Sigma_o\}\, e\, \{\Sigma_o\}\, d\, \{\Sigma_o\}$$

Note that at this stage we are generating all sequential pairs that appear uniquely in the trace in question.

4. Consider $(ababcc, f_2)$. From this log we get two further critical observations:

$$\theta_9 = \{\Sigma_o\}\, a\, \{\Sigma_o\}\, c\, \{\Sigma_o\} \quad \theta_{10} = \{\Sigma_o\}\, b\, \{\Sigma_o\}\, c\, \{\Sigma_o\}$$

Also it is of interest to note that every soft event is the trivial Σ_o. Both this and the pairing phenomenon are brought about because we as of yet lack of lot of information about the underlying system, and indeed many of these are actually not sufficient in 'reality'. As more logs are added we would expect some of these to become sequential triples first, in addition to seeing non-trivial soft events.

The diagnoser Δ_L then starts with $\Theta = \{\theta_1, \ldots, \theta_{10}\}$ as the recogniser. Suppose now that Δ_L is in use and the system produces a trace like $cccdabac$: Δ_L then recognises θ_1, θ_5, θ_9, θ_{10} so the diagnosis of $cccdabac$ by Δ_L is $\{f_1, f_2\}$.

Now we show the incremental update process. Suppose a new log, $(dbbdad, f_2)$, is added to L. Should this log not have been tagged as f_2, Δ_L would return f_1 as a diagnosis due to the match against both θ_3 and θ_5. However, as we do have a fault mode, Θ can be revised and refined.

Firstly, we determine whether this trace is recognised, and if so, if it is correctly identified—here we see matches on patterns meant to recognise f_1. Therefore θ_3 and θ_5 are no longer sufficient and too abstract by Theorem 4.1. Upon updating $LgT(L)$, we now consider these in turn.

We reverse the relaxation of θ_3 until we reach a sufficient sub-observation. One step backwards we find one: $\{\Sigma_o\}\, a\, \{\Sigma_o\}\, b\, \{\Sigma_o\}\, d\, \{\Sigma_o\}$, where $\{\Sigma_o\}\, b\, \{\Sigma_o\}$ had been collapsed in our original search. We resume abstracting at this point, and find that none of the children of this are sufficient, and so it becomes the new critical sub-observation θ_3'. We similarly consider θ_5, and find that two steps backwards are required for sufficiency, arriving at $\{\Sigma_o\}\, e\, \{\Sigma_o\}\, b\, \{\Sigma_o\}\, d\, \{\Sigma_o\}\, a\, \{\Sigma_o\}$. Resuming abstraction we find that this sub-observation is already superseded by $\theta_6, \theta_7, \theta_8$, and that all remaining children are not sufficient. This causes θ_5 to be dropped from Θ entirely.

Further, we now search this new log for any new critical observations, and find three new critical observations:

$$\theta_{11} = \{\Sigma_o\}\, b\, \{\Sigma_o\}\, a\, \{\Sigma_o\}\, d\, \{\Sigma_o\}$$
$$\theta_{12} = \{\Sigma_o\}\, b\, \{\Sigma_o\}\, b\, \{\Sigma_o\}\, a\, \{\Sigma_o\}$$
$$\theta_{13} = \{\Sigma_o\}\, d\, \{\Sigma_o\}\, a\, \{\Sigma_o\}\, d\, \{\Sigma_o\}$$

This new set of critical observations then leads to the new diagnoser $\Delta_{L'=L\cup\{(dbbdad,f_2)\}}$. Now, considering again the same trace $cccdabac$ as above: the diagnosis of $\Delta_{L'}$ becomes only f_2.

Suppose now that $(aebeba, N)$ is added to L'. This particular trace matches $\theta_4, \theta_6, \theta_7$, and θ_{12}. It would be diagnosed as $\{f_1, f_2\}$ were this not an update, as it matched some θ for both fault modes. Since we have conflicting tags for several different critical observations, each must be updated.

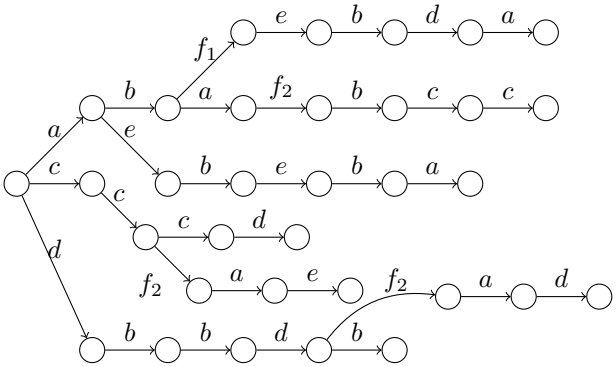

Figure 5: Update Log tree of L, with $(dbbdad, f_2)$ and $(aebeba, N)$ added.

We start with θ_4. We backtrack until we find a sufficient ancestor in $\{\Sigma_o\}\, b\, \{\Sigma_o\}\, e\, \{\Sigma_o\}\, d\, \{\Sigma_o\}$, and then abstracts to θ_8 with no other sub-observations being sufficient, allowing us to drop θ_4 entirely. Considering θ_6 a similar thing happens. It is reversed to $\{\Sigma_o\}\, e\, \{\Sigma_o\}\, b\, \{\Sigma_o\}\, d\, \{\Sigma_o\}$ which is similarly caught by θ_8, and has no other sufficient children. Consider now θ_7, similarly to θ_4 and θ_6, we end up at θ_8 via the nearest sufficient ancestor $\{\Sigma_o\}\, e\, \{\Sigma_o\}\, d\, \{\Sigma_o\}\, a\, \{\Sigma_o\}$, and consequently θ_7 is also dropped. Lastly, consider θ_{12}. In this case we step back to $\{\Sigma_o\}\, b\, \{\Sigma_o\}\, b\, \{\Sigma_o\}\, a\, \{\Sigma_o\}\, d\, \{\Sigma_o\}$, and discover that only one sufficient child exists, and is equal to θ_{11}.

The net result of the addition of this single log was the removal of four incorrect, then redundant, critical observations. The resulting state of the log tree is shown in Figure 5.

6 Related work

Fault diagnosis in discrete event systems is a problem that has been intensively investigated. This problem was introduced in [13; 14] and a recent survey can be found in [7]. In these contributions, the knowledge of the system is complete (*white-box models*), and the diagnosis methods consist usually in compiling diagnosis information into finite-state machines that are able to provide diagnosis updates after the occurrence of a new observed event. The main issue we face with this particular approach is the combinatorial explosion of the diagnoser. One approach to this problem is to compute abstractions that make the resulting diagnoser less complex while maintaining similar accuracy to the non-abstracted version [15; 16]. The theory of critical observations provides another way to perform these abstractions [11]. Then, by construction, a diagnoser that is based on critical observations simply becomes a pattern recogniser similar in nature to the one presented in [10].

Solving the fault diagnosis problem without any explicit model of the system requires the use of learning techniques. Our proposal can be considered a type of supervised learning technique closer in style to grammatical inference [17], which mainly tries to identify the underlying system by learning automata [18]. We do not attempt to learn the model of the system [19], but rather a set of concise representations of the fault signatures such as proposed in [8; 20] in a unsupervised learning context (classification).

7 Conclusion

In this work, we present an original approach for fault diagnosis in discrete event systems. In contrast to many of the approaches that deal with this problem, our proposal does not rely on the knowledge of a model but only on an available set of tagged logs. By using critical observations, we propose generating fault signatures based on the given logs that are as abstract as possible but are guaranteed to not contradict the available information about the system.

Based on a given log, the diagnoser can diagnose any future trace to be faulty if it is certain it is similar to one that is known to produce the same fault according to the current set of tagged logs. Using critical observations serves two goals: to generalise by abstraction the available knowledge about the system (knowledge inference), and to represent fault signatures as a set of hard/soft events which require only the implementation of a recogniser that performs diagnosis in a similar fashion to a chronicle recogniser (computational performance). As new logs may become available, fault signatures are revised. This revision of the fault signatures always consists of a refinement of the set of critical observations, making the diagnoser progressively more precise when new logs are available.

The proposed approach is monolithic and purely data-driven, however, it might be interesting to consider as an extension that we know some of the structural model of the system (a set of interacting, possibly specified, components) making it possible to design a distributed set of log-based diagnosers. Another avenue would be to start from untagged logs and to use the generation of critical observations as a way to generate the fault class of the underlying system. Lastly, other techniques from grammatical inference and automata learning could be used to improve the structure of the log-tree, in particular for identifying loops that are present in the underlying system.

References

[1] Yannick Pencolé and Marie-Odile Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164(2):121–170, 5 2005.

[2] Yassine Qamsane, Abdelouahed Tajer, and Alexandre Philippot. Distributed supervisory control synthesis for discrete manufacturing systems. In *8th IFAC Conference on Manufacturing Modelling, Management and Control*, Troyes, France, June 2016.

[3] Luca Console et al. WS-DIAMOND: Web services: Diagnosability, monitoring, and diagnosis. In *At Your Service: Service-Oriented Computing from an EU Perspective*, pages 213–240. MIT PRESS, 2008.

[4] Thierry Jéron, Hervé Marchand, Sophie Pinchinat, and Marie-Odile Cordier. Supervision patterns in discrete event systems diagnosis. In *8th International Workshop on Discrete Event Systems*, pages 262–268, Ann Arbor, MI, United States, 6 2006.

[5] Houssam-Eddine Gougam, Yannick Pencolé, and Audine Subias. Diagnosability analysis of patterns on bounded labeled prioritized Petri nets. *Discrete Event Dynamic Systems*, 27(1):143–180, 2017.

[6] M. Zanella and G. Lamperti. *Diagnosis of active systems*. Kluwer Academic Publishers, 2003.

[7] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37:308–320, 2013.

[8] Françoise Fessant and Fabrice Clérot. An efficient som-based pre-processing to improve the discovery of frequent patterns in alarm logs. In *DMIN*, pages 276–282, 2006.

[9] M. O. Cordier, L. Travé-Massuyès, and X. Pucel. Comparing diagnosability in continuous and discrete-event systems. In *Proceedings of the 17th International Workshop on Principles of Diagnosis (DX-06)*, pages 55–60, 2006.

[10] Christophe Dousson. Extending and unifying chronicle representation with event counters. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, pages 257–261, 2002.

[11] C. Christopher and Al. Grastien. Formulating event-based critical observations in diagnostic problems. In *54th IEEE Conference on Decision and Control (CDC-15)*, pages 4462–4467, 2015.

[12] C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publishers, 1999.

[13] Feng Lin. Diagnosability of discrete event systems and its applications. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 4(2):197–212, 5 1994.

[14] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *Transactions on Automatic Control*, 40(9):1555–1575, 9 1995.

[15] B. Guerraz and C. Dousson. Chronicles construction starting from the fault model of the system to diagnose. In *International Workshop on Principles of Diagnosis (DX04)*, pages 51–56, Carcassonne, France, 2004.

[16] Anika Schumann and Yannick Pencolé. Efficient online failure identification for discrete-event systems. In *6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, pages 1294–1299, Beijing, China, 8 2006.

[17] Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[18] M. A.L. Thathachar and P. S. Sastry. Varieties of learning automata: An overview. *Trans. Sys. Man Cyber. Part B*, 32(6):711–722, December 2002.

[19] Alexander Maier, Oliver Niggeman, and Jens Eickmeyer. On the learning of timing behavior for anomaly detection in cyber-physical production systems. In *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-15)*, pages 217–224, Paris,France, 2015.

[20] A. Subias, L. Travé-Massuyès, and E. Le Corronc. Learning chronicles signing multiple scenario instances. In *25th international workshop of diagnosis (DX14)*, Graz, Autriche, 2014.